UNIVERSITAT DE BARCELONA

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)

UNIVERSITAT DE BARCELONA

- Tutor:

  Dr. Maite López-Sánchez

  University of Barcelona

Universitat de Barcelona

UNIVERSITAT DE 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. student Eva Bou (co-sup.: J.A. Rodríguez- Aguilar, IIIA)
    - PhD. thesis by Jordi Campos (co-sup: Dr. Marc Esteva, IIIA).
    - Ph.D. thesis on on-line norm synthesis by Javier Morales. Co-supervisor: Dr. Juan A. Rodríguez-Aguilar (IIIA-CSIC)
    - Research collaborations: Dr. Jaime S. Sichman (Univ. Sao Paulo), Dr. Wamberto Vasconcelos (Univ. of Abeerdeen), Prof. Michael Wooldridge (Univ. of Oxford).

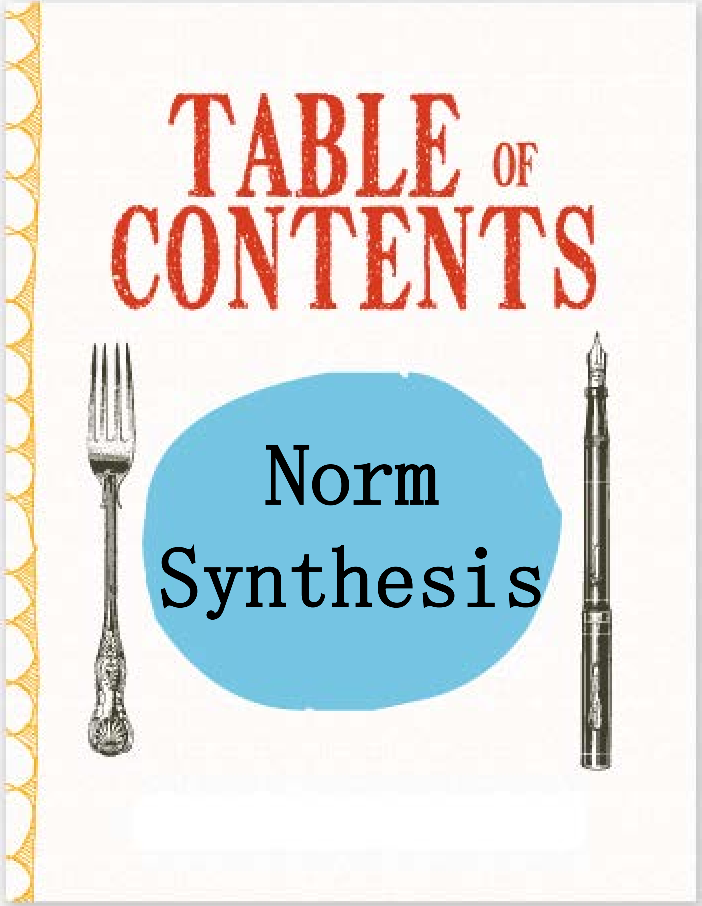
- Tutorial material available online at:
  - Tutorial slides:
    - http://www.maia.ub.es/~maite/Teaching.html
  - On-line Norm Synthesis source code:

http://normsynthesis.github.io/NormLabSimulators/

http://normsynthesis.github.io/NormSynthesisMachine/

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. Overview of approaches to norm synthesis.
3. On-line automatic norm synthesis.
4. Demo and hands-on activity

UNIVERSITAT DE BARCELONA

Schedule

The tutorial will last for 3h:30 min or 4h

- 9:15 h: NMAS introduction
- 9:45 h: Norm synthesis overview (45’ aprox.)
- 10:30 h: Break (15’)
- 10:45 h: On-line automatic norm synthesis
- 11:45 h: Break (15’)
- 12:00 h: Demo and hands-on activity (45’ if interest in playing with the code)
- 12:45 h: Wrap-Up & comments

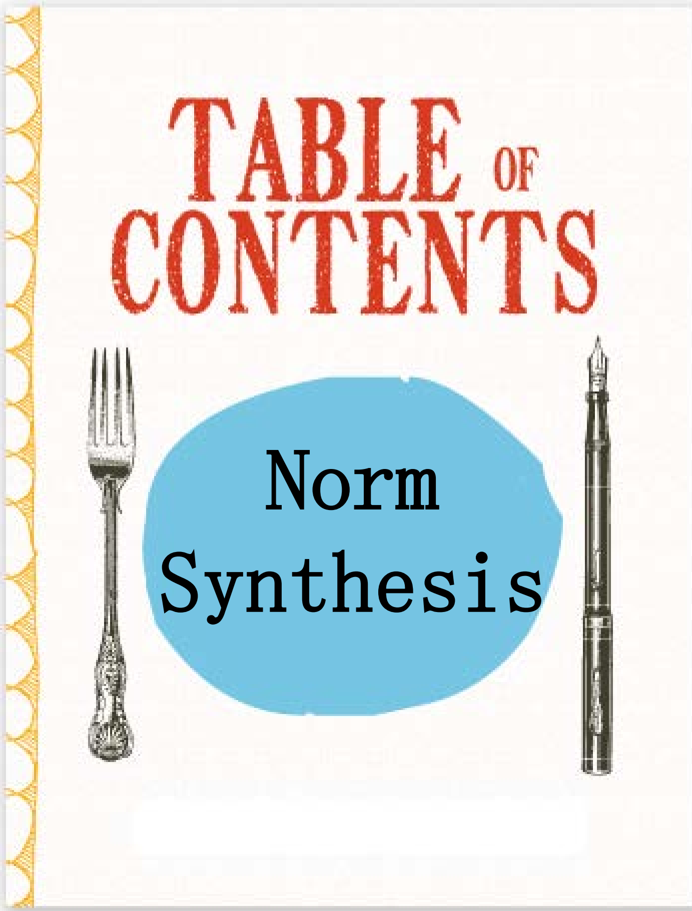
Objectives for the tutorial

- To introduce NMAS
  - MAS Design considerations.
  - Norms, laws, social conventions and rules as agent coordination mechanisms
- Be familiar with a framework to test on-line NMAS
  - On-line automatic norm synthesis
  - Put it in practice

UNIVERSITAT DE BARCELONA

Tutorial Outline

Contents: Modules



1. **Introduction to Norms and Normative MAS**.
2. Overview of approaches to norm synthesis.
3. On-line automatic norm synthesis.
4. Demo and hands-on activity

Norms

Norm definition: Lewis

- 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).

Norms

Norm definition: Merriam-Webster dict.

- Norm definition from Merriam-Webster dictionary:
  - a principle of right action binding upon the members of a group and serving to guide, control, or regulate proper and acceptable behavior
  - a pattern or trait taken to be typical in the behavior of a social group
  - ...

Norms

Norm definition: Britannica encycl.

- Norm definition from Britannica.com:

Norm, also called Social Norm, rule or standard of behaviour shared by members of a social group.

Norms may be **internalized** —*i.e.*, incorporated within the individual so that there is conformity without external rewards or punishments, **or** they may be **enforced** by positive or negative sanctions from without.

The social unit sharing particular norms may be small (*e.g.*, a clique of friends) or may include all adult members of a society.

Norms are more specific than values or ideals: honesty is a general value, but the rules defining what is honest behaviour in a particular situation are norms.

Norms

Norm definition in MAS

- Coordination by norms and social laws, "Introduction to MAS" book by Wooldridge:
  - A norm is an **established, expected pattern of behaviour**.
    - Human example: to form a queue when waiting for a bus
    - Norms may not be enforced
    - Social laws usually carry with them some authority
- Alternative definition:
  - Norms = constraints + punishment

Norms

Norm as a MAS coordination mechanism

- Conventions are key in the social process:
  - Provide agents with a template upon which to structure their action repertoire -> simplify agent's decision-making process
  - **Balance** between:
    - Individual freedom and
    - The goal of the agent society

Norms

Norm origin

- How do norms come to exist within a society?
  - Offline design
  - Emergence
  - On-line generation

Norm categories I

Tuomela

- R. Tuomela:
  - The Importance of Us: A Philosophical Study of Basic Social Notions. Stanford Series in Philosophy, Stanford University Press (1995)
    - Rule norms: imposed by authority based on an agreement between the members (e.g. one has to pay taxes).
    - Social norms: apply to large groups (e.g. one should not litter)
    - Moral norms: appeal to one's conscience (e.g. one should not steal).
    - Prudential norms: based on rationality (e.g one ought to maximize one's expected utility)

Norm categories II & III

(II) Elster (III) Boella & van der Torre

- Norm Categories:
  - 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)…
  - Boella and van der Torre
    - Regulative norms: obligations, prohibitions and permissions
    - Constitutive norms: create institutional facts like property or marriage as well as the modification of normative system itself

Norms & agents

- Norms and BDI agents
  - Normtive decision theory: BOID
  - “Norm-based behaviour modification in BDI agents” Meneguzzi and Luck AAMAS’09
  - Dignum et at.
  - Introducing obligations in agents
- We take a system perspective: NMAS

Normative MAS

http://www.dagstuhl.de/07122

- Normative MAS @Dagstuhl 2007

Normative MAS

http://www.dagstuhl.de/07122

- Normative MAS @Dagstuhl 2007
- A normative multiagent system is a multiagent system together with normative systems in which:
  - **Agents can decide** whether **to follow** the explicitly represented **norms**, and
  - the normative systems specify how and in which extent the agents can **modify the norms**.

UNIVERSITAT DE BARCELONA

Normative MAS

http://www.dagstuhl.de/07122

- Normative MAS @Dagstuhl 2007
  - AAMS journal 2008 by Boella, van der Torre and Verhagen:

    A normative MAS contains mechanisms to:
    - Represent, communicate, distribute, detect, create, modify, and enforce norms.
    - Deliberate about norms and detect norm violation and fulfillment.

Normative MAS

http://www.dagstuhl.de/12111

- Dagstuhl 2012

Normative MAS

Example: Answers for a Traffic scenario?

- Some related questions:
  - Who dictates norms?
  - Who spreads them?
  - What norm does apply to an agent?
  - How the agent decides whether to fulfill or violate it?
  - Who/how detects if the agent complies with it?
  - What are the consequences?
  - Should this norm change?

5 min

- Some related questions:
  - Who dictates norms?
  - Who spreads them?
  - What norm does apply to an agent?
  - How the agent decides whether to fulfill or violate it?
  - Who/how detects if the agent complies with it?
  - What are the consequences?
  - Should this norm change?

Further Questions II

- Further questions:
  - Are norms hierarchical?
  - Are norms local?
  - Are norms imprecise?
  - Do agents internalise (adopt) norms?
  - If other agents do not comply with a norm, should an agent bother about it?
  - Do we need additional incentives? (rewards, environment)

Further Questions III

- Further questions:
  - Should we consider norm exceptions?
  - How norms relate to organisations? (modularity, abstractions)
  - How do we design norms? (to coordinate, organize, guide, regulate, or control interaction)
  - Norm representation (logics, operational,..)
  - Can we have conflicts between norms?

Normative MAS

http://www.dagstuhl.de/15131

- Normative MAS @Dagstuhl 2015
  - Normative systems are systems in the behavior of which norms play a role and which need normative concepts in order to be described or specified.
    - **deal with obligations, permissions and prohibitions**
  - A normative MAS combines models for normative systems with models for MAS. […]
    - **They use sociological theories from sociology, economics, legal science, etc.**

Norms and Philosophy

UNIVERSITAT DE BARCELONA

- Deontic Logic (DL):
  - Despite the philosophical position that norms are neither true nor false
  - Obligations treated as goals in AI
  - NORMAS: action and time DL
- Two distinct philosophical traditions:

J. Hansen. Imperatives and Deontic Logic. PhD thesis, University of Leipzig, 2008

  - Von Wright: norms and normative propositions
  - Alchourron: prescriptive and descriptive obligations

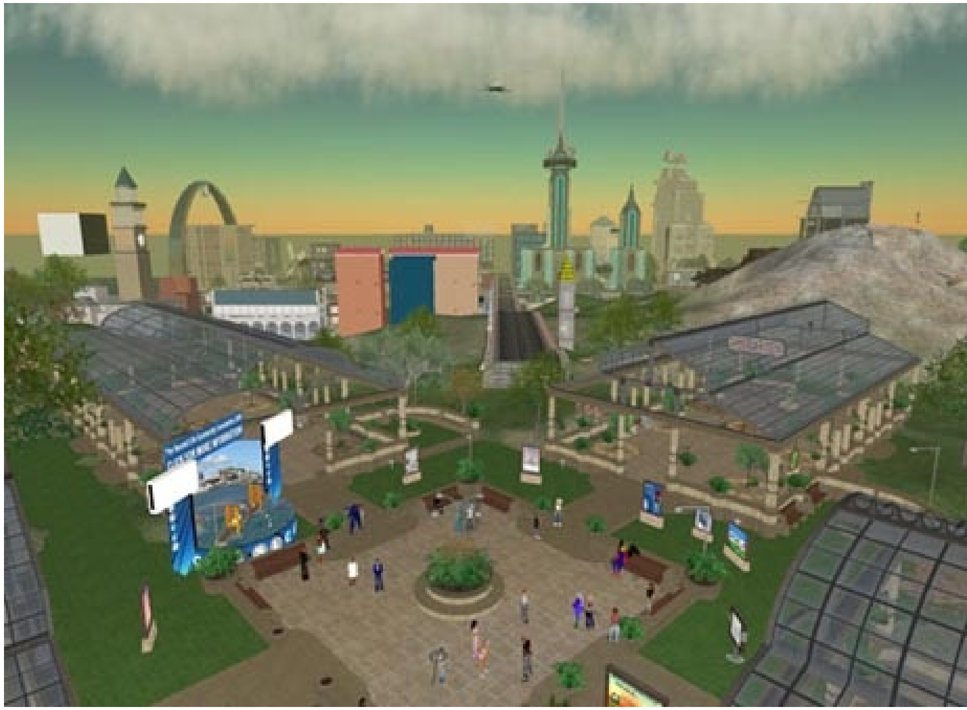
Norms and game theory

Further issues

- Norms as a mechanism in a game-theoretic setting:
  - D. Lewis “master and slave” game
  - E. Bulygin “rex, minister and subject” game
  - G. Boella c.s.: violation games, institutionalized games, negotiation games, norm creation games, control games
  - DTGT vs DL (van der Torre):
    - DTGT: each agent has its own utility
    - DL: there is a single global utility

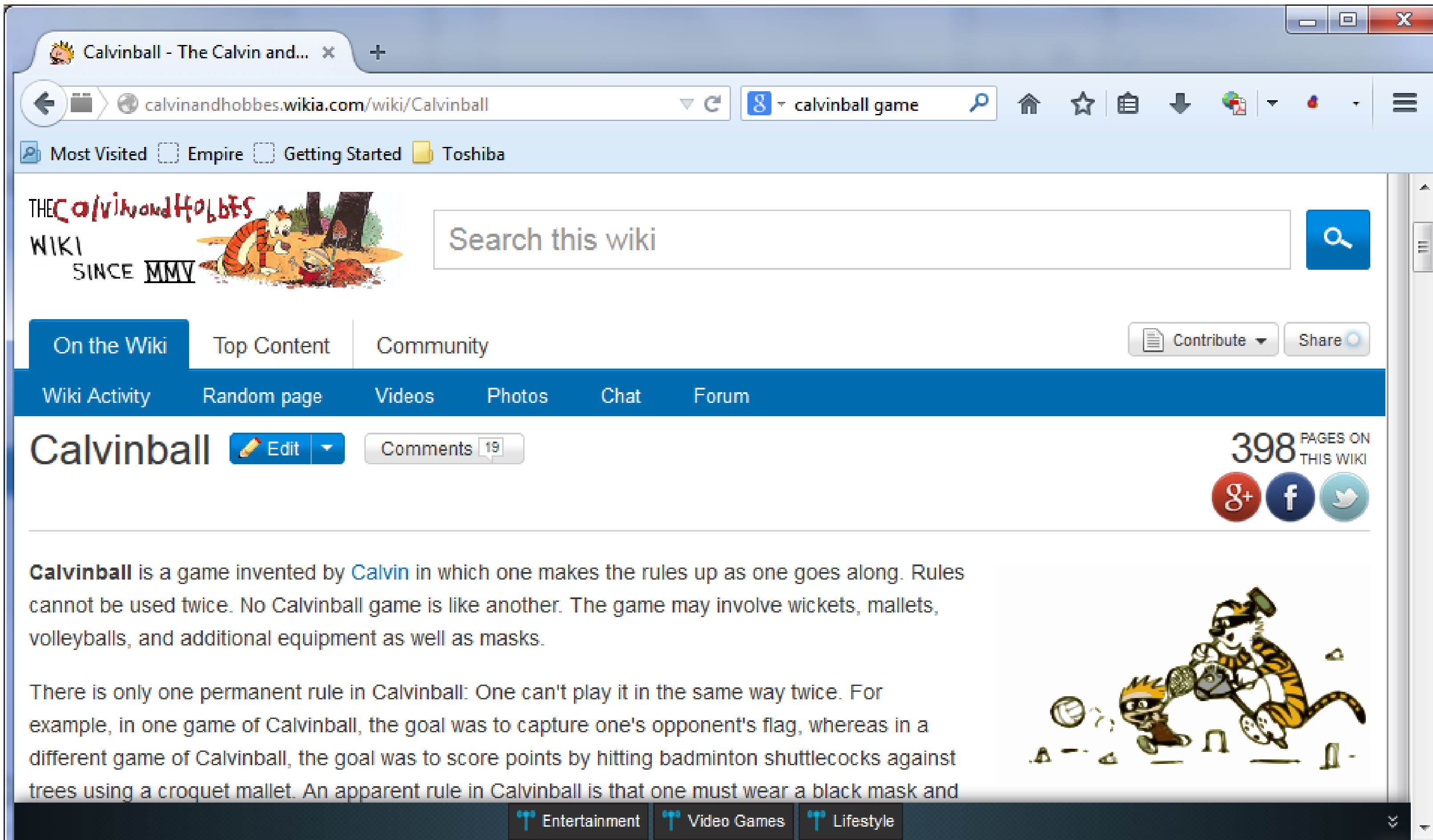
Applications

- Applications:
  - Contracts (e-commerce)
  - International trade
  - Social norms in 3D VW (e.g. Second Life)
  - Human Computer Interaction
  - "What if" scenarios for policy makers
  - Organizations
  - What else?



UNIVERSITAT DE BARCELONA

Norm changes



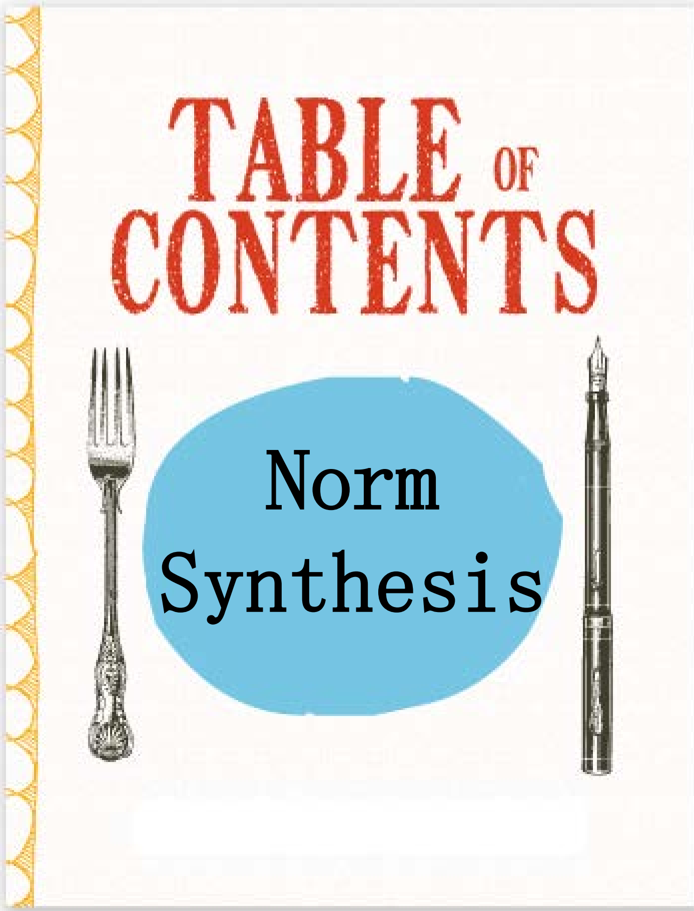
UNIVERSITAT DE BARCELONA

Other practical issues

- Implementation issues:
  - Are norms explicitly represented in the system?
    - If so: In which language? Are they “hardcoded”?
    - How do agents reason about them?
  - Define how norms support agent coordination
  - Decide whether norms:
    - are created by a legislation authority,
    - emerge spontaneously or
    - are negotiated among agents,

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. **Overview of approaches to norm synthesis.**
   - Off-line norm synthesis.
   - ...
3. On-line automatic norm synthesis.
4. Demo and hands-on activity

Norm design

practice I

- Social norms in practice: traffic laws (Shoham and Tennenholtz, 1996).
  - Two-dimensional grid world populated by mobile robots.
  - More than one robot at a cell is a collision.
  - Robots collect and transport items from cell to cell
  - Goal: design a law that prevents collisions

Which norms would you define?

Norm design

practice I

- Social norms in practice: traffic laws (Shoham and Tennenholtz, 1996).
  - First option: law which completely constrains the movements of robots, so that they must all follow a single, completely predetermined path, leaving no possibility of collision:

.

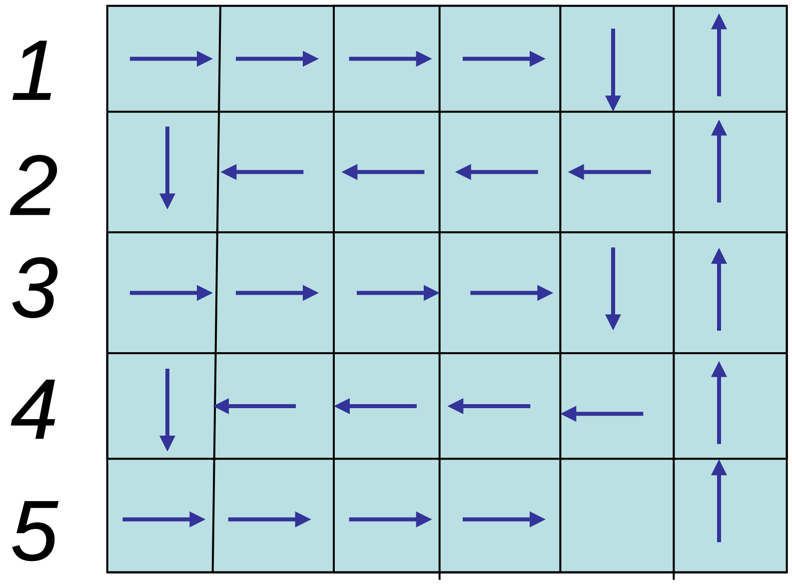
.

Norm design

practice I

- Social norms in practice: traffic laws (Shoham and Tennenholtz, 1996).
  - First option: law which completely constrains the movements of robots, so that they must all follow a single, completely predetermined path, leaving no possibility of collision:
    - 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.

- 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.

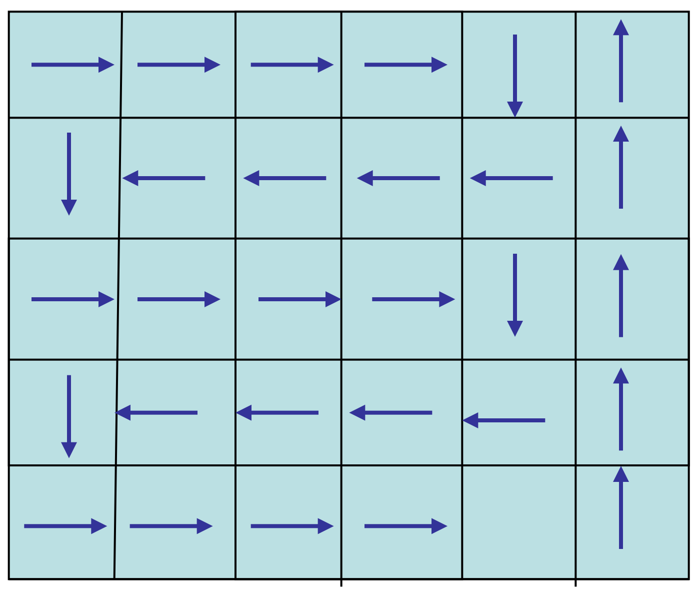


Norm design

practice I

- This rule:
  - 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: changing directions help.

1
2
3
4
5



Offline norm design

- Off-line norm design:
  - 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)

- Environment: $Env = \langle E, e_0, \tau \rangle$
  - $\boldsymbol{E}$ a finite set of discrete, instantaneous **states**:

$$E = \{e, e', \ldots\}.$$

  - Agent **actions** transform the environment:

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

  - A *run*, r, is a sequence of interleaved environment states and actions:

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

  - A **state transformer** function represents behavior of the environment $\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$

UNIVERSITAT DE BARCELONA

From MAS Intro Lesson

Abstract Model of Environment & Agents (II)

- Agent: $Ag : \mathcal{R}^E \to Ac$
  - Function which maps runs to actions
    - An agent makes a decision about what action to perform.
    - Perceive function: maps environment states to **percepts** $E \to Per$
  - Reactive agent: maps percepts to actions $Per^* \to Ac$
  - Deliberative agent:
    - **Action function**: maps internal states to actions $I \to Ac$
    - **Next function**: maps an internal state and a percept to an internal state $I \times Per \to I$
- A *system* is an agent-environment pair
  - $\mathcal{R}(Ag, Env)$: set of runs of agent Ag in environment Env

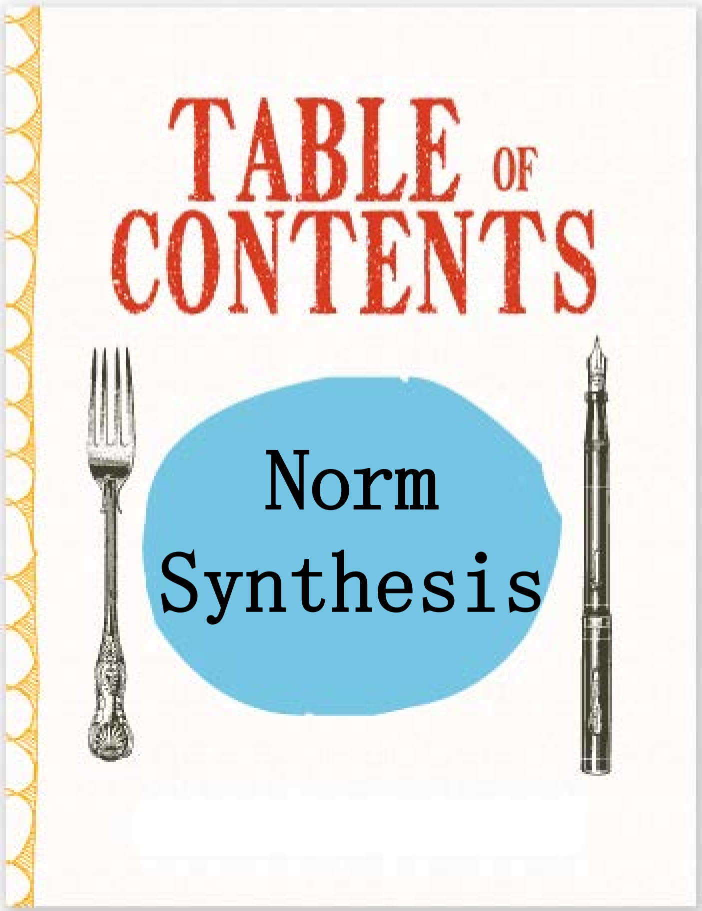
- Offline norm design:
  - Define agents as functions from runs (end in environmental states) to actions:
    - Ag: $R^E \rightarrow Ac$
    - A **constraint** is then a pair <E',α> where
      - $E' \subseteq E$ is a set of environment states
      - $\alpha \in Ac$ is an action

    Reading: "if the environment is in some state $e \in E'$, then the action α is forbidden".
  - A **social law** is a set of such constraints
  - An **agent is legal** to respect a social law if it never attempts to perform a forbidden action in this law.

- Offline norm design:
  - When social laws are useful?
    - Focal states: always legal: agents should be always be able to ‘visit’ them:
    - If the environment is at $e \in F$, it should be possible for the agent to act so as to be able to guarantee that any other state $e' \in F$ is brought about.
    - A useful social law is one that does not constrain the actions of agents so as to make this impossible.

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. **Overview of approaches to norm synthesis.**
   – Off-line norm synthesis.
   – Norm emergence
   – Norm agreement
   – Norm adaptation
   – On-line norm synthesis

Norm mechanisms

- Two approaches for building a normative behaviour in an agent.
  - Top-down: institutional mechanism specifies norms.
  - Bottom-up: mechanisms that can help a norm to emerge: "Don't do to them what you don't want them to do to you"

Norm emergence

- Norm Emergence:
  - Agents have to reach a global agreement on the use of social conventions by using only locally available information:
    - Global: all agents use it
    - Local: each agent decides to adopt one based solely on its own experiences

- Norm Emergence:
  - Scenario: the tee shirt game:
    - All agents have a blue and a red tee shirt
    - They should end up wearing the same colour
    - Play: initially, random colour selection
      - Rounds: each round :
        - » form pairs of agents: they see the colour of the other agent in the pair
        - » At the end: they are allowed to change colour (no messages)
    - Agents need a decision making process based on their memory about previously encountered agents

Let's play it !

Norm emergence

Tee Shirt Game II

- Some decision making alternatives:
  - Simple majority: adopt the most seen colour
  - Simple majority with agent types:
    - Agents are of two types
    - Agents of the same type exchange memories and adopt them as if they were their own (type confidence).
  - Simple majority with communication on success:
    - Communicate useful memories (to pair agents) only if a certain success is reached (prevents noise communication)
  - Highest cumulative reward:
    - Use the strategy with highest cumulative payoff (required)

If still in the mood: play the first one…

Norm emergence

Tee Shirt Game III

- Moreover:
  - Agents can periodically forget everything (memory restart)
    - Agents are open to new ideas
  - Efficiency measure: time to convergence
  - Colour adoption can be seen as a strategy or convention to adopt.
  - Issues:
    - Strategy changing cost
    - Stability: keep agreements in the society

Norm emergence

Tee Shirt Game IV

- Results:
  - All alternatives led to emergent conventions
  - Best results: Highest cumulative reward:
    - Bounded time to convergence
    - It is stable: once reached, agents do not diverge from the norm.
    - Efficient: agents' payoff is not worse than the one they would have received had they stuck with the strategy they initially chose.

Norm emergence

Optional Homework: Tee Shirt Game V

- Exercise: implement a simulation of the tee shirt game with agents
  - Choose one strategy for agents that considers previously encountered agents
  - Check convergence
  - Run it a number of times
  - Optional: combine different strategies

Implementation

UNIVERSITAT DE BARCELONA

Research paper 1

- Title:
  - A categorization of simulation works on norms
- Authors:
  - Bastin Tony Roy **Savarimuthu** and Stephen **Cranefield** (Univ of Otago, New Zeland)
- Year:
  - 2009

Research paper 1

A categorization of simulation works on norms

- Abstract:
  - **Norms are expectations of behaviours** of the agents in a society.
  - Being autonomous:
    - agents might not always follow the norms.
    - they themselves can evolve new norms while adapting to changing needs.
  - Paper:
    - Propose a **life-cycle model for norms** (based on simulation)
    - discuss different **mechanisms** used by researchers to study norm creation, spreading, enforcement and emergence

Research paper 1

A categorization of simulation works on norms

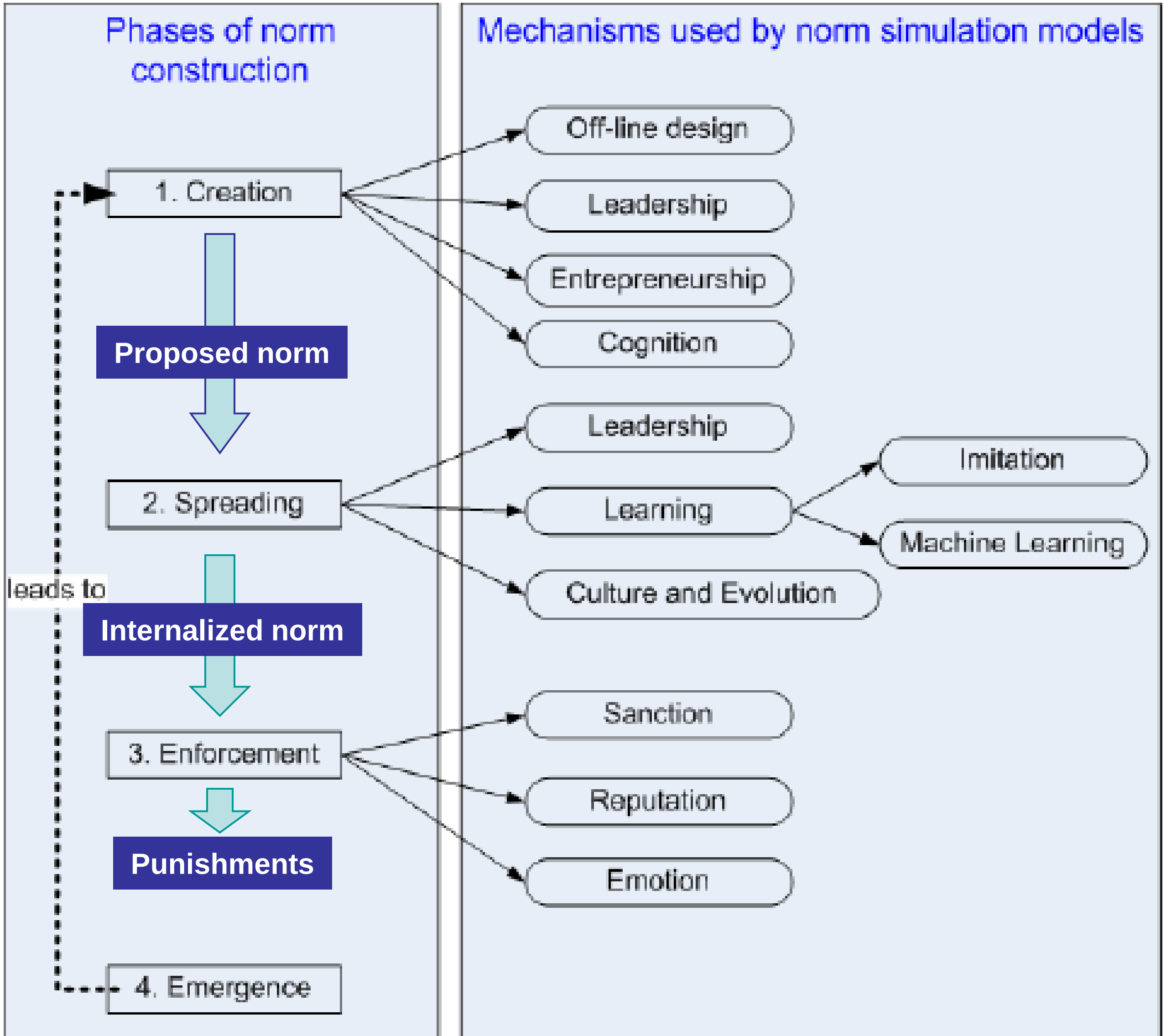
- What are norms?
  - Expectations of an agent about the behaviour of other agents in the society
    - help in sustaining social order and increase the predictability of behaviour in the society.
  - Fulfilling a generalized expectation of behaviour
  - = to conventions
  - Members adhere to norms:
    - for shame,
    - fear to authority,
    - rational appeal to norms,
    - willingness to follow the crowd…
  - Violations may be punished

- Norm aspects:
  - Normative expectation of a behavioural regularity +
  - Norm spreading factor:
    - Notion of advice from powerful leaders
    - Sanctioning mechanism:
      - Monetary (or utilitarian)
      - Physical
      - Emotional (reputation, isolation,..)
    - Imitation
    - Learning

Research paper 1

A categorization of simulation works on norms

- Phases for norm life-cycle



- Phases of norm-life cycle:

1. Norm creation -> proposed norm
2. Spreading -> internalized norms (agents subscribe to norms)
3. Enforcement -> punishment (norm violator)
4. A norm has emerged if:
    - It has spread (i.e. it is followed by a considerable proportion of an agent society and this fact is recognized by most agents)
    - without being explicitly created.

- 1.- Norm creation:
  - The mechanism by which an agent in the society comes to know what the norm of the society is.
  - Approaches:
    - Top-down:
      - Off-line designer:
        - + control
        - - predictability
      - Powerful leader
        - Norm leader provides norms to follower agents
    - Enterpreneurial:
      - An agent comes up with a norm and recommends it to other agents
    - Cognitive:
      - Agents recognize what the norms of a society are
      - based on the observations of interactions (inference).

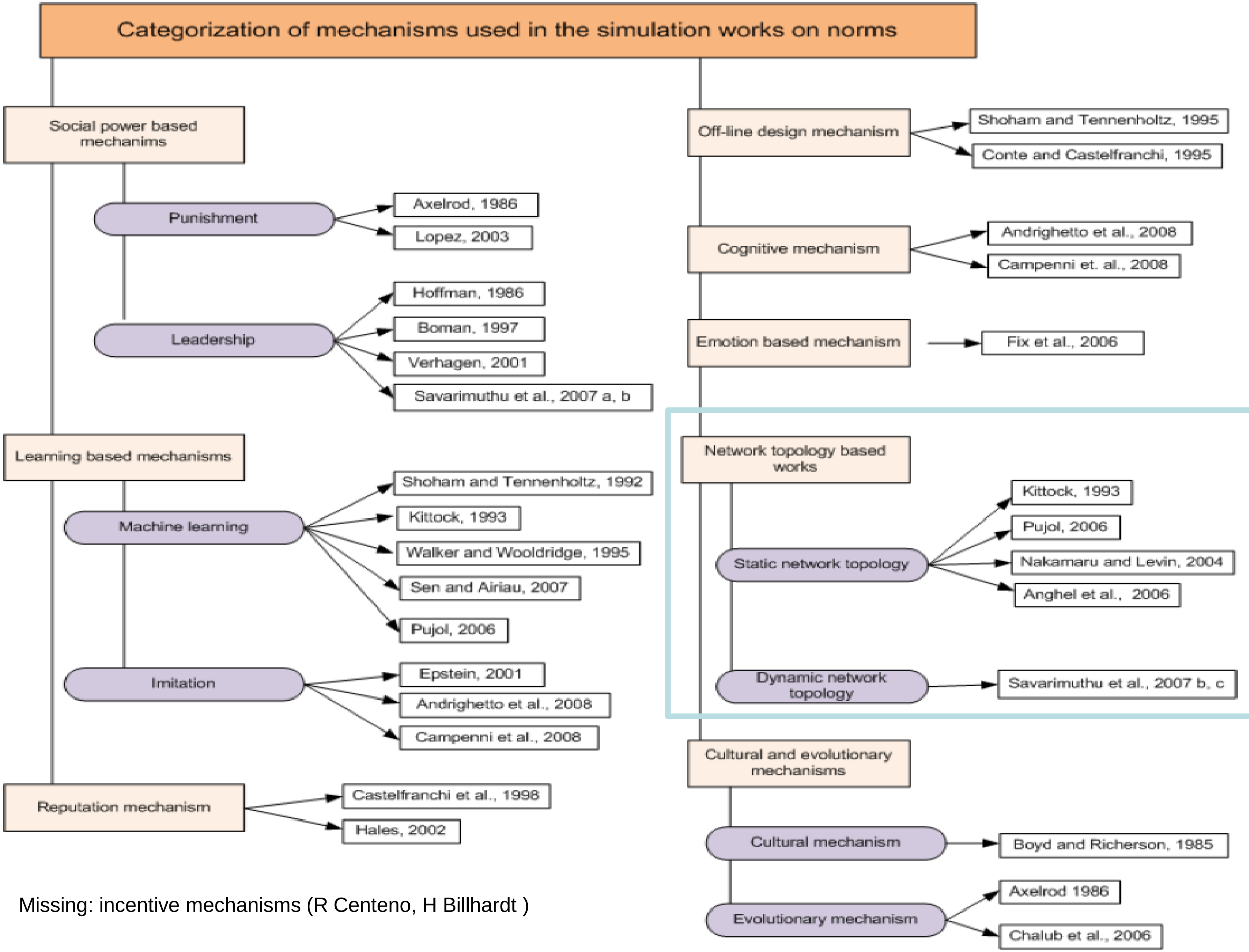
- 2.- Norm spreading
  - Mechanisms:
    - leadership,
    - Imitation: “When in Rome do as the Romans”
    - machine learning,
    - cultural and evolutionary

- 3.- Norm enforcement
  - Process by which norm **violators** are **discouraged** through some form of sanctioning:
    - Punishment (fitness, emotion)
    - Reputation
  - Can be considered as part of 2.-spreading (Axelrod)

Research paper 1

A categorization of simulation works on norms

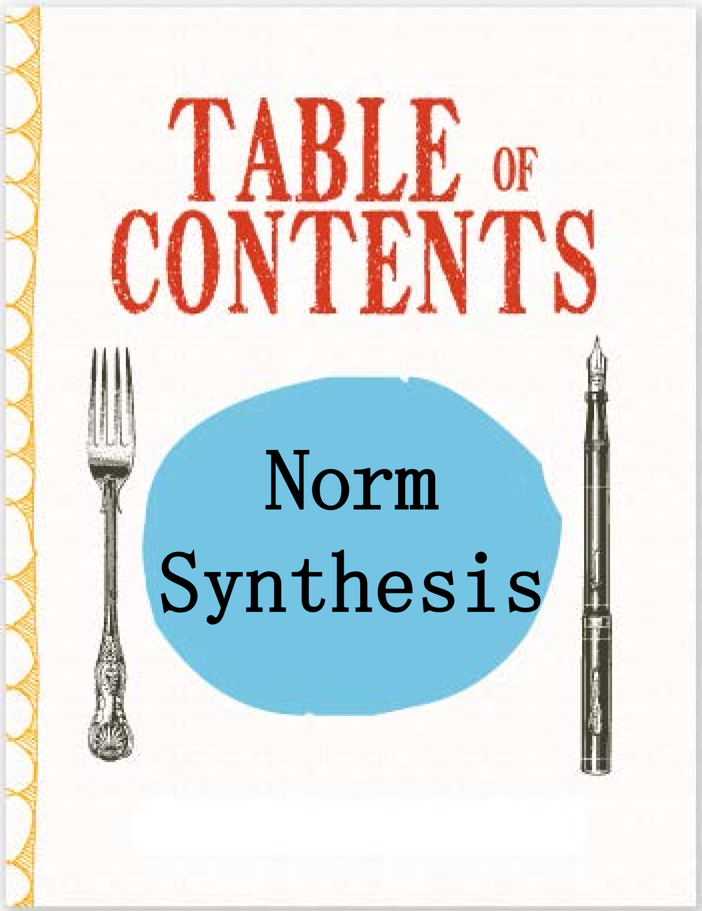
4. Norm emergence:
   - Reaching some significant threshold in the extent of the spread of a norm.
     - Ex.: a society is said to have a norm of gift exchange at Christmas if more than x% of the population follows such a practice.
     - The value of x has varied from 35 to 100 across different simulation based studies of norms.
   - Approaches:
     - An agent comes to know about a norm through mechanisms such as leadership or imitation and when it accepts the norm it contributes to norm spreading and emergence.
     - A cognitive agent generates a personal norm based on observation:
       - Many cognitive agents could generate similar personal norms and for an external observer it might seem that a norm has emerged in a society.
       - Cognitive agents could communicate norms and verify norms.
         - » micro interactions between agents lead to the macro effect of establishing a norm



Missing: incentive mechanisms (R Centeno, H Billhardt)

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. **Overview of approaches to norm synthesis.**
   – Off-line norm synthesis.
   – Norm emergence
   – Norm agreement
   – Norm adaptation
   – On-line norm synthesis

Research paper 2

- Dynamic Specifications for Norm-Governed Systems (A. Artikis, D. Kaponis, J. Pitt, 2009)
  - Previous work: framework for executable specification of norm-governed MAS:
    - Specified at design-time
  - Paper research: specifications may be modified at run-time by the members of the system
    - Action language to encode specifications
    - Scenario: argumentation protocol

Dynamic Norm Specification

Motivation

- In some open MAS,
  - environmental,
  - social or other conditions

  may favour, or even require, specifications to be modifiable during the system execution.
- Ex.:
  - A malfunction of a large number of sensors in a sensor network,
  - Manipulation of a voting procedure due to strategic voting,
  - When an organisation conducts its business in an inefficient manner.

- Assumptions:
  - Open MAS:
    - Agents developed by different parties
    - No direct access to agent's internal state
    - Agents may fail to conform to the system specification in order to achieve their individual goals.
    - Examples:
      - Virtual Organisations, electronic marketplaces, argumentation (dispute resolution) protocols, negotiation protocols
  - Adoption of a bird's eye view of the system

Dynamic Norm Specification

– Normative System:

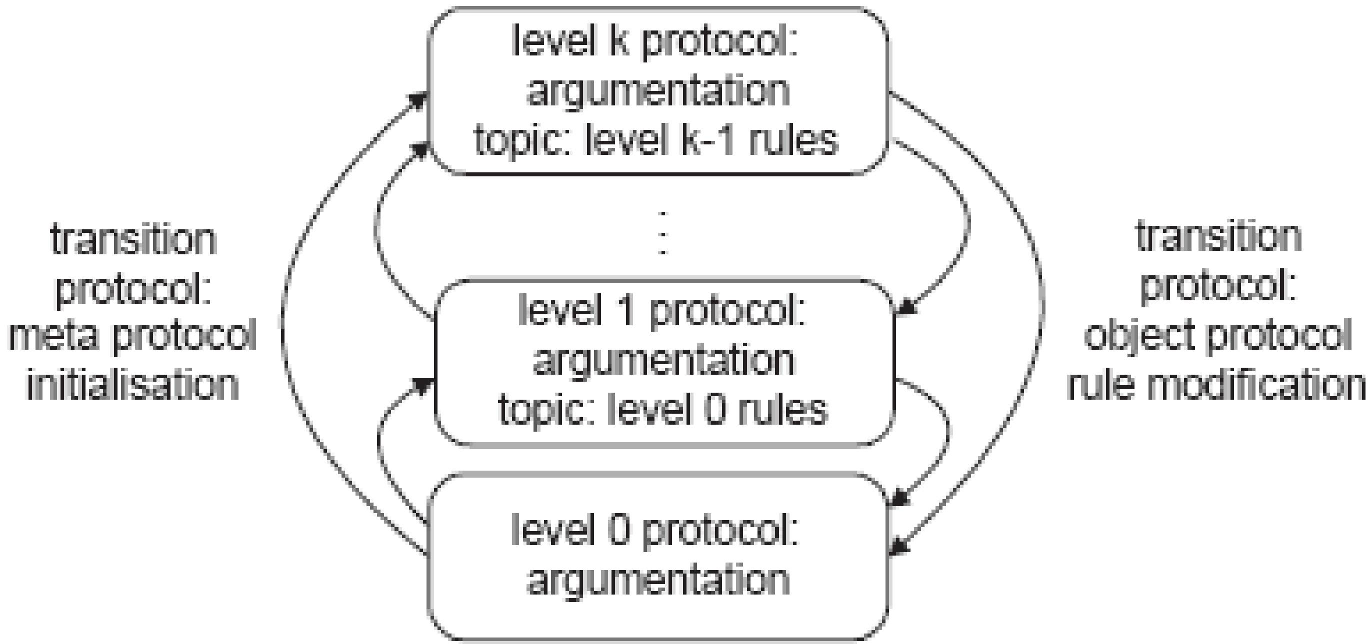
- Actuality and ideality do not necessarily coincide
  - Actuality: what is the case
  - Ideality: what ought to be the case
- Specification of what is permitted, prohibited and obligatory.
- Institutional power:
  - Designated agents, when acting in specified roles, are empowered by an institution to create specific relations or state of affairs
  - Ex.: an agent is empowered to award a contract (to create a bundle of normative relations between the contracting parts).

Dynamic Norm Specification

- Brewka's dynamic argument systems:
  - Argument systems in which, at any point in the disputation, participants may start a meta level debate:
    - The rules of order become the current point of discussion with the intention of altering these rules

- Protocol participants can alter the rules of a protocol *P* during its execution:
  - *P*: **object protocol**
  - Participants start a **meta protocol** to alter *P:*
    - Add a new rule-set
    - Delete an existing one or
    - Replace an existing rule-set with a new one
  - This can be done recursively

– Ex. scenario:
  - Both object and meta protocols are argumentation protocols



Event Calculus

Kowalski and Sergot '86

- Event Calculus:
  - action language to formalize system specifications for representing and reasoning about actions or events and their effects.
- Basic elements:
  - **Fluent** *F*:
    - property that can have different values along time
  - **Term** *F*=*V* : fluent *F* has value *V* if:
    - *F*=*V* has been **initiated by an action** before and
    - **Not terminated by another** action in the meantime.

An Argumentation Protocol

Formalization I

- RTFD* (Rescher's Theory of Formal Disputation)
  - Argumentation (dispute resolution) protocol
  - Formalisation in Event Calculus:
    - Roles:
      - Proponent: claims the topic of the argumentation
      - Opponent
      - Determiner
    - Proponent starts the protocol (claims the topic)

An Argumentation Protocol

Formalization II

- Protagonists (proponent & opponent) take turns to perform actions:

Table 2. Main Actions of RTFD*.

Action	Textual Description
$claim(Protag, Q)$	$Protag$ claims Q
$concede(Protag, Q)$	$Protag$ concedes to Q
$retract(Protag, Q)$	$Protag$ retracts Q
$deny(Protag, Q)$	$Protag$ denies Q
$declare(Det, Protag)$	Det declares $Protag$ the winner of the disputation
$objected(Ag)$	Ag objects to an action

- *Ag*'s action *Act* is followed by a time period where:
  - *Ag* may not perform any actions,
  - The other participants may object to *Act*

An Argumentation Protocol

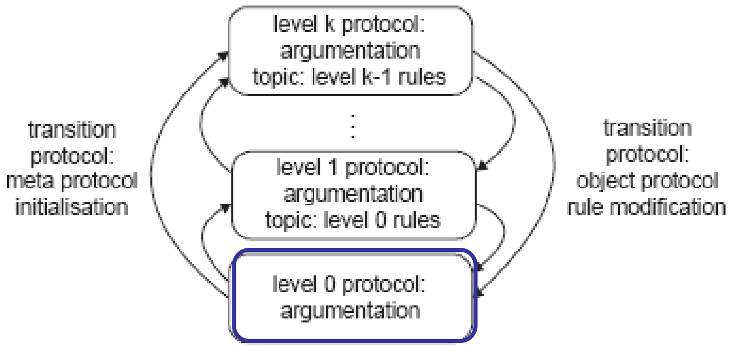
Formalization III

- When the period of the argumentation elapses the Determiner declares a winner that is:
  - The proponent if both participants accept the topic.
  - The opponent if proponent does not accept the topic.
  - Any of them if the proponent accepts the topic and the opponent does not

A Dynamic Argum. Prot.

A dynamic argumentation protocol I

- Each protocol level (PL) has its own protocol state:
  - Add a parameter into the representation:
    - Ex.: Claim(Protag,Q, PL)
- Rules of the argumentation protocol:
  - ‘core’: always part of the protocol specification
  - ‘replaceable’: by meta-protocol during protocol execution
    - Ex: replace the `accept' rule with the `sic' rule

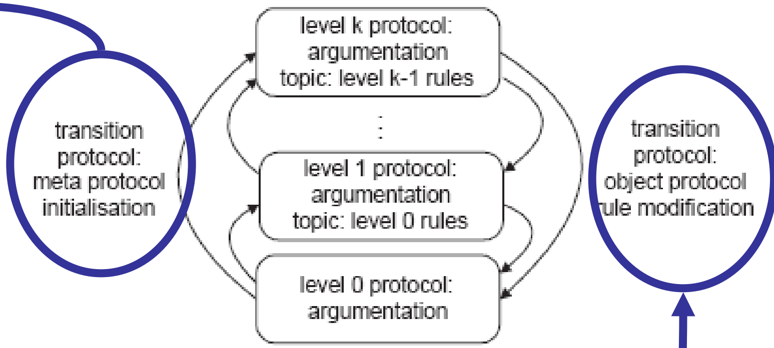


- Rule sets:
  - ‘active’: part of the protocol at given PLs (ex.:1,2): holdsAt(active(*R*)=[0,2],t)
  - ‘inactive’.

UNIVERSITAT DE BARCELONA

A Dynamic Argum. Prot.

A dynamic argumentation protocol II



- **Transition protocol**
  - To start a protocol of level n+1

to modify the protocol rules of level n.

- Example:
  - PL=n, a protagonist proposes a modification of the rules of this protocol level.
    - If the protagonist is empowered to propose such a change then the protocol of level n+1 begins;
    - Otherwise the proposal is ignored.
  - The topic of the n+1 protocol is the proposed rule modification
  - If the agent that successfully proposed the modification is declared the winner of the argument of level n+1 then the rules of level n will be modified.

A Dynamic Argum. Prot.

Animation of a 2-level argument system I

- Replaceable rule: sic

Time	Action
0	$claim(agent_1, murderer(jack), 0)$
	...
14	$claim(agent_1, on(blood, shoe), 0)$
14	$objected(agent_2, 0)$
15	$oTimeout(0)$
18	$claim(agent_2, illegal\_info(on(blood, shoe)), 0)$
30	$pTimeout(0)$
31	$concede(agent_1, illegal\_info(on(blood, shoe)), 0)$
45	$oTimeout(0)$
46	$propose(agent_2, replace(sic, sic\_ill\_info), 0, 0)$
49	$claim(agent_2, replace(sic, sic\_ill\_info), 1)$
	...
77	$dTimeout(1)$
78	$declare(det, agent_2, 1)$
93	$endTimeout(1)$
	...
135	$endTimeout(0)$

A Dynamic Argum. Prot.

Animation of a 2-level argument system II

- PL= 0:
  - t=0
    - proponent = ag1
    - Topic Q: “Jack is a murderer”
  - t=14
    - New evidence Q' on Q by ag1: “victim’s blood on Jack’s shoe”
    - Opponent= ag2
    - Ag2 objects to Q'
    - Effects of Q’:

premise(agent1 ; on(blood; shoe); 0)=t
premise(agent2 ; on(blood; shoe); 0)=u

    - Ag2’s objection does not block the effects because Q' is not objectionable

Time	Action
0	$claim(agent_1, murderer(jack), 0)$
	...
14	$claim(agent_1, on(blood, shoe), 0)$
14	$objected(agent_2, 0)$
15	$oTimeout(0)$
18	$claim(agent_2, illegal\_info(on(blood, shoe)), 0)$
30	$pTimeout(0)$
31	$concede(agent_1, illegal\_info(on(blood, shoe)), 0)$
45	$oTimeout(0)$
46	$propose(agent_2, replace(sic, sic\_ill\_info), 0, 0)$
49	$claim(agent_2, replace(sic, sic\_ill\_info), 1)$
	...
77	$dTimeout(1)$
78	$declare(det, agent_2, 1)$
93	$endTimeout(1)$
	...
135	$endTimeout(0)$

A Dynamic Argum. Prot.

Animation of a 2-level argument system III

- PL= 0:
  - t=14
    - sic rule applies: both protagonists accept Q'
  - t=15 (Ag2 turn to speak)
  - t=18:
    - Ag2 claims Q'': "Q' was obtained illegally "
  - t=30 (Ag1 turn to speak)
  - t=31
    - Ag1 concedes Q'' (but does not change Q' acceptance due to sic)

Time	Action
0	$claim(agent_1, murderer(jack), 0)$
	...
14	$claim(agent_1, on(blood, shoe), 0)$
14	$objected(agent_2, 0)$
15	$oTimeout(0)$
18	$claim(agent_2, illegal\_info(on(blood, shoe)), 0)$
30	$pTimeout(0)$
31	$concede(agent_1, illegal\_info(on(blood, shoe)), 0)$
45	$oTimeout(0)$
46	$propose(agent_2, replace(sic, sic\_ill\_info), 0, 0)$
49	$claim(agent_2, replace(sic, sic\_ill\_info), 1)$
	...
77	$dTimeout(1)$
78	$declare(det, agent_2, 1)$
93	$endTimeout(1)$
	...
135	$endTimeout(0)$

A Dynamic Argum. Prot.

Animation of a 2-level argument system IV

- PL= 0:
  - t=45 (Ag2 turn to speak)
  - t=46
    - Ag2 proposes a rule change sic -> sic_ill_info that is successful:
- PL=1
  - t=49:
    - proponent = ag2
    - Topic Q1: “change sic for sic_ill_info” to deal with illegal information
  - t=77 Determiner turn
  - t=78
    - Det declares ag2 the winner.

Time	Action
0	$claim(agent_1, murderer(jack), 0)$
	...
14	$claim(agent_1, on(blood, shoe), 0)$
14	$objected(agent_2, 0)$
15	$oTimeout(0)$
18	$claim(agent_2, illegal\_info(on(blood, shoe)), 0)$
30	$pTimeout(0)$
31	$concede(agent_1, illegal\_info(on(blood, shoe)), 0)$
45	$oTimeout(0)$
46	$propose(agent_2, replace(sic, sic\_ill\_info), 0, 0)$
49	$claim(agent_2, replace(sic, sic\_ill\_info), 1)$
	...
77	$dTimeout(1)$
78	$declare(det, agent_2, 1)$
93	$endTimeout(1)$
	...
135	$endTimeout(0)$

A Dynamic Argum. Prot.

Animation of a 2-level argument system V

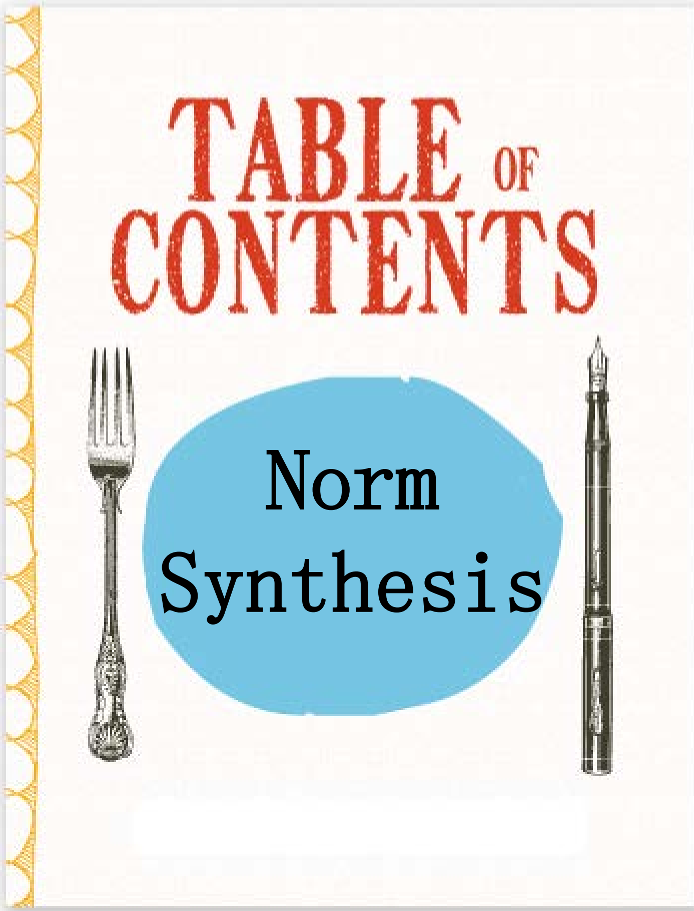
- PL=1
  - t=78
    - Det declares ag2 the winner:
- PL=0 the rule is changed
  - sic becomes inactive at PL=0
- PL=1
  - t=93: PL 1 Ends
- PL=0
  - t=94: If retroactive effects:
    - accepts(Ag1,Q',0) != true
    - accepts(Ag2,Q',0) != true

Time	Action
0	$claim(agent_1, murderer(jack), 0)$
	...
14	$claim(agent_1, on(blood, shoe), 0)$
14	$objected(agent_2, 0)$
15	$oTimeout(0)$
18	$claim(agent_2, illegal\_info(on(blood, shoe)), 0)$
30	$pTimeout(0)$
31	$concede(agent_1, illegal\_info(on(blood, shoe)), 0)$
45	$oTimeout(0)$
46	$propose(agent_2, replace(sic, sic\_ill\_info), 0, 0)$
49	$claim(agent_2, replace(sic, sic\_ill\_info), 1)$
	...
77	$dTimeout(1)$
78	$declare(det, agent_2, 1)$
93	$endTimeout(1)$
	...
135	$endTimeout(0)$

UNIVERSITAT DE BARCELONA

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. **Overview of approaches to norm synthesis.**
   – Off-line norm synthesis.
   – Norm emergence
   – Norm agreement
   – Norm adaptation
   – On-line norm synthesis

UNIVERSITAT DE BARCELONA

Research Paper 3

Adaptation of Autonomic Electronic Institutions through norms and institutional agents

Eva Bou, Maite López-Sánchez, J. A. Rodríguez-Aguilar

Institut d'Investigació en Intel·ligència Artificial (IIIA-CSIC)

Universitat de Barcelona (UB)

- Electronic Institution: regulated virtual environment where agents interact
- Autonomic Electronic Institution:

$$<PS, N, DF, G, P_i, P_e, P_a, V, \delta, \gamma>$$

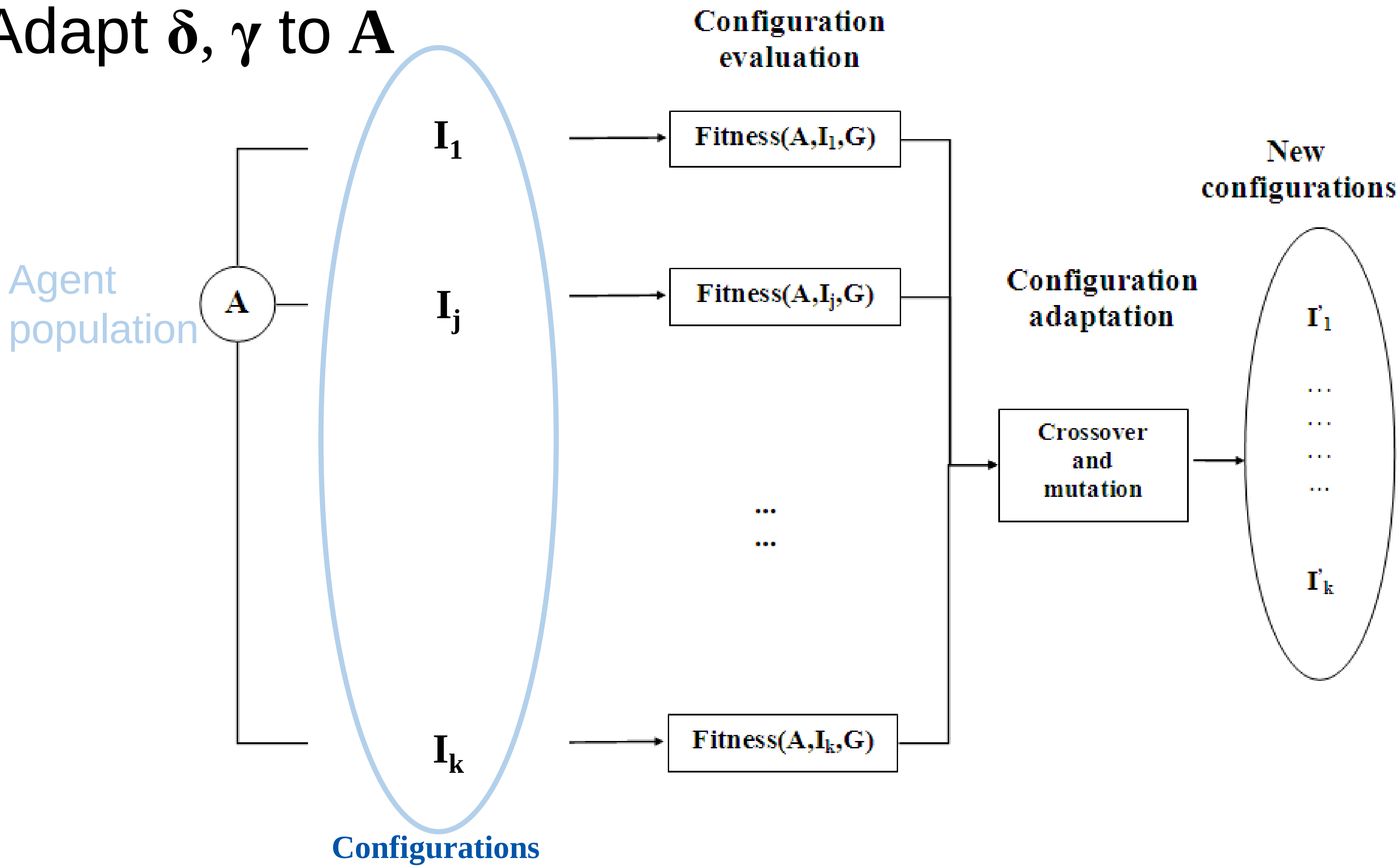
- N: set of Norms, norm N_i has parameters $<p^N_{i1}, \ldots, p^N_{im_i}>$
- G = $\{c_1, \ldots, c_p\}$ set of institutional Goals

  defined as constraints: c_i is an expression $g_i(V) \lhd [m_i, M_i]$
- δ : N x G x V → N *normative transition function*, to ***adapt*** to changing circumstances

Learning Model

Genetic Algorithm

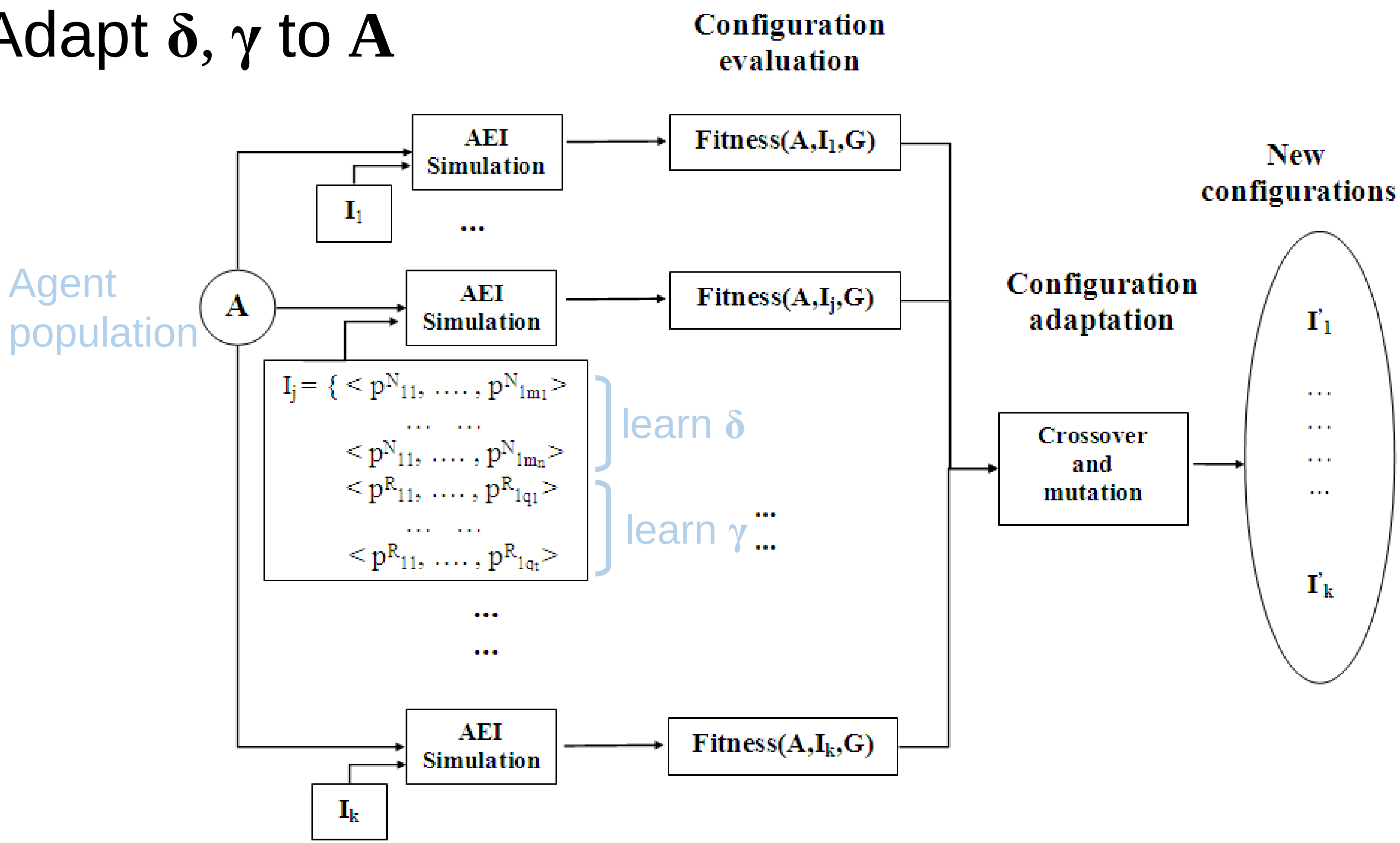
- Adapt **δ**, **γ** to **A**



Learning Model II

Genetic Algorithm

- Adapt **δ**, **γ** to **A**



Case Study

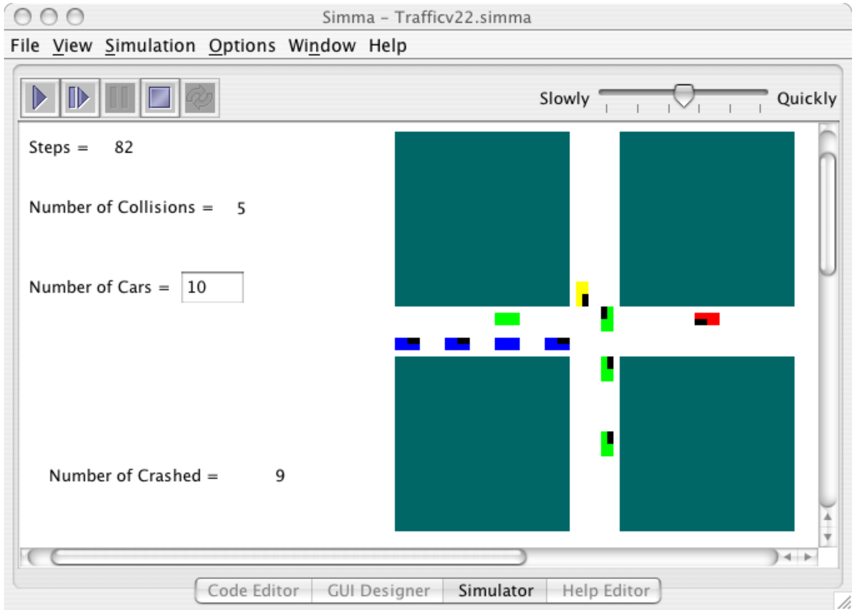
Traffic Control

- Traffic Regulation Authority as an AEI.
  - Simulation: Simma MAS tool
  - We focus on a two-road junction (traffic scene)
    - Cars: external agents
    - Agents' institutional state:

$P_a = <a_1, \ldots, a_n>$,

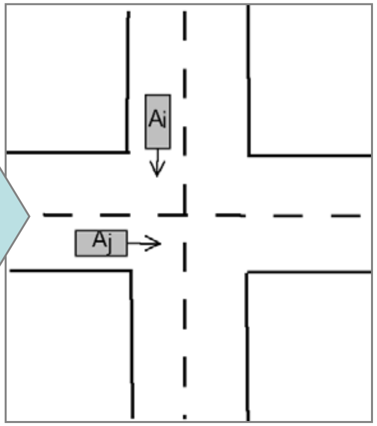
a_j represents A_j

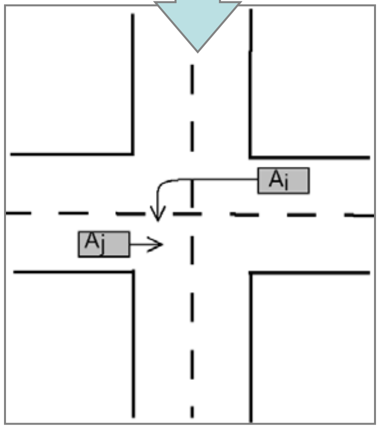
$a_j = <x_j, y_j, h_{jx}, h_{jy}, speed_j, indicator_j, offenses_j, accidents_j, distance_j, points_j>$



- Norms:
  - Have associated penalties (point reductions).
  - Related to actions performed by cars:
    - Right priority norm
    - Front priority norm

Action	$in(a_i, J_{BE}, t-1) \wedge$ $in(a_i, (x_i^{t-1} + h_{ix}^{t-1}, y_i^{t-1} + h_{iy}^{t-1}), t) \wedge$ $\neg indicator(a_i, right, t-1)$
Pre-conditions	$right(a_i, a_j, t-1)$
Consequence	$points_i^t = points_i^t - fine_{right}$





Traffic AEI IV

- Car agents decide wheter to comply with a norm based on four parameters:

<fulfill_prob, high_punishment, inc_prob, police>

$$final\_prob = \begin{cases} police \cdot fulfill\_prob & fine \leq high\_punishment \\ police \cdot (fulfill\_prob + inc\_prob) & fine > high\_punishment \end{cases}$$

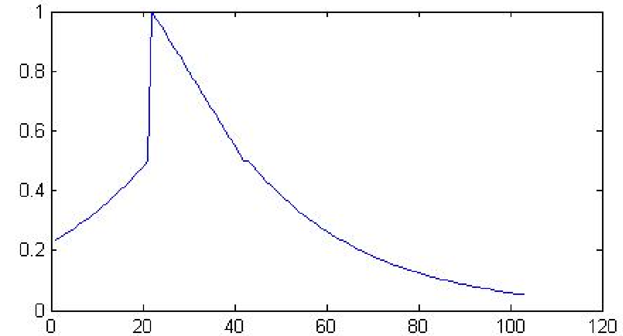
- Institutional agents in the traffic scene represent Traffic Authority employees (police agents).

- Goals:
  - constraints upon a combination of reference values:

$$G = \langle\, g(col) \in [0, maxCol],\ g(off) \in [0, maxOff],\ g(crash) \in [0, maxCrash],\ g(block) \in [0, maxBlock],\ g(expel) \in [0, maxExpel],\ g(police) \in [0, maxPolice]\,\rangle$$

  - g_i function over the reference values
  - degree of satisfaction of a goal $f(x,[m,M],\mu)$

$$f(x,[m,M],\mu) = \begin{cases} \dfrac{\mu}{e^{k\frac{m-x}{M-m}}} & x < m \\ 1-(1-\mu)\dfrac{x-m}{(M-m)} & x \in [m,M] \\ \dfrac{\mu}{e^{k\frac{x-M}{M-m}}} & x > M \end{cases}$$



m=20, M=40, μ=0.5, k=0.75

Traffic AEI VII

Goals II

- Fitness function to combine multiple goals:

$$O(V) = \sum_{i=1}^{|G|} w_i \sqrt{f(g_i(V), [m_i, M_i], \mu_i)}$$

  - w_i weighting factors

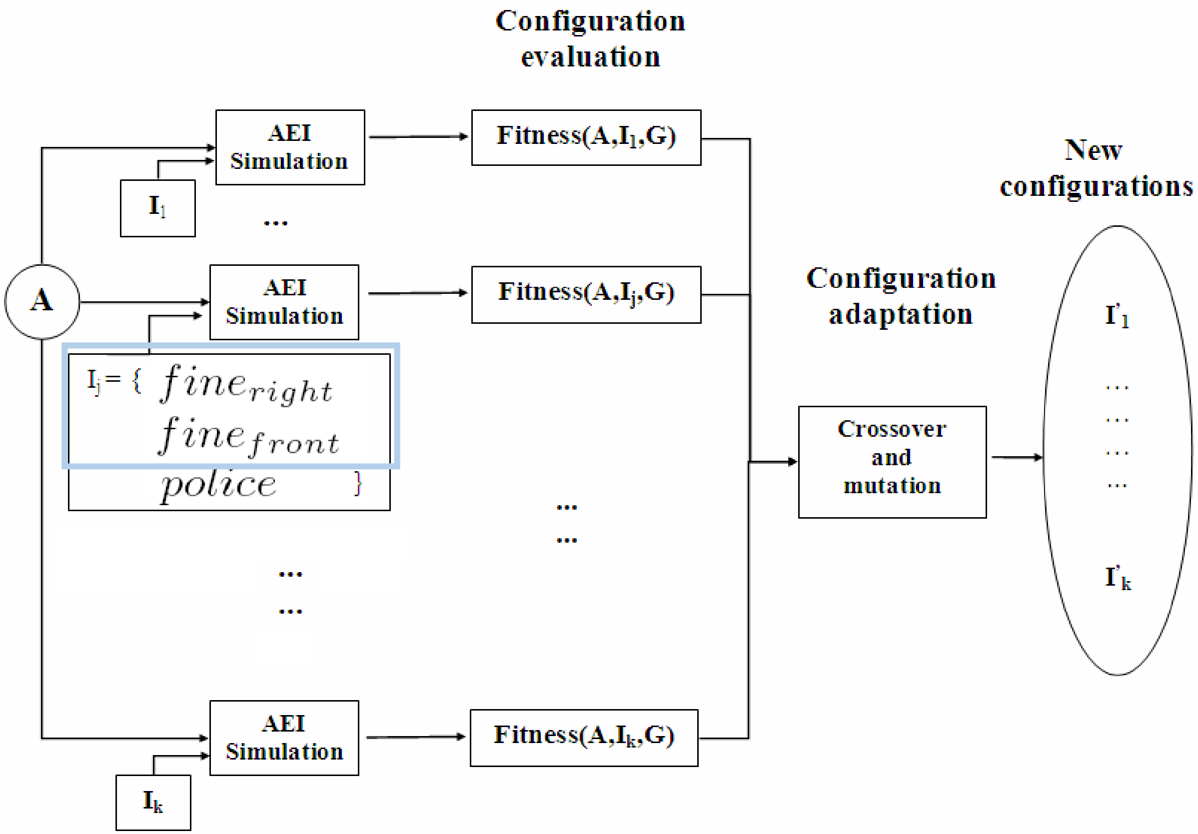
$$O(V) = \frac{4}{10} \cdot \sqrt{f(g(col), [0, maxCol], \frac{1}{2})} + \frac{4}{10} \cdot \sqrt{f(g(off), [0, maxOff], \frac{1}{2})} +$$

$$\frac{1}{10} \cdot \sqrt{f(g(expel), [0, maxExpel], \frac{3}{4})} + \frac{1}{10} \cdot \sqrt{f(g(police), [0, maxPolice], 0)}$$

Results

Learning

Parameters	population1	population2	population3	population4	population5
$fulfill\_prob$	0.5	0.5	0.5	0.5	0.5
$high\_punishment$	5	8	10	12	14
inc_prob	0.2	0.2	0.2	0.2	0.2



Results II

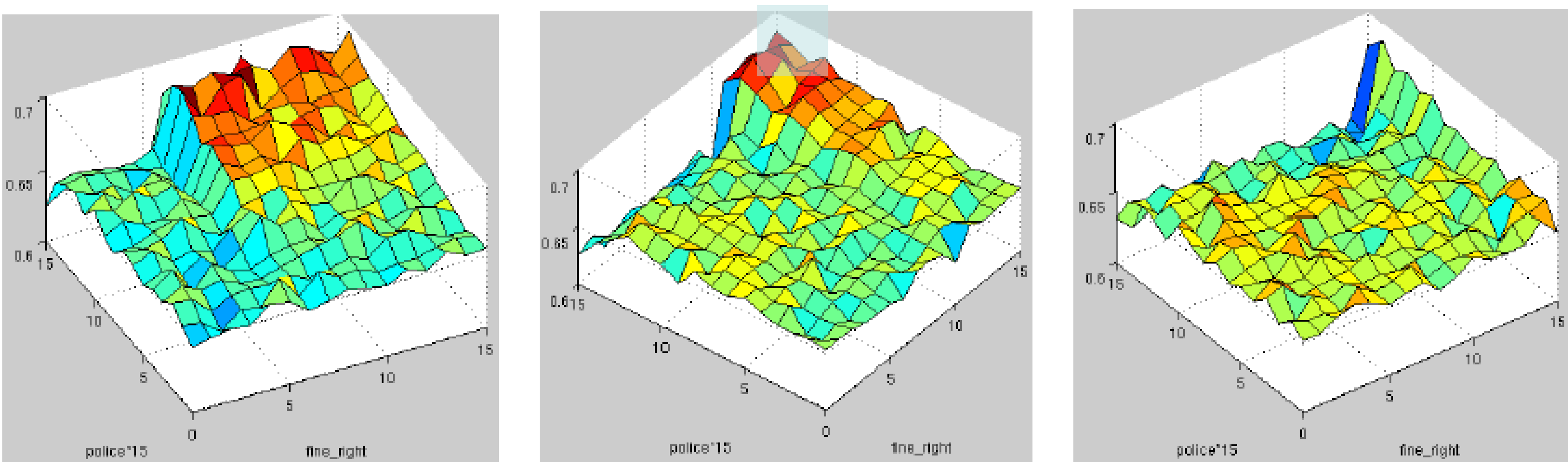
Parameters	population1	population2	population3	population4	population5
$fulfill\_prob$	0.5	0.5	0.5	0.5	0.5
$high\_punishment$	5	8	10	12	14
inc_prob	0.2	0.2	0.2	0.2	0.2

Population	**Learned** $fine_{right}$	**Learned** $fine_{front}$	**Learned** $police$	Goal satisfaction
population1	**15, 12, 7**	**8, 14, 13**	**0.93, 0.93, 0.93**	0.699, 0.7, 0.691
population2	**13, 13, 14**	**10, 11, 9**	**0.93, 0.93, 0.93**	0.689, 0.694, 0.691
population3	**15, 12, 15**	**14, 11, 15**	**0.93, 0.87, 0.93**	0.685, 0.681, 0.685
population4	**15, 13, 15**	**14, 13, 13**	**0.93, 0.93, 0.87**	0.676, 0.686, 0.68
population5	**15, 15, 15**	**15, 15, 8**	**0.93, 0.93, 0.93**	0.668, 0.674, 0.677

$$final\_prob = \begin{cases} police \cdot fulfill\_prob & fine \leq high\_punishment \\ police \cdot (fulfill\_prob + inc\_prob) & fine > high\_punishment \end{cases}$$

Results III

Parameters	population1	population2	population3	population4	population5
$fulfill\_prob$	0.5	0.5	0.5	0.5	0.5



(a) Population1 (b) Population3 (c) Population5

AEI learns traffic norms that fulfill its goals for different agent populations

Future Work

- Extend institutional adaptation capabilities to dynamically adapt to any change in the population.
  - CBR approach
- Develop a more complex traffic network:
  - decentralized approach where different areas (i.e., junctions) are regulated by different institutions.

Research Paper 4

Self-adaptation in Autonomic Electronic Institutions through Case-Based Reasoning

Eva Bou, Maite López-Sánchez, J. A. Rodríguez-Aguilar
Institut d'Investigació en Intel·ligència Artificial (IIIA-CSIC)
Universitat de Barcelona (UB)

Adapting Autonomic Electronic Institutions to Heterogeneous Agent Societies

Eva Bou, Maite López-Sánchez, J. A. Rodríguez-Aguilar, Jaime S. Sichman
IIIA-CSIC, UB, Univ. de Sao Paulo

- Adapt $\boldsymbol{\delta}$, γ to $\mathbf{A}$

1st step: **Genetic Algorithms (GA)**

Lean best parameters for **prototypical** agent populations

2nd step: **Case-Based Reasoning (CBR)**

Adapt to **any** agent population

– CBR: Solves new problems reusing past experiences:
  – uses solutions from similar problems previously learnt (cases).

– Problem: given the current agent population, provide the best parameters so to accomplish institutional goals.

- Case similarity function: (distance)
  - Aggregated function:

$$S(C^i, C^j) = w_1 \cdot s\_AEI(C^i, C^j) + w_2 \cdot s\_V(C^i, C^j) + w_3 \cdot s\_pop(C^i, C^j)$$

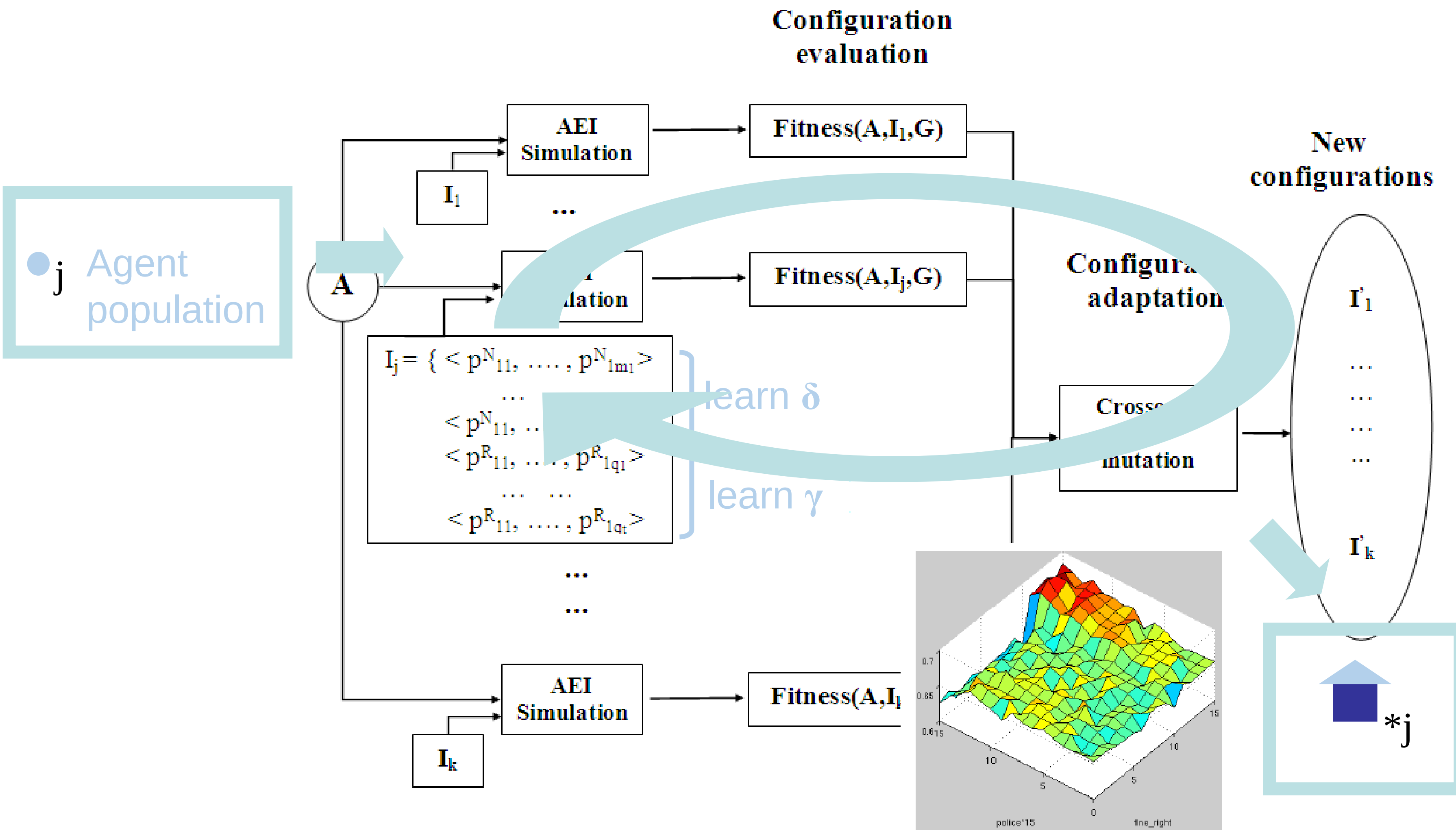
  - attribute distance:

$$sim(attr^i, attr^j) = \frac{|attr^i - attr^j|}{max(attr) - min(attr)}$$

UNIVERSITAT DE BARCELONA

Traffic AEI

Learning: Genetic Algorithm



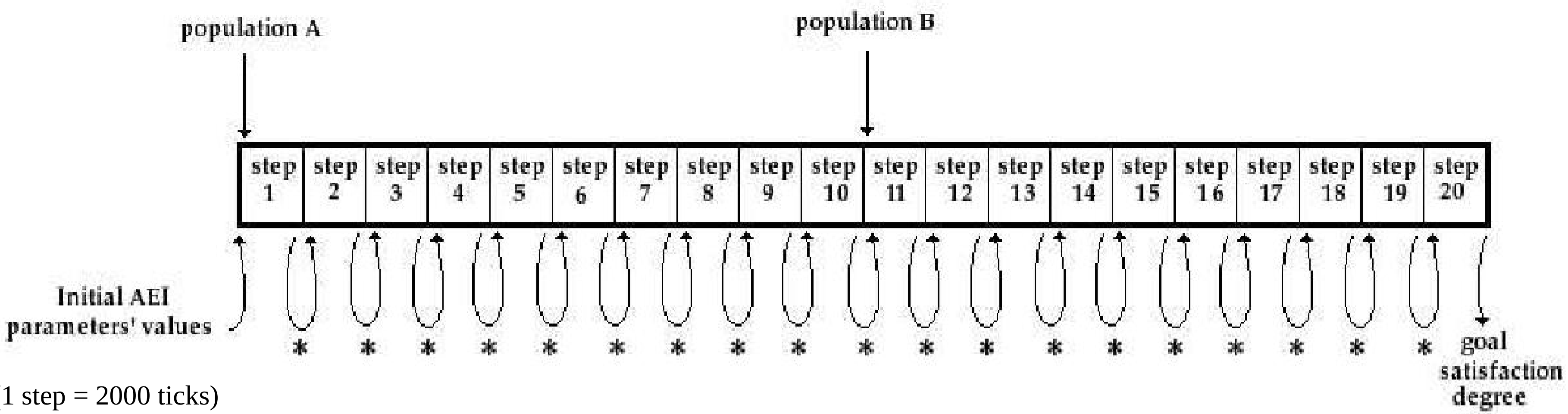
- Case definition
  - N^p : norm parameters ($fine_{right}$, $fine_{front}$)
  - PS^p: performative structure parameter (*police*)
  - V: reference values (*col, crash, off, block, expel*)
  - *pop*: statistic data about agent population's behaviour (*meanOff, medianOff, …*)
  - N^{p*}: norm parameters' best values ($fine^*_{right}$, $fine^*_{front}$)
  - PS^{p*}: performative structure parameter's best value (*police**)

- Case generation:
  - 7 prototypical populations
  - AEI's 108 (=6x6x3) different parameters:
    - $fine_{right}$, $fine_{front}$ ∈ {0, 3, 6, 9, 12, 15}
    - $police$ ∈ { 0.8, 0.9, 1 }

756 cases
2000 ticks each

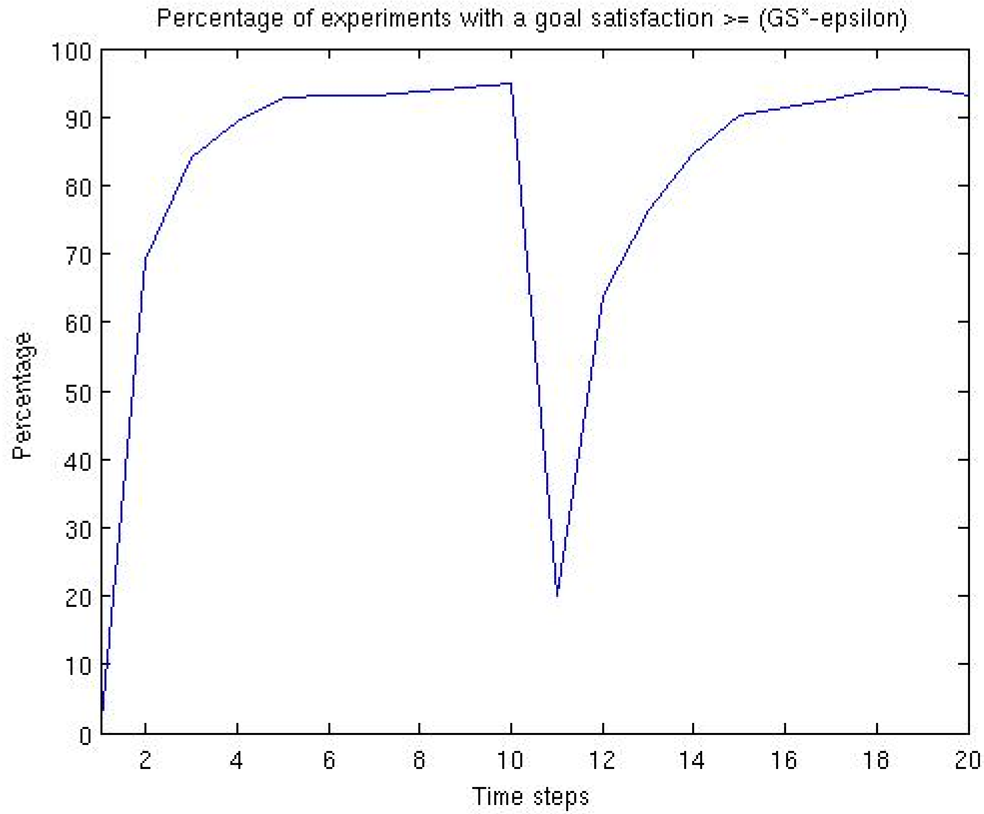
Populations	Pop. 1	Pop. 2	Pop. 3	Pop. 4	Pop. 5	Pop. 6	Pop. 7
$fulfill\_prob$	0.5	0.5	0.5	0.5	0.5	0.5	0.5
$high\_punishment$	0	3	5	8	10	12	14
inc_prob	0.4	0.4	0.4	0.4	0.4	0.4	0.4
$fine^*_{right}$	2	5	8	11	13	14	15
$fine^*_{front}$	1	4	6	9	12	13	15
$police^*$	1	1	1	1	1	1	1

- Can the AEI adapt to any agent population?
  - Experimental setting:
    - Initially: $fine_{right}$ = $fine_{front}$=0 and $police$=0.8
    - Population A = Pop1 ….Pop15 , Population B = Pop7
    - Every step AEI checks if adaptation is required
      - If Goals are not satisfied (G<G*-Ɛ) → Retrieve a case from the KB



(1 step = 2000 ticks)

- Can the AEI satisfy its goals?
  - G ≥ G*- ε
  - 750 experiments:
    - 15 pop x 50 runs



- Can the AEI satisfy its goals?
  - Most times YES
    - Number of experiments stabilized in first 10 steps (population A = Pop1 …Pop15):

Steps	1	2	3	4	5	6	7	8	9	10	Not stabilized
Stabilized	0	518	153	36	19	9	5	2	1	1	6
Percentage stabilized	0	69	20.4	4.8	2.5	1.2	0.7	0.3	0.1	0.1	0.8

    - Number of experiments stabilized in last 10 steps (change to population B = Pop7):

Steps	11	12	13	14	15	16	17	18	19	20	Not stabilized
Stabilized	157	332	102	70	46	15	15	8	4	1	0
Percentage stabilized	20.9	44.2	13.6	9.3	6.1	2	2	1	0.5	0.1	0

UNIVERSITAT DE BARCELONA

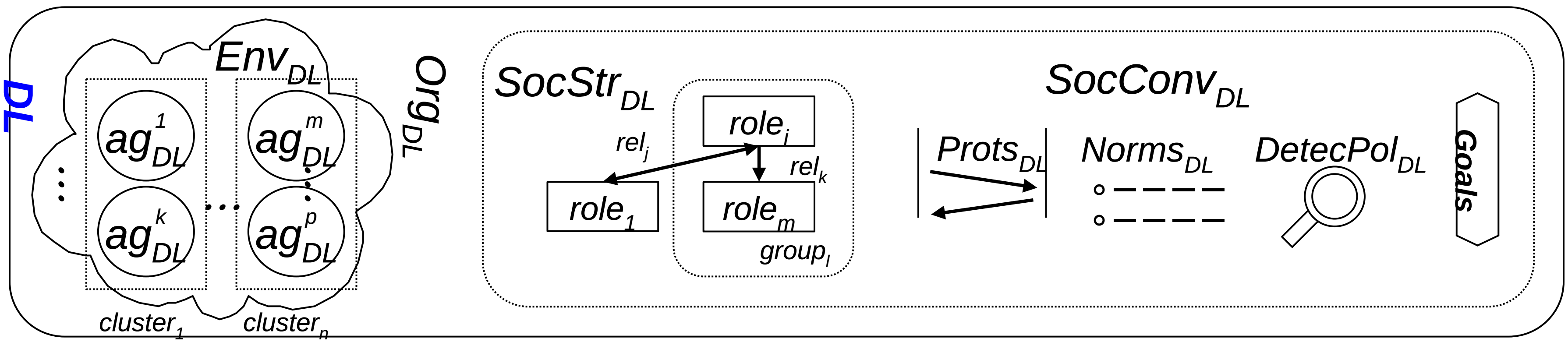
Adaptive Organisation-Centred Multi-Agent Systems

Jordi Campos Miralles

Summary of Ph.D. Dissertation
Barcelona, July 2011

Supervisors:
Maite López-Sánchez (Universitat de Barcelona, UB)
Marc Esteva (Institut d'Intel·ligència Artificial, IIIA-CSIC)

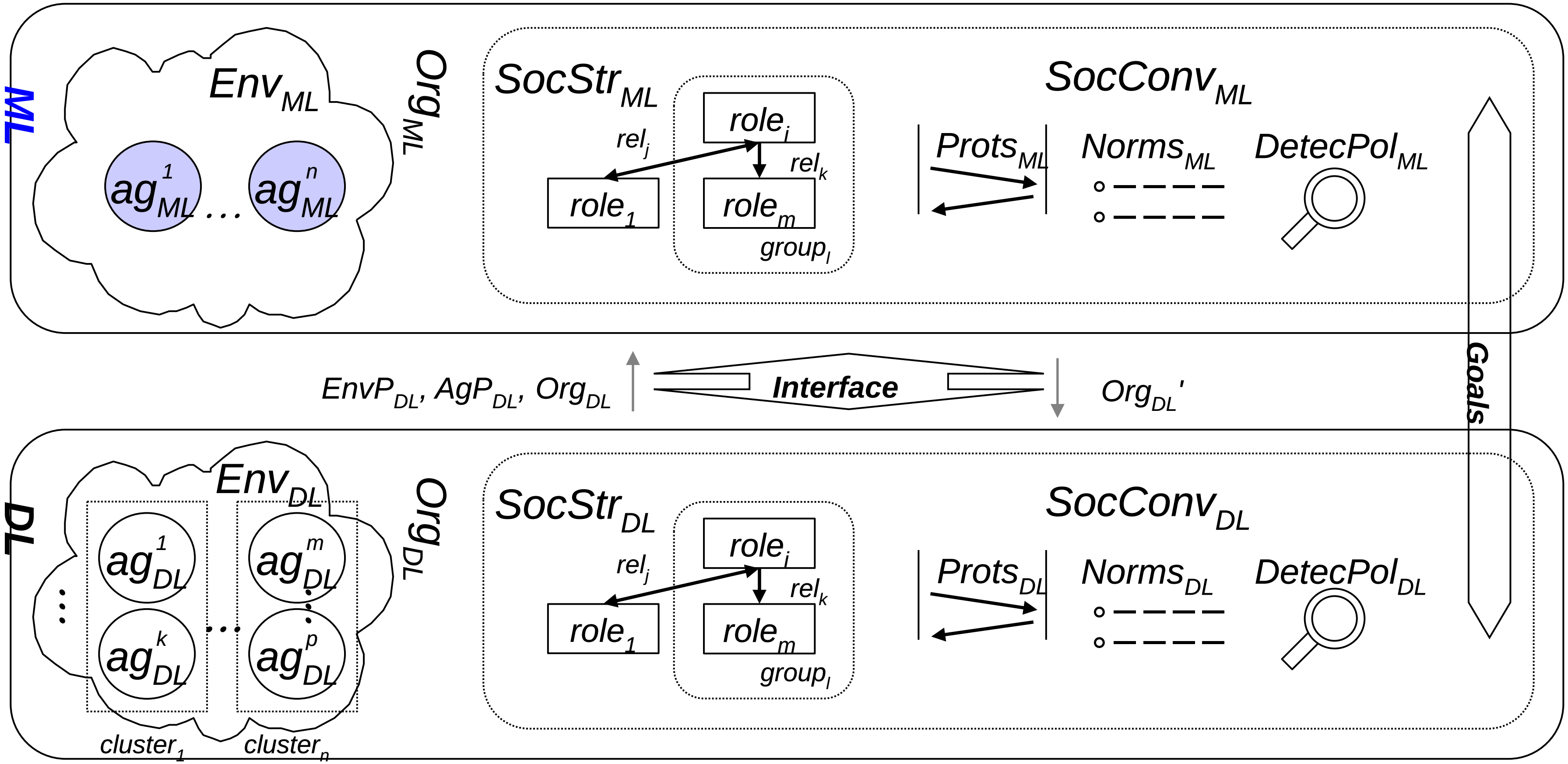
- **Abstract architecture with 2 levels:**



- a **Domain-Level (DL)** = agents organised to perform domain's activity

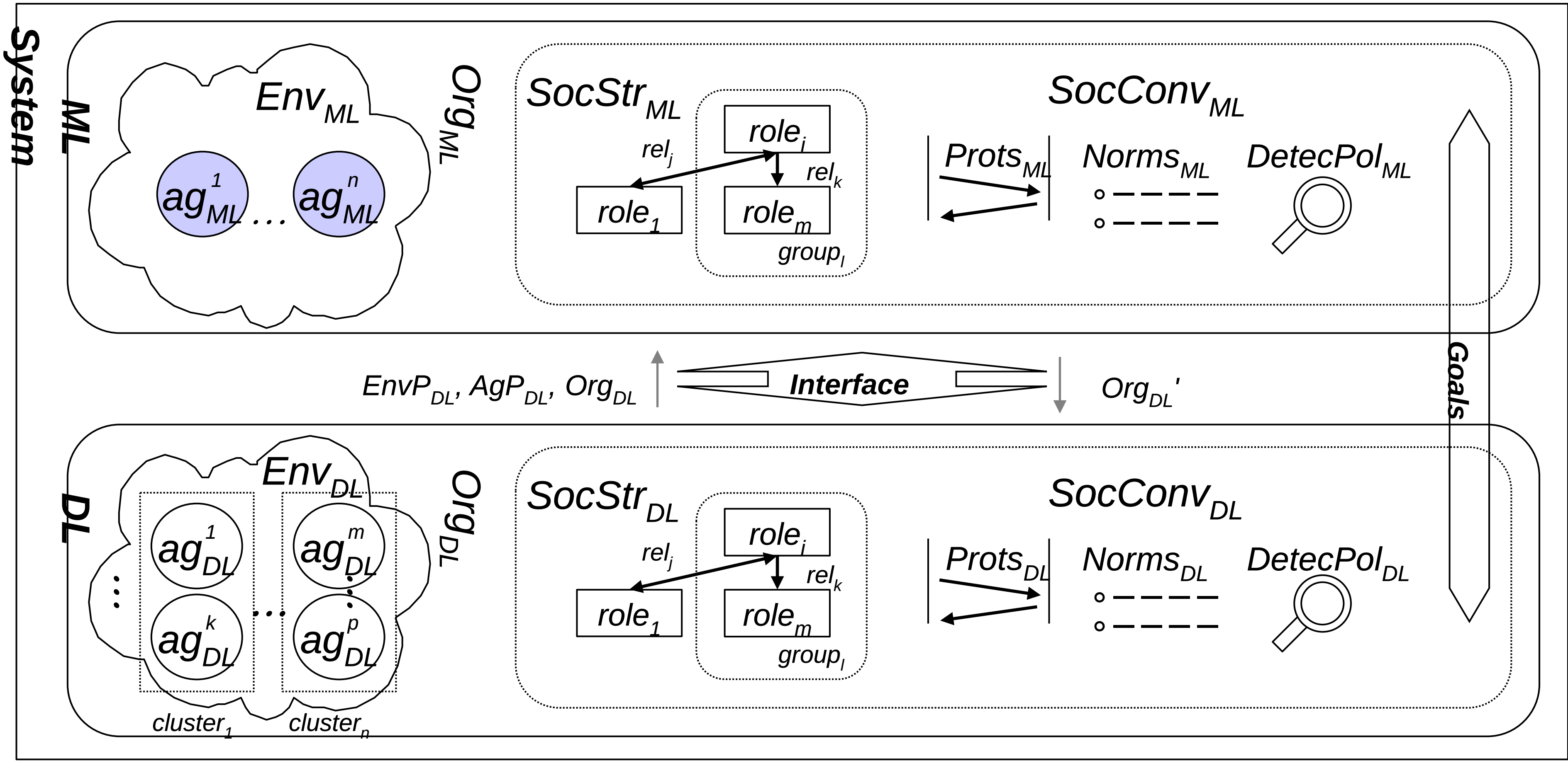
AOCMAS approach

Meta Level



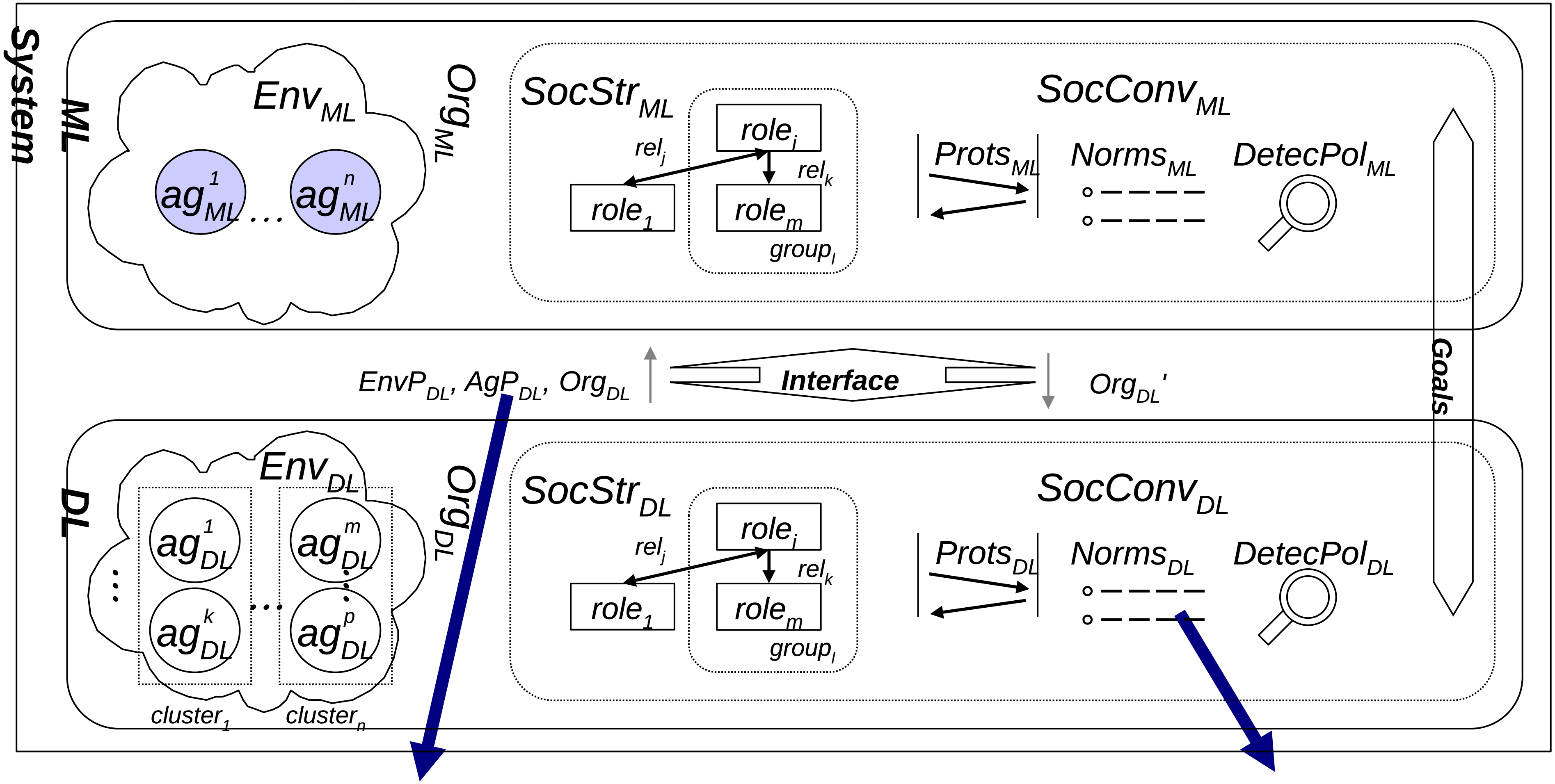
- and a **Meta-Level (ML)** = agents (*assistants*) organised to assist DL (eg. to adapt its org.)

2-LAMA



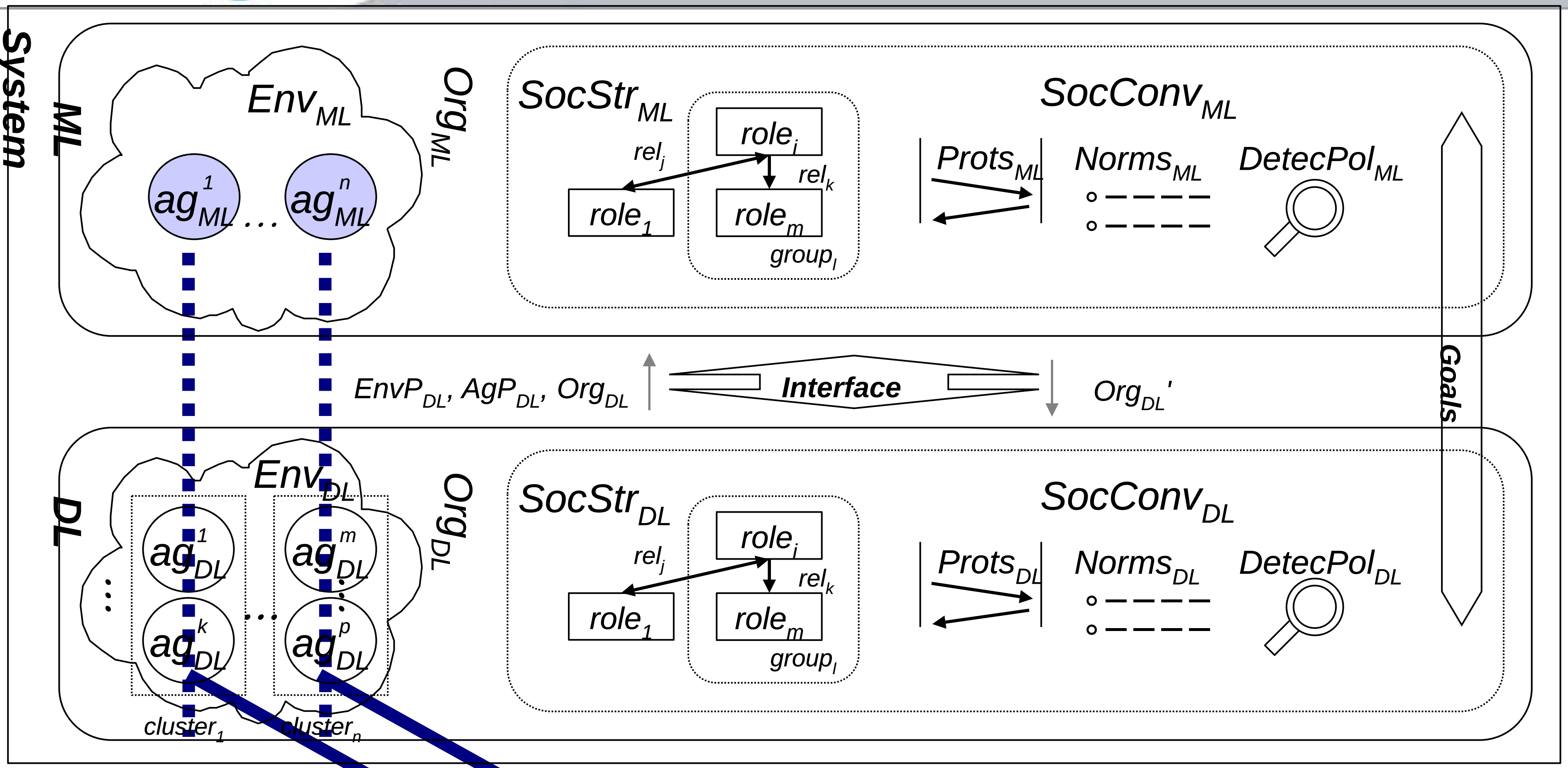
- Hence we call this abstract architecture:
  Two-Level Assisted MAS Architecture (2-LAMA)

2-LAMA: adaptation



- α^N: $EnvP_{DL} \times AgP_{DL} \times Norm_{DL} \times Goals_{DL} \rightarrow Norm_{DL}$
  Norm adaptation function

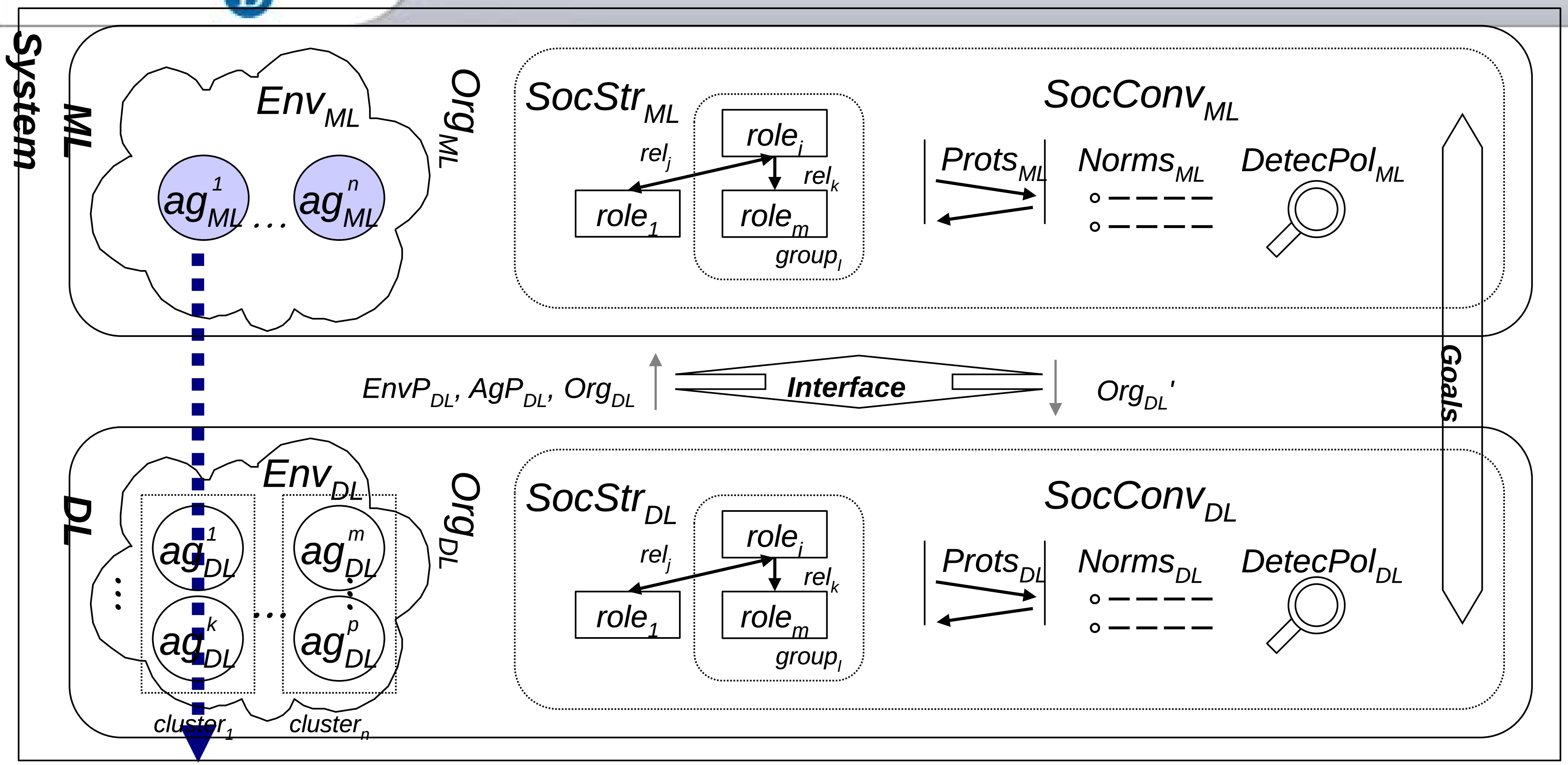
2-LAMA: distributed



- α^N: EnvP$_{DL}$ x AgP$_{DL}$ x Norm$_{DL}$ x Goals$_{DL}$ $\rightarrow$ Norm$_{DL}$
- $\alpha^N = \beta_{\alpha^N}$ ($\{\alpha_1^N..\alpha_n^N\}$) *agreement function*

2-LAMA: information

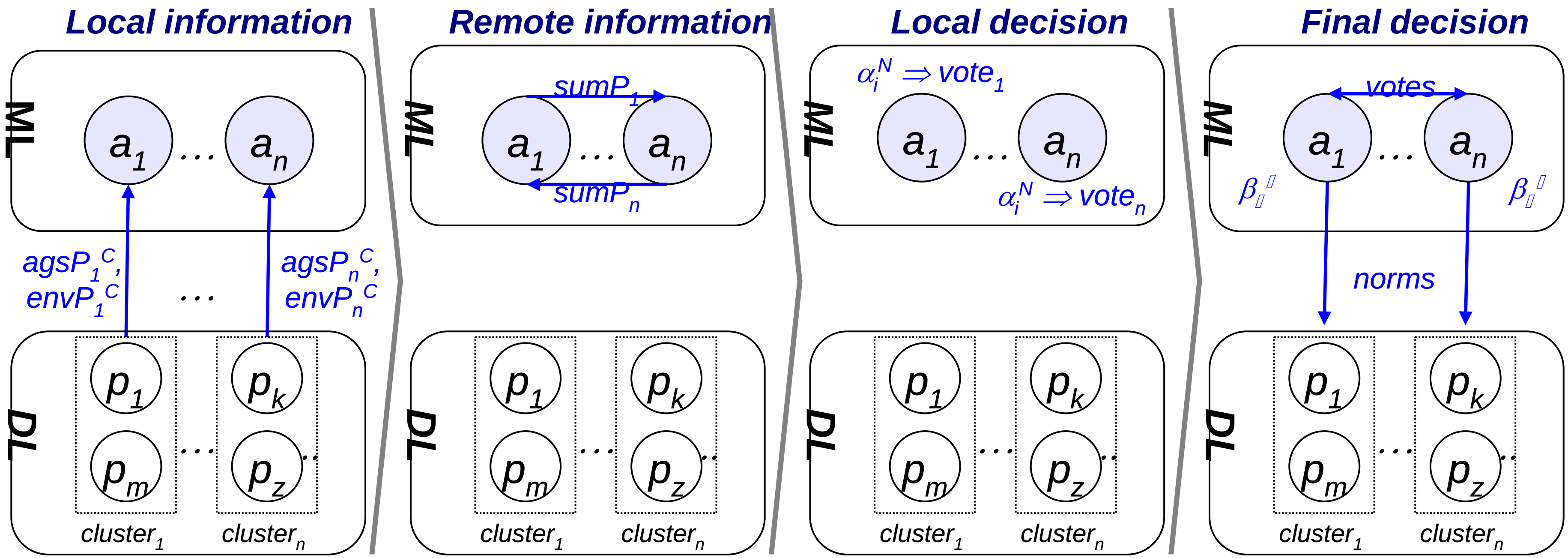
UNIVERSITAT DE BARCELONA



- α_i^N: $\underbrace{EnvP_i \times AgP_i}_{\text{Local}} \times \underbrace{(SumP_j)^{n-1}}_{\text{Remote}} \times Norms \times Goals \rightarrow Norms$

Information: *Local* *Remote = summaries of other local info.*

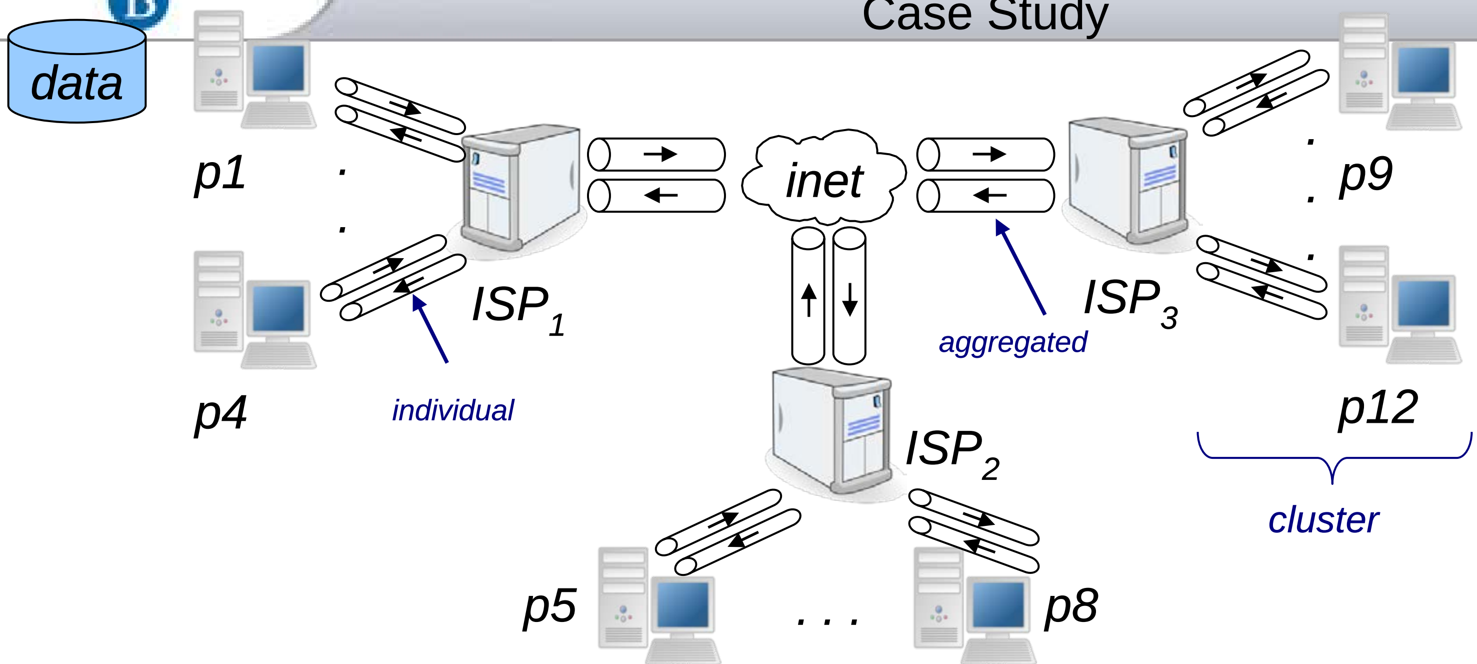
Adaptation steps



- adaptation frequency should keep the adapt. cost below the benefits it generates.
  - This cost depends on: information retrieval, computation, adoption and transition.

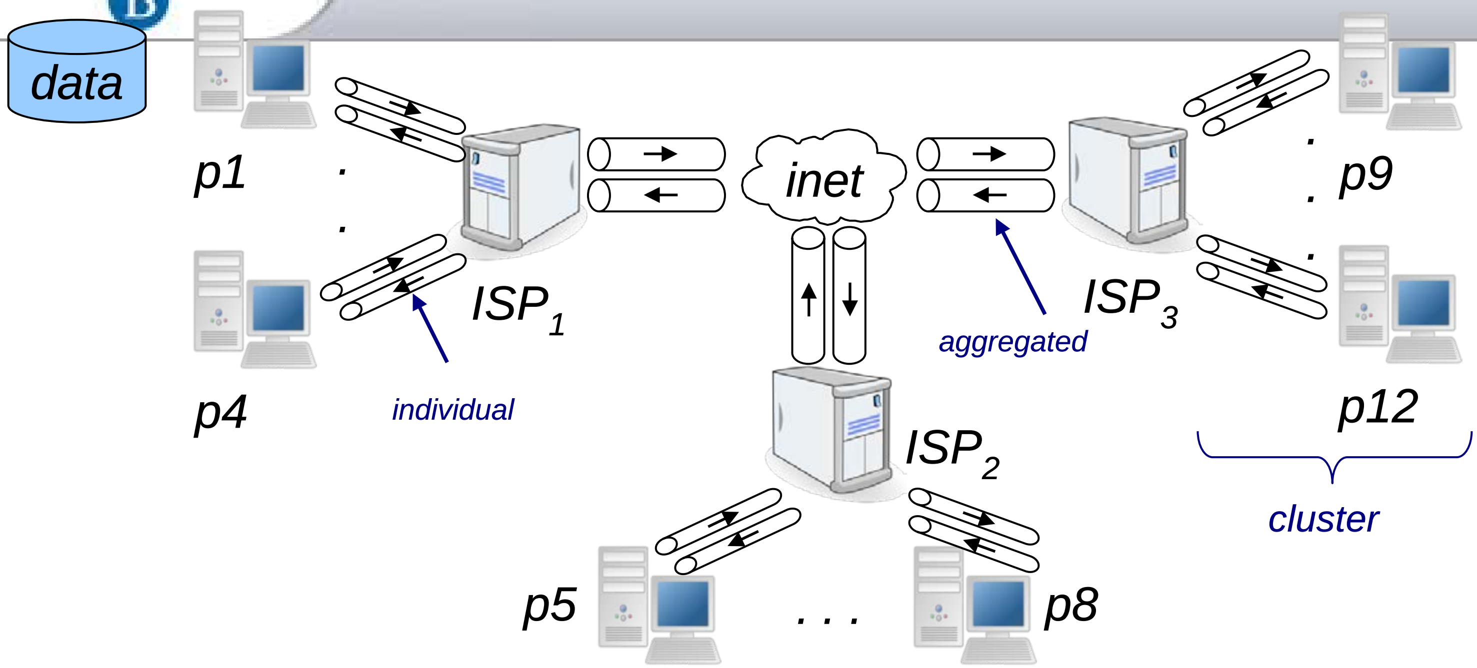
P2P sharing network

Case Study



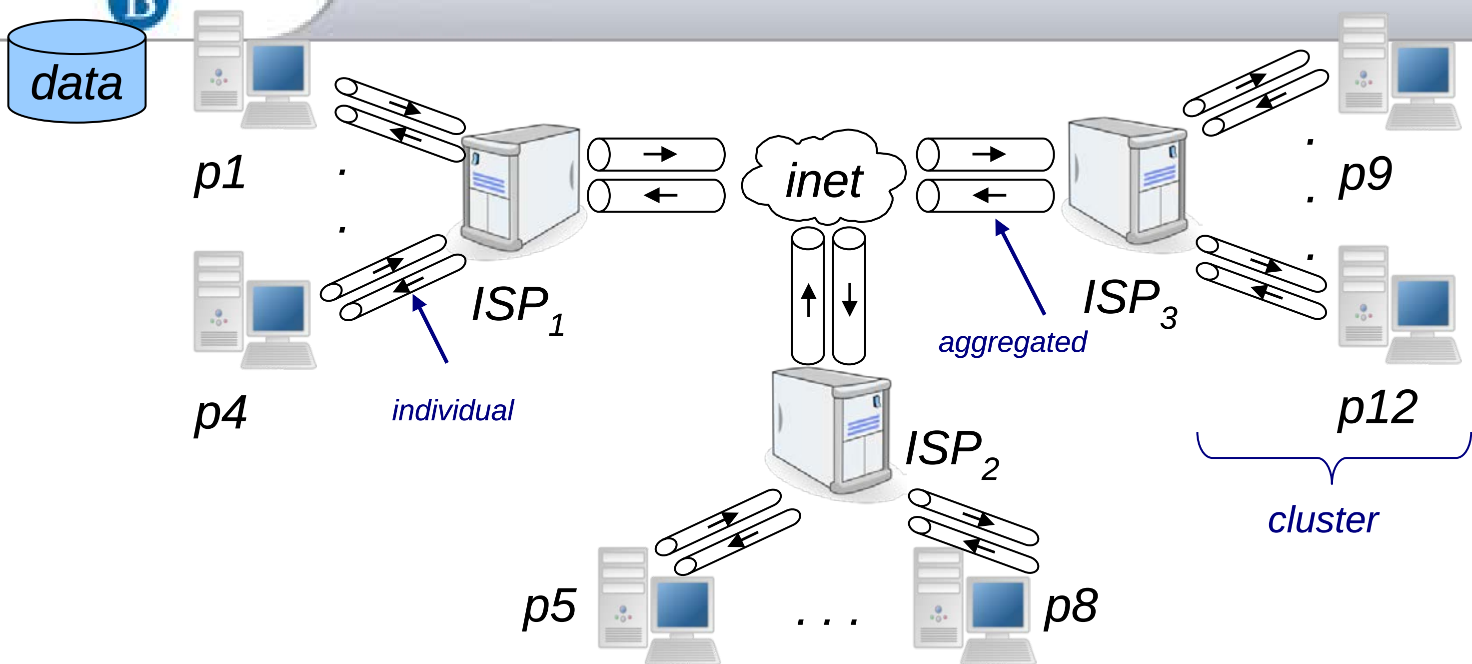
- P2P **data sharing network**:
  - **to share 1 piece of data** among all computers (*peers*) following a simplified version of the standard *BitTorrent* protocol, consuming the **minimum time** (*goal*).

P2P as an OCMAS

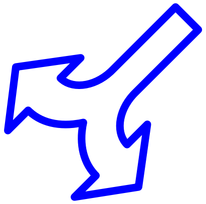


- **OCMAS** view:
  - Comput. = **Agents**
  - Net = **Environment**
  - Protocols, Social struc., Restrictions = **Org.**

P2P as an AOCMAS

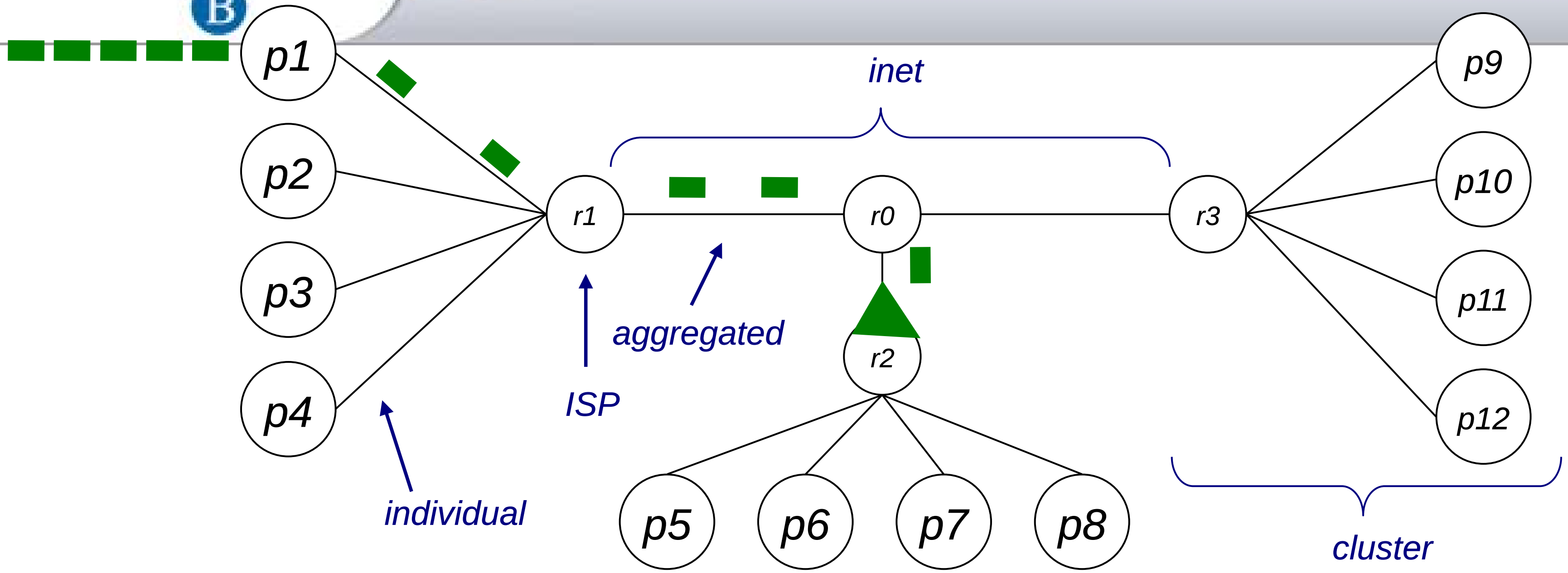


- **AOCMAS** view:
  - Comput. = **Agents**
  - Net = **Environment**
  - Protocols, Social struc., Restrictions = **Org.**



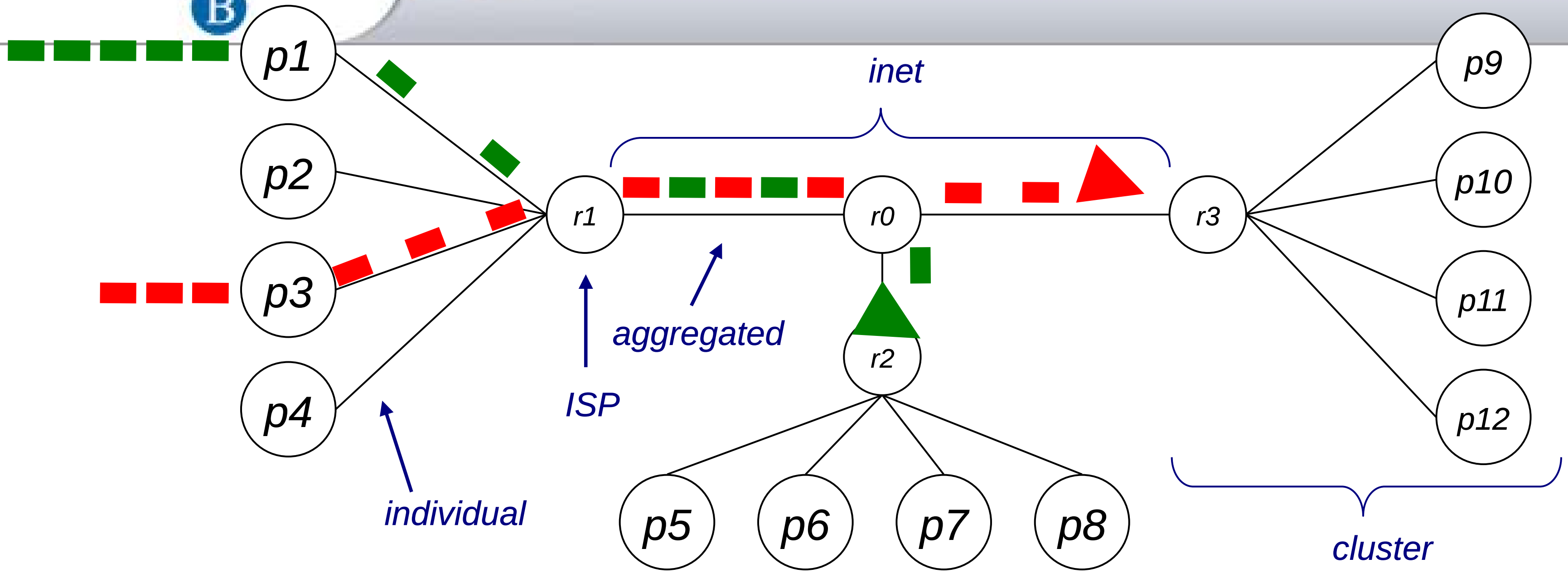
***Adaptation** to env./pop. changes may improve perf.*

Network abstraction



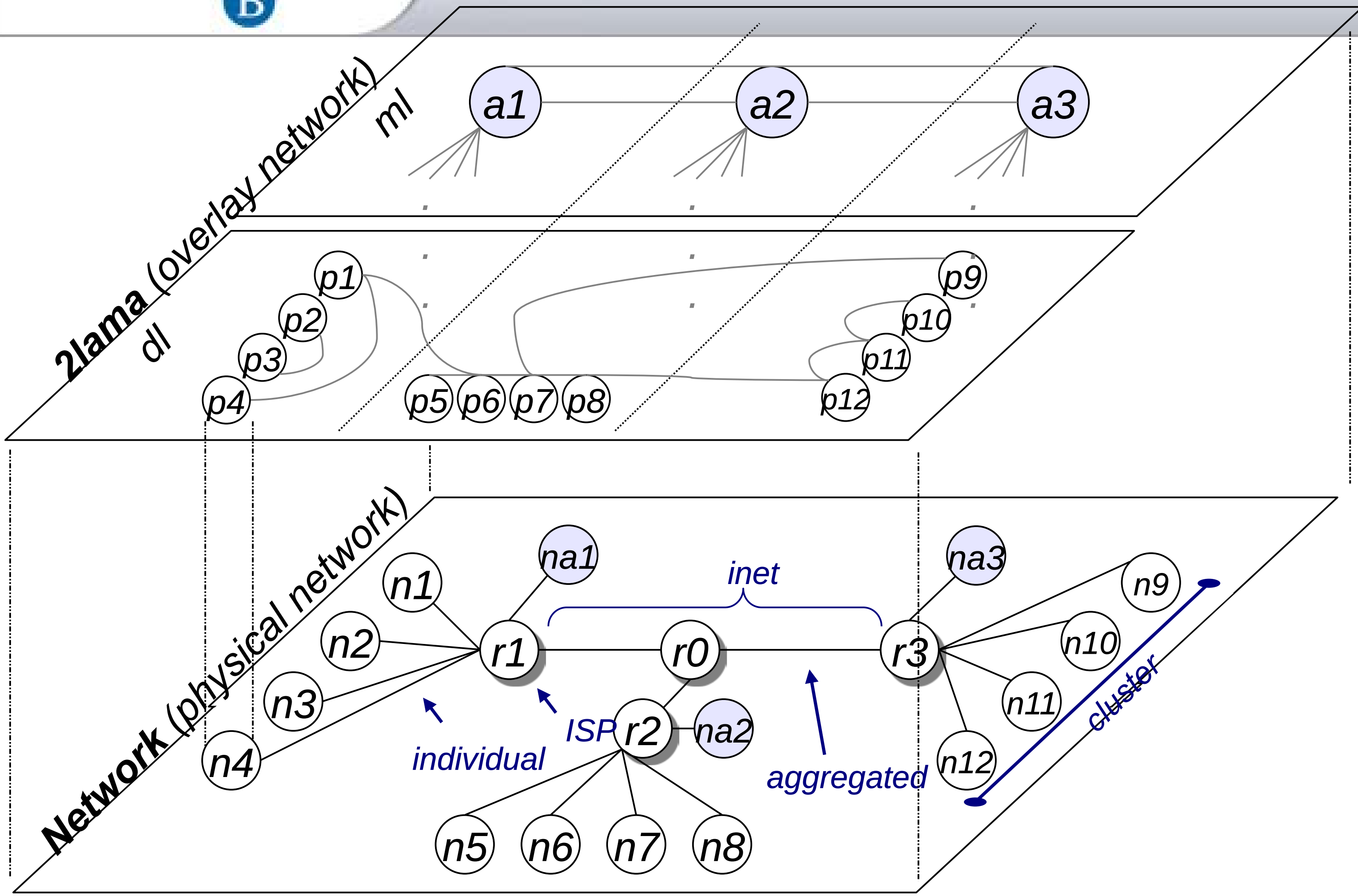
- **Network: packet switching transport**
  - Msgs split into packets that share links in time

Network abstraction

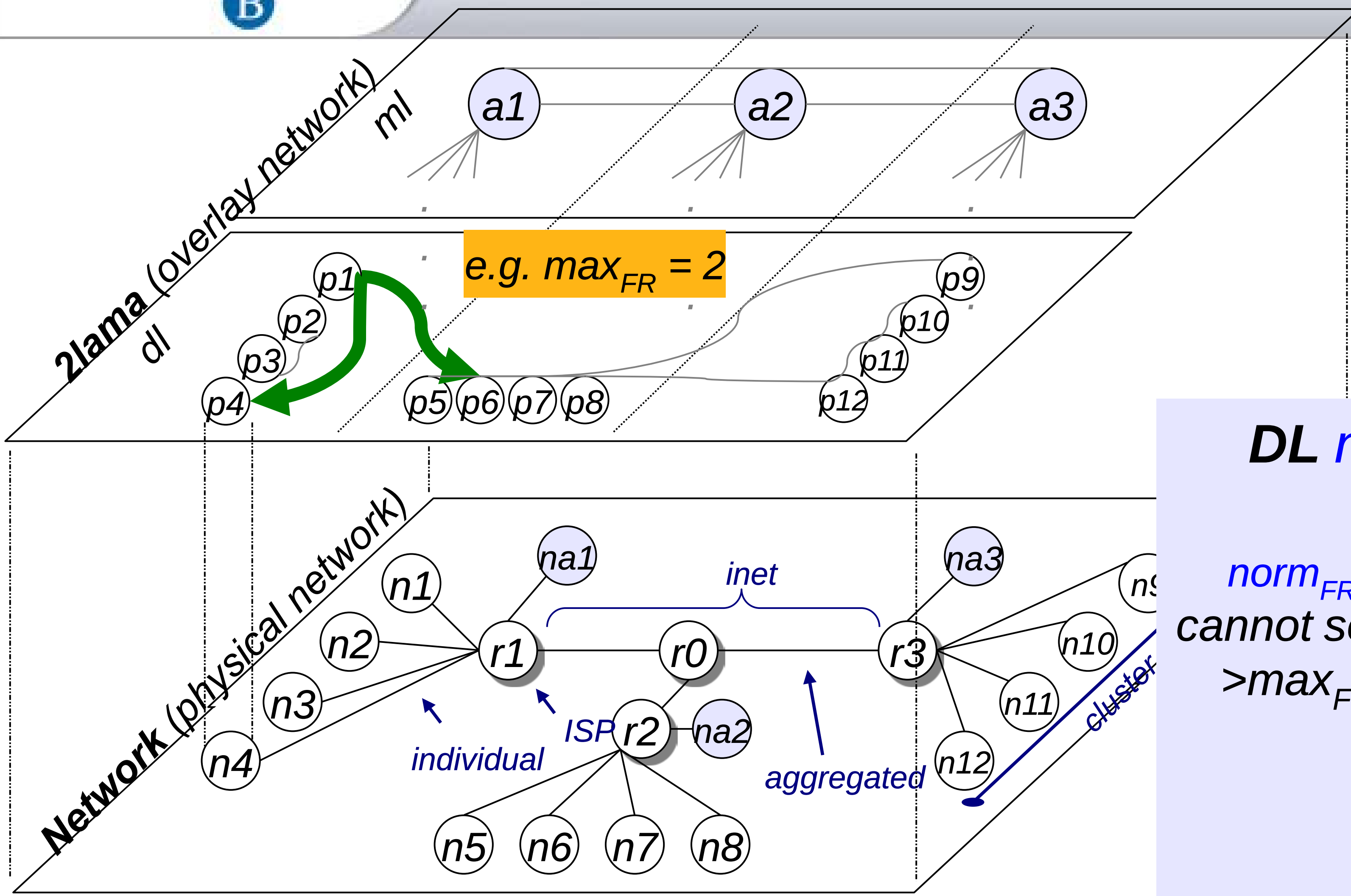


- **Network: packet switching transport**
  - Msgs split into packets that share links in time
    - $\mathbf{msg}_{\mathbf{latency}}$ = f(msg.length, #links, links.usage)

2-LAMA on P2P scenario



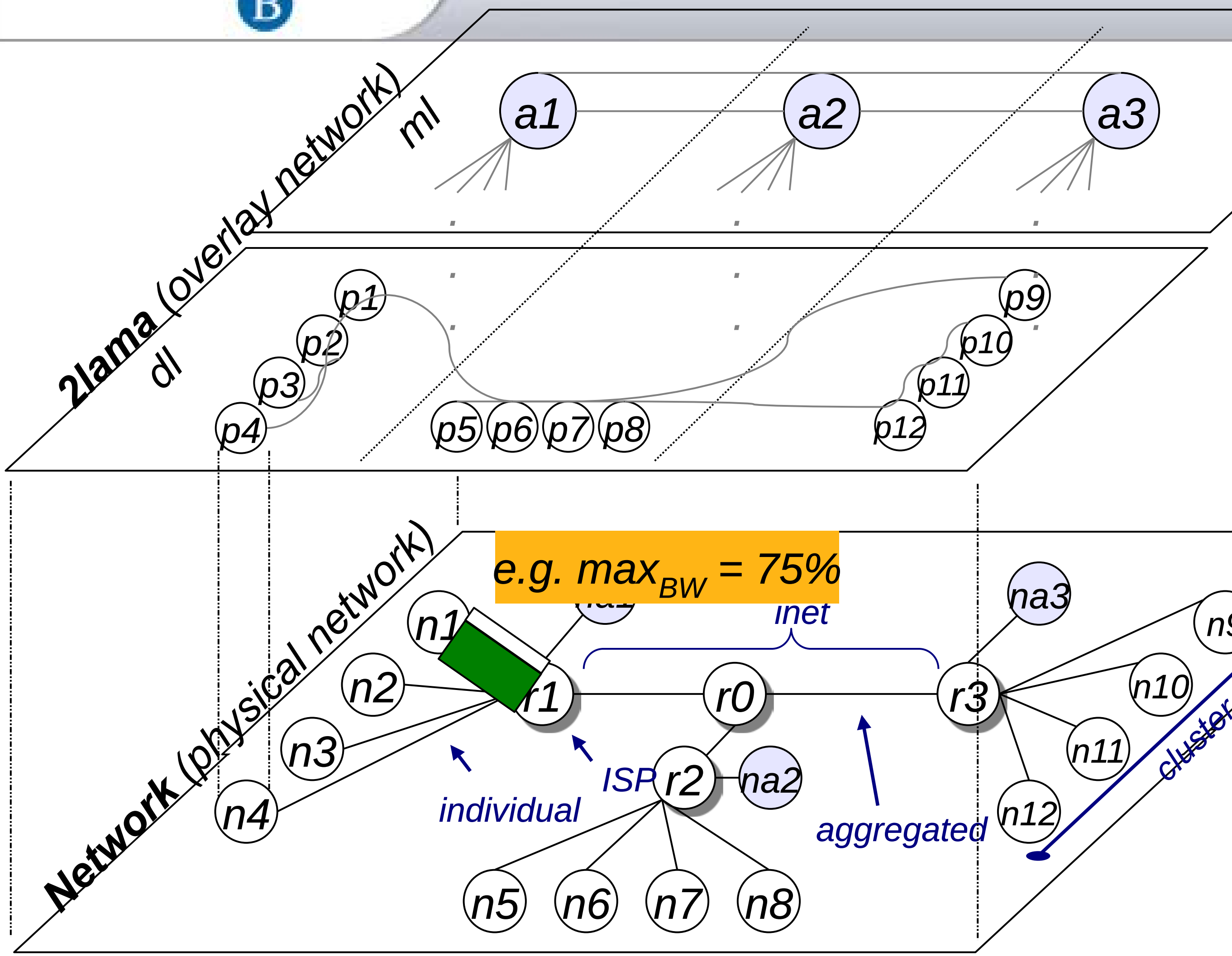
2-LAMA on P2P scenario



***DL** norms:*

$norm_{FR}$: "a peer cannot send data to $>max_{FR}$ simult."

2-LAMA on P2P scenario



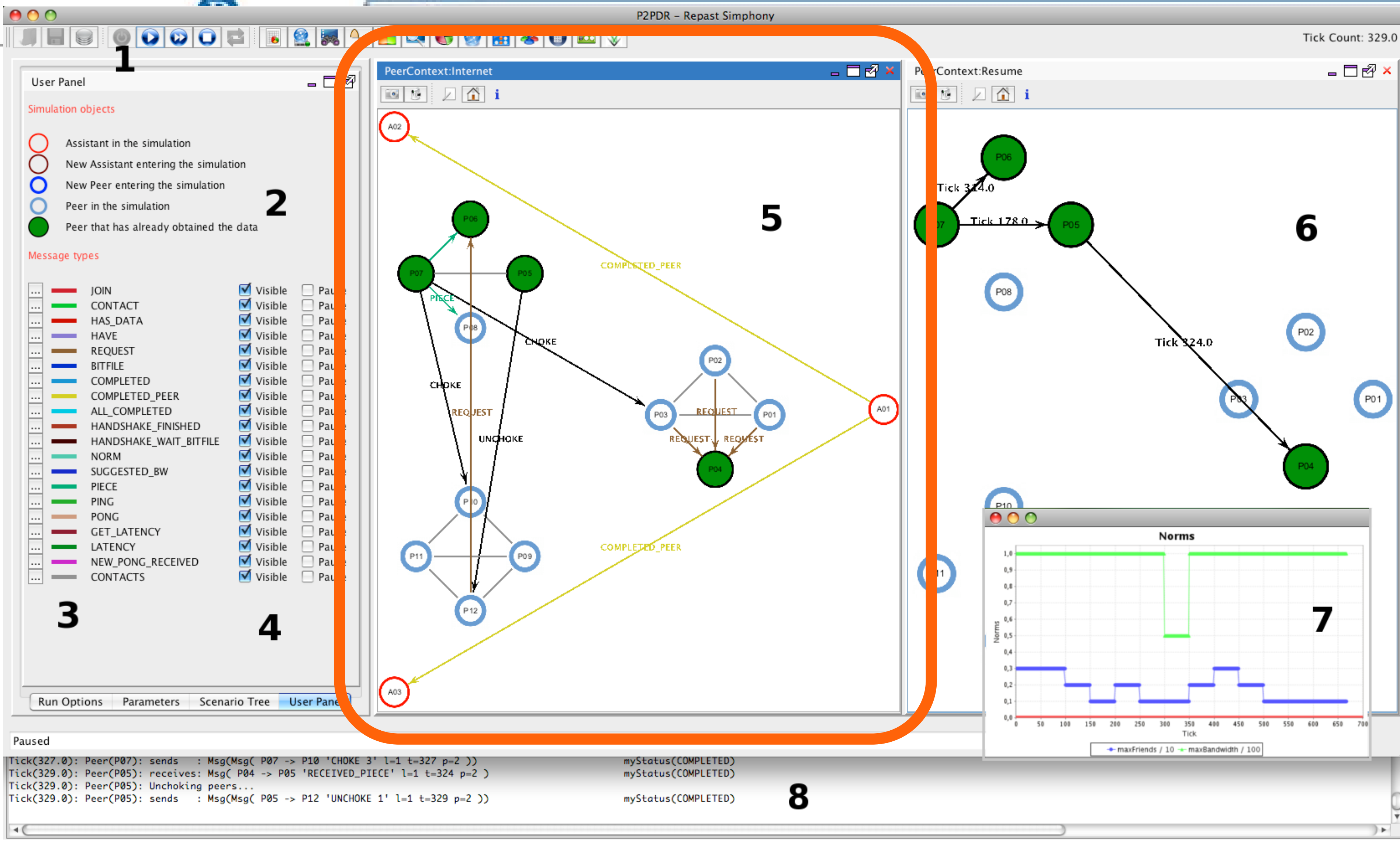
DL norms:

$norm_{FR}$: "a peer cannot send data to $>max_{FR}$ simult."

$norm_{BW}$: "a peer cannot use $>max_{BW}$ bandwidth."

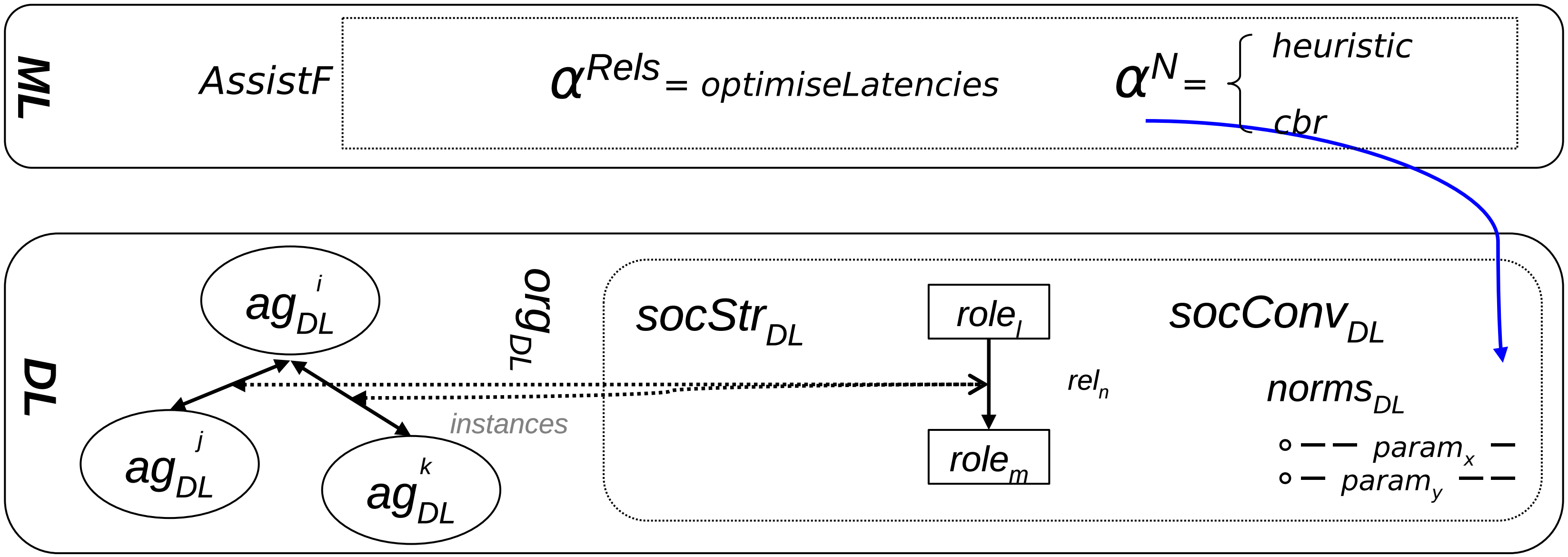
UNIVERSITAT DE BARCELONA

Simulator



Visual representation of Meta-Level and Domain-Level activity.

Adaptation Mechanisms



Norm Adaptation: CBR

ML	*AssistF*	$\alpha^{Rels} = optimiseLatencies$	$\alpha^{N} = \begin{cases} heuristic \\ cbr \end{cases}$

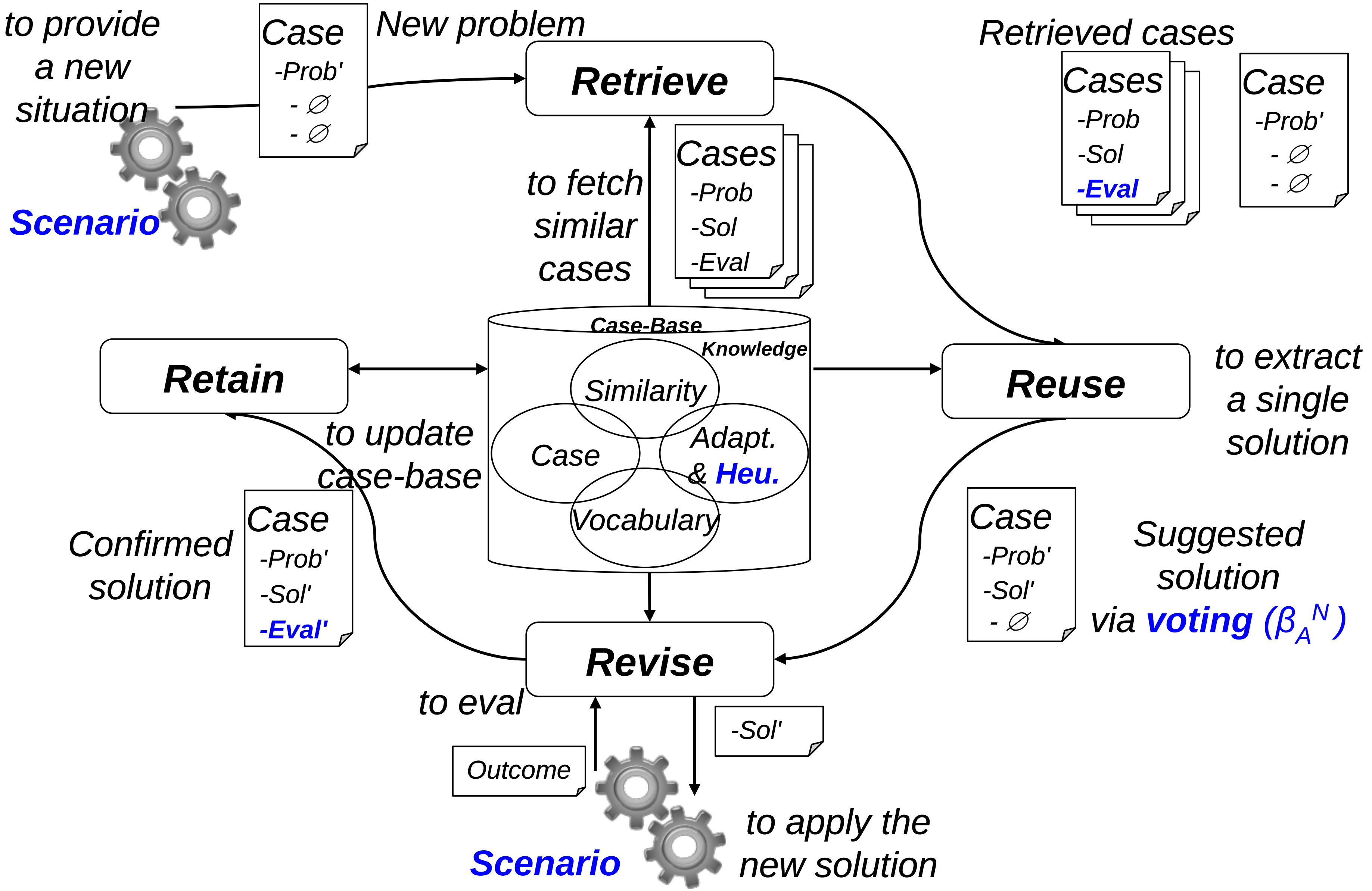
- $\alpha_i^N : \underbrace{KnowP \times Goals \times Norms}_{State} \underbrace{\rightarrow}_{relation} Norms$
- **Idea**: to use machine learning to learn this relation.
- **Issues**:
  - unknown appropriate norms
  - credit assignment problem
  - large state space

a tailored version of Case-Based Reasoning (CBR)

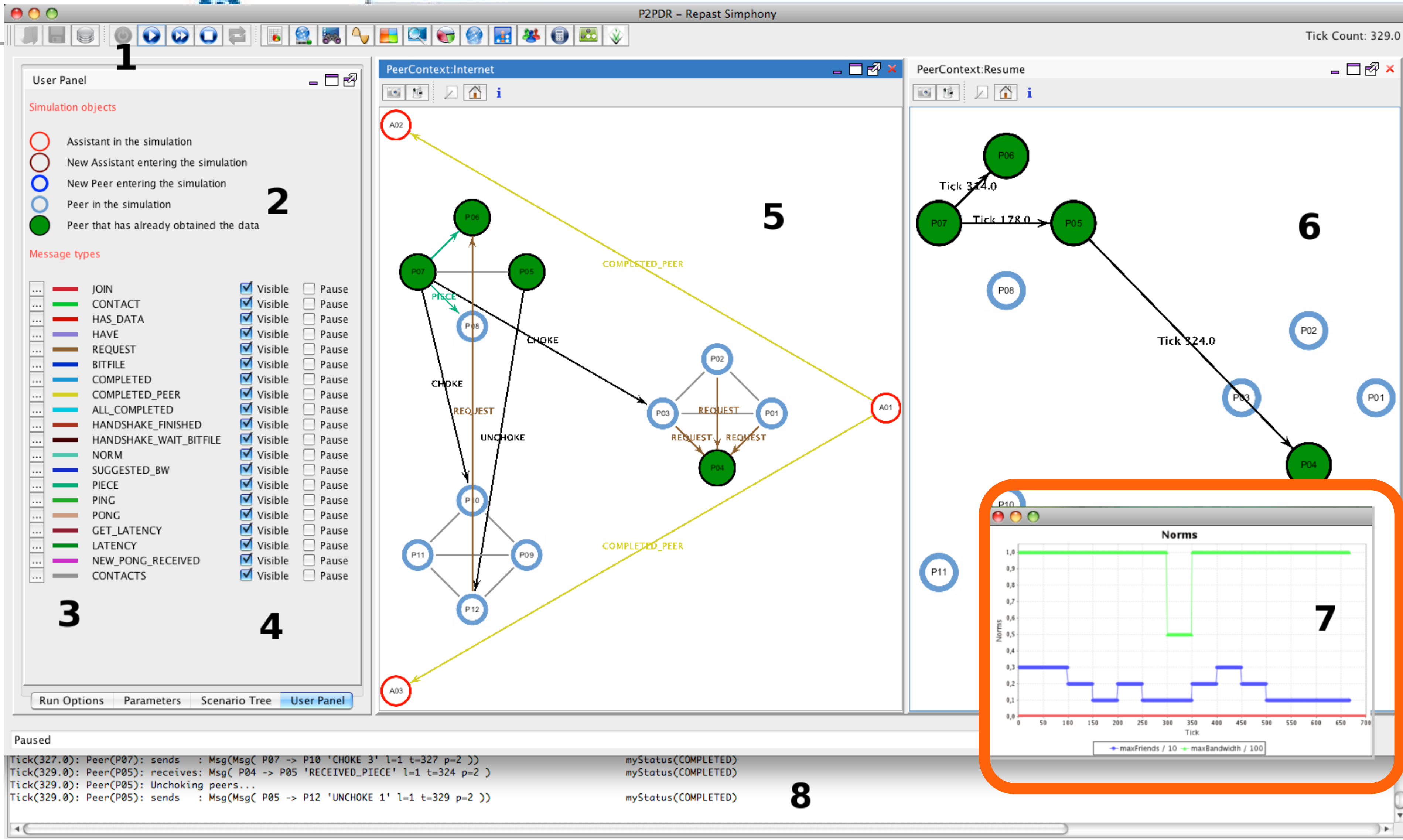
Tailored CBR: a case

- **Case**:
  - **Problem**:
    - *sharing state*: Completeness, Waiting
    - *comm. capacity:* SrvBW, RcvBW, RcvEffBW
    - *current norms*: OldMaxFR, OldMaxBW
    - *absolute values in order to normalise*: SeedBW, LeechBW, NumPeers
  - **Solution**:
    - *new norms*: NewMaxFR, NewMaxBW
  - **Evaluation**:
    - *effectiveness of the solution*: Goodness

Tailored CBR



Simulator

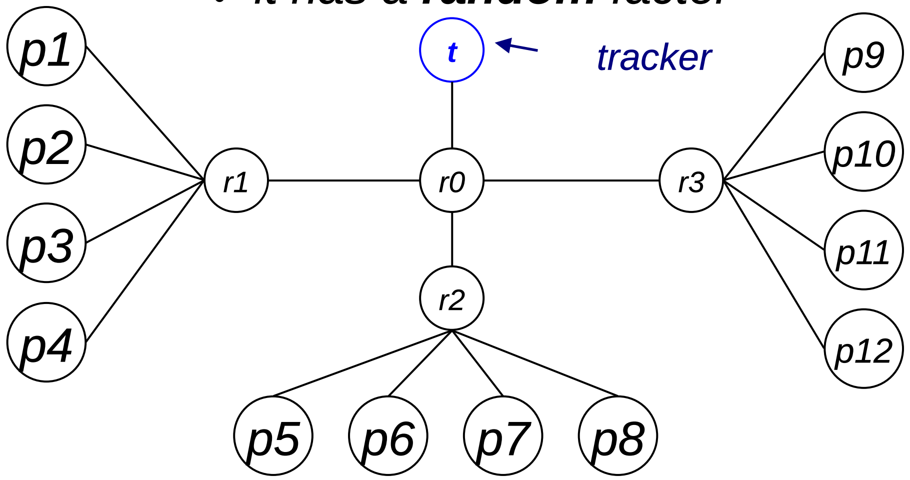


Visual exploration of norm evolution.

Coordination Models: BT

BT

- Name: ***BitTorrent*** *standard protocol (BT)*
- Type: ***non-adaptive*** *coordination model*
- Observations:
  - *we use it as a* ***base-line***
  - *it has a single* ***Tracker*** *(~ directory service)*
  - *it has a* ***random*** *factor*



Coord. Models: 2L-CBR

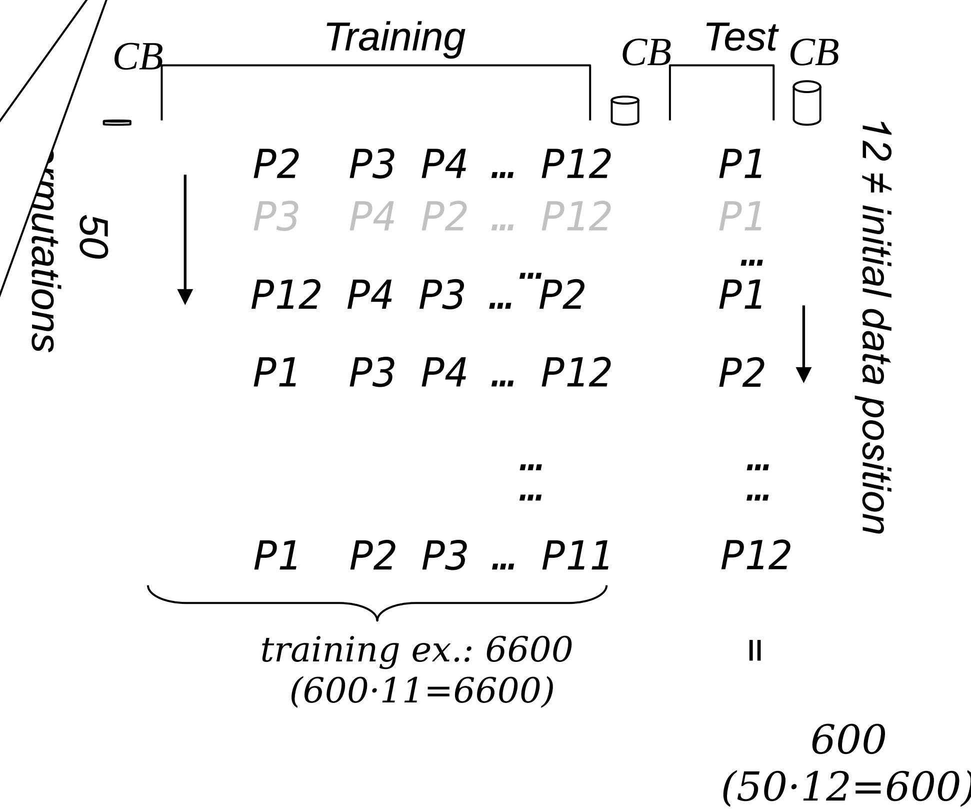
BT | **2L-S** | **2L-S-N-Heu** | **2LAMA-CBR**

- Name: ***2-LAMA*** *with social relationships adaptation and* ***Norm adaptation*** *using a* ***CBR*** *approach (2L-S-N-CBR)*

-Type: ***adaptive*** *coord. model*

Observations:

- ***dynamic adaptation*** *method*
  - *requires* ***training***
    - *the* ***order*** *of initial* ***data positions*** *during the training is* ***relevant***



#execs.

Results

	BT	2L-S	2L-S-N-Heu	2LAMA-CBR
time	986.2	793.1	744.5	732.6
net	206,592.0	338,448.3	306,755.0	348,399.6
netSat	0.177	1.813	2.126	1.846

*2LAMA-CBR is **the fastest**. results statistically significants*

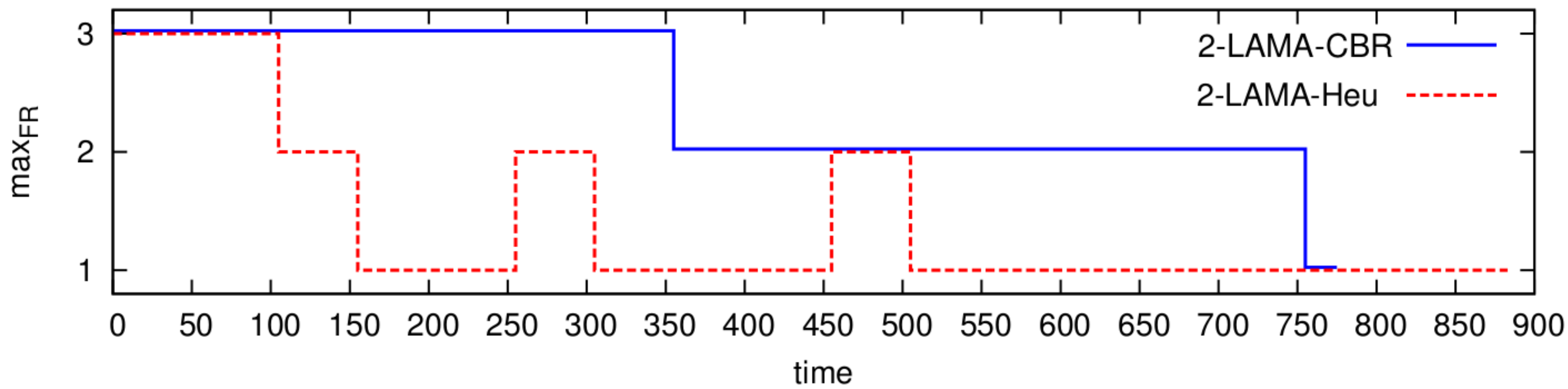
*2LAMA-CBR is **uses the network most**.*

*2LAMA-CBR learns to use intensively the network **without** achieving **too much saturation**.*

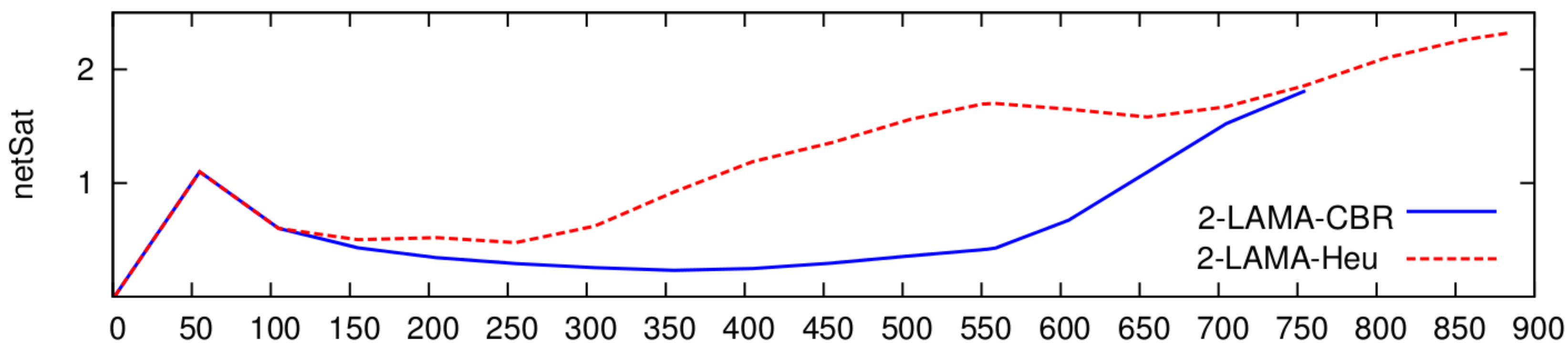
Results

Particular example (datum in p3):

(a) max_{FR} when the data is initially in p3



(b) netSat when the data is initially in p3

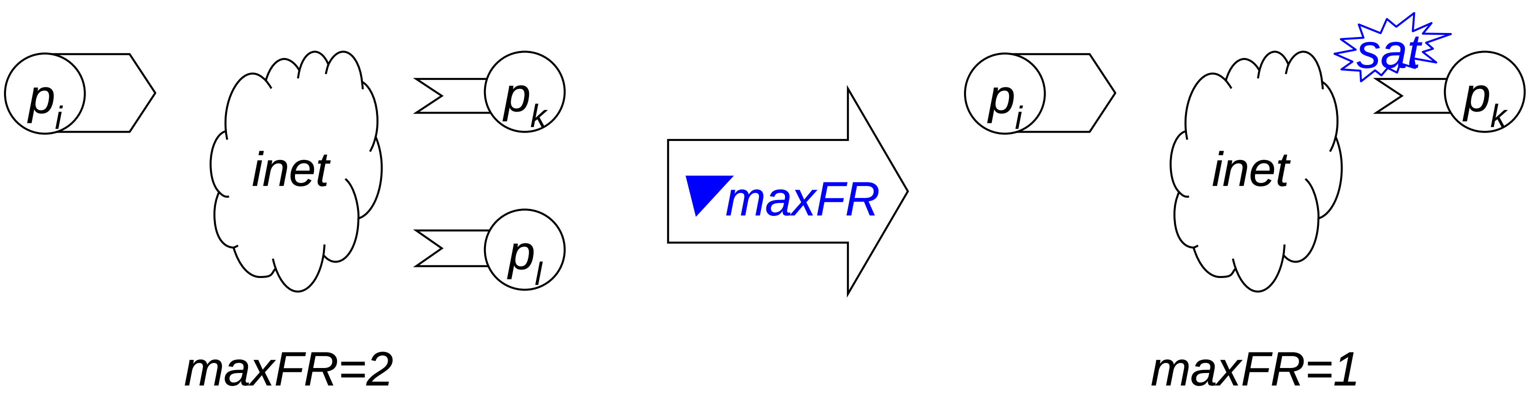


2LAMA-CBR learns to use intensively the network ***without*** *achieving* ***too much saturation****.*

Results

CBR learns: ▼maxFR does not always imply ▼netSat.

Example:
If a source is serving to some agents with smaller BW, ▼maxFR may saturate receivers' individual links.



*2LAMA-CBR learns to use intensively the network **without** achieving **too much saturation**.*

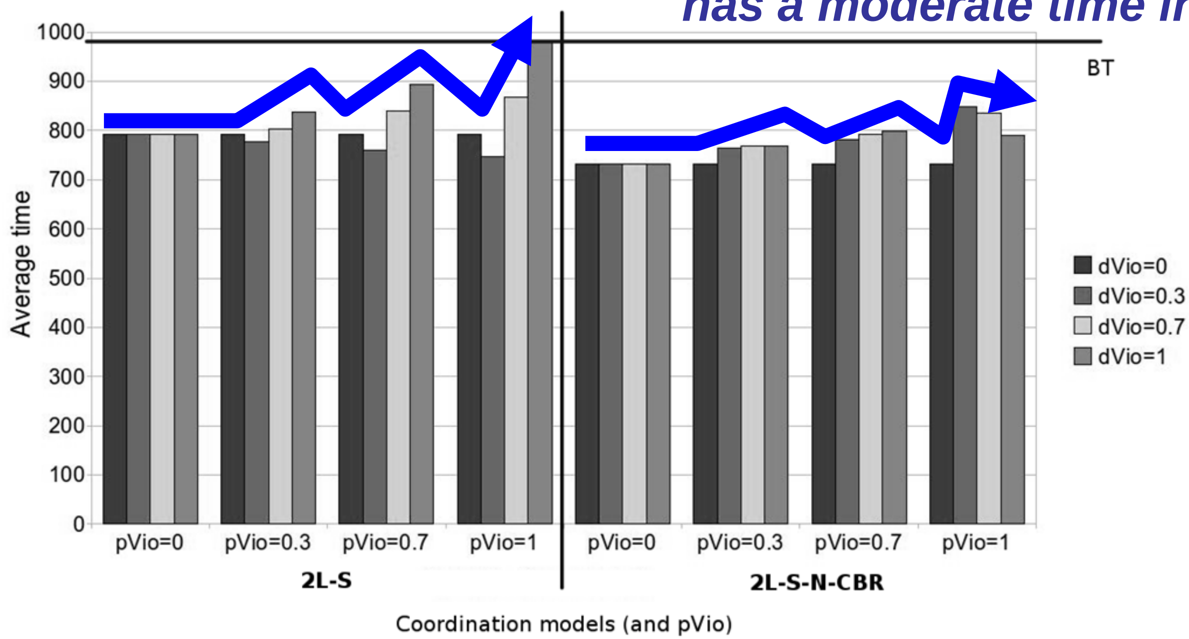
Exploring Open MAS issues

- Preliminary results about:
  - Entering / Leaving agents
  - Norm violations

	BT	2L-S	2L-S-N-Heu	2LAMA-CBR
time	1220.7	847.9	822.8	805.5

2LAMA-CBR is still the fastest

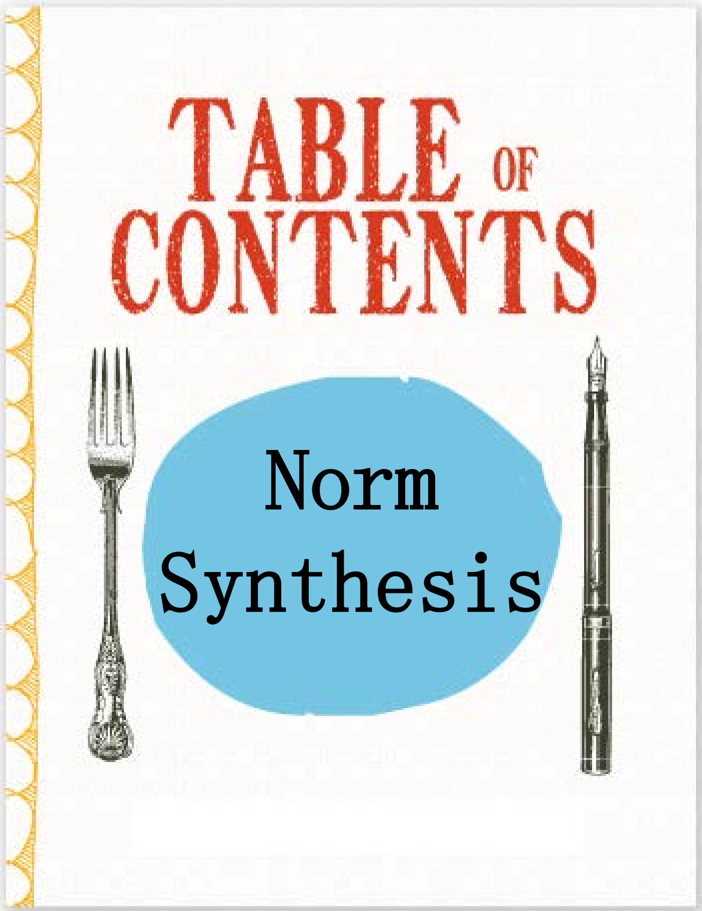
has a moderate time increment



2-LAMA is robust in Open MAS contexts.

Tutorial Outline

Contents: Modules



1. Introduction to Norms and Normative MAS.
2. Overview of approaches to norm synthesis.
   - Off-line norm synthesis.
   - ...
3. **On-line automatic norm synthesis.**
4. Demo and hands-on activity

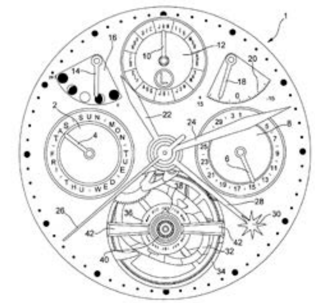
Norm set-up

Related Work: Open Challenges

- Norm synthesis: (Shoham & Tennenholtz)
  - Formal, exhaustive, **NP-complete**
  - Disallow (& ensure) access to undesirable (& goal) states in the state space

Norm set-up

Related Work: Open Challenges



- Norm synthesis: (Shoham & Tennenholtz)
  - Formal, exhaustive, **NP-complete**
  - Disallow (& ensure) access to undesirable (& goal) states in the state space

- Norm agreement: (Artikis et al.)
  - Democratic, convergence
  - Agents **enriched** with agreement capabilities.

Norm set-up

Related Work II: Open Challenges

- Norm emergence: (Conte, Sen, Villatoro,...)
  - Ex: driving on the left/right
  - Convergence (**initial conditions**)
  - Agents choose a solution from a space of alternative solutions (**known at design time**)
    - Repeated two-player games
    - Topology of relationships
    - Observation for norm adoption / Internalisation

UNIVERSITAT DE BARCELONA

Norm set-up

A proposal

- **Automatic norm generation**
  - Regulatory agents propose norms to avoid conflicts in agent interactions
    - Requires conflict detection
    - Does not search the complete state space
  - Norm evaluation based on
    - Agent responses (violations and compliances)
    - Consequences (conflicts)

Norm Generation

A proposal

Norm Generation

A proposal

Norm Generation

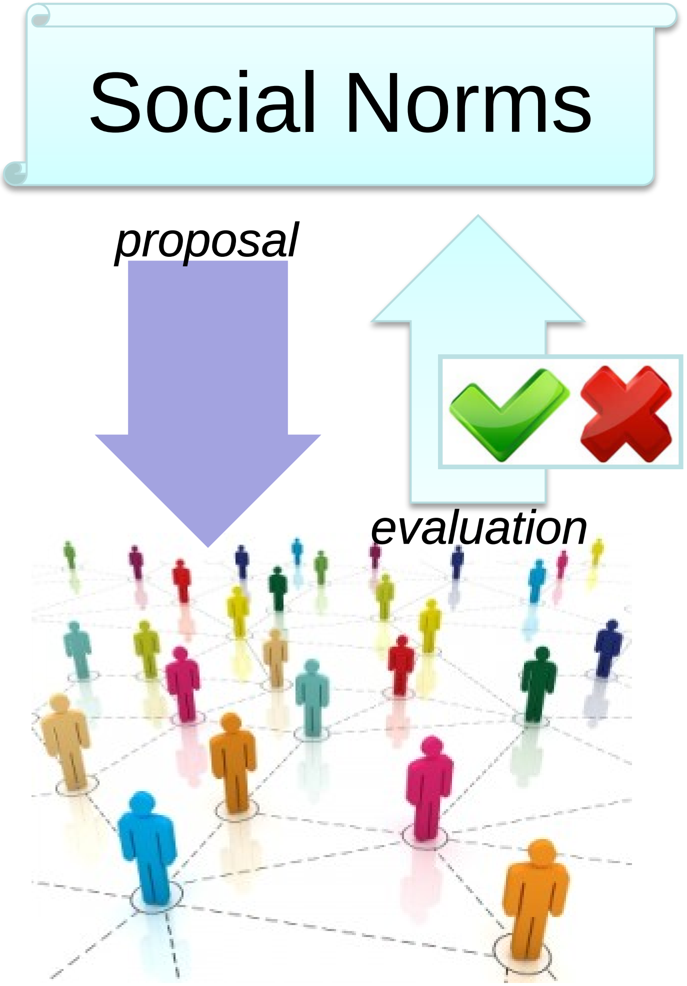
A proposal

Social Norms

proposal

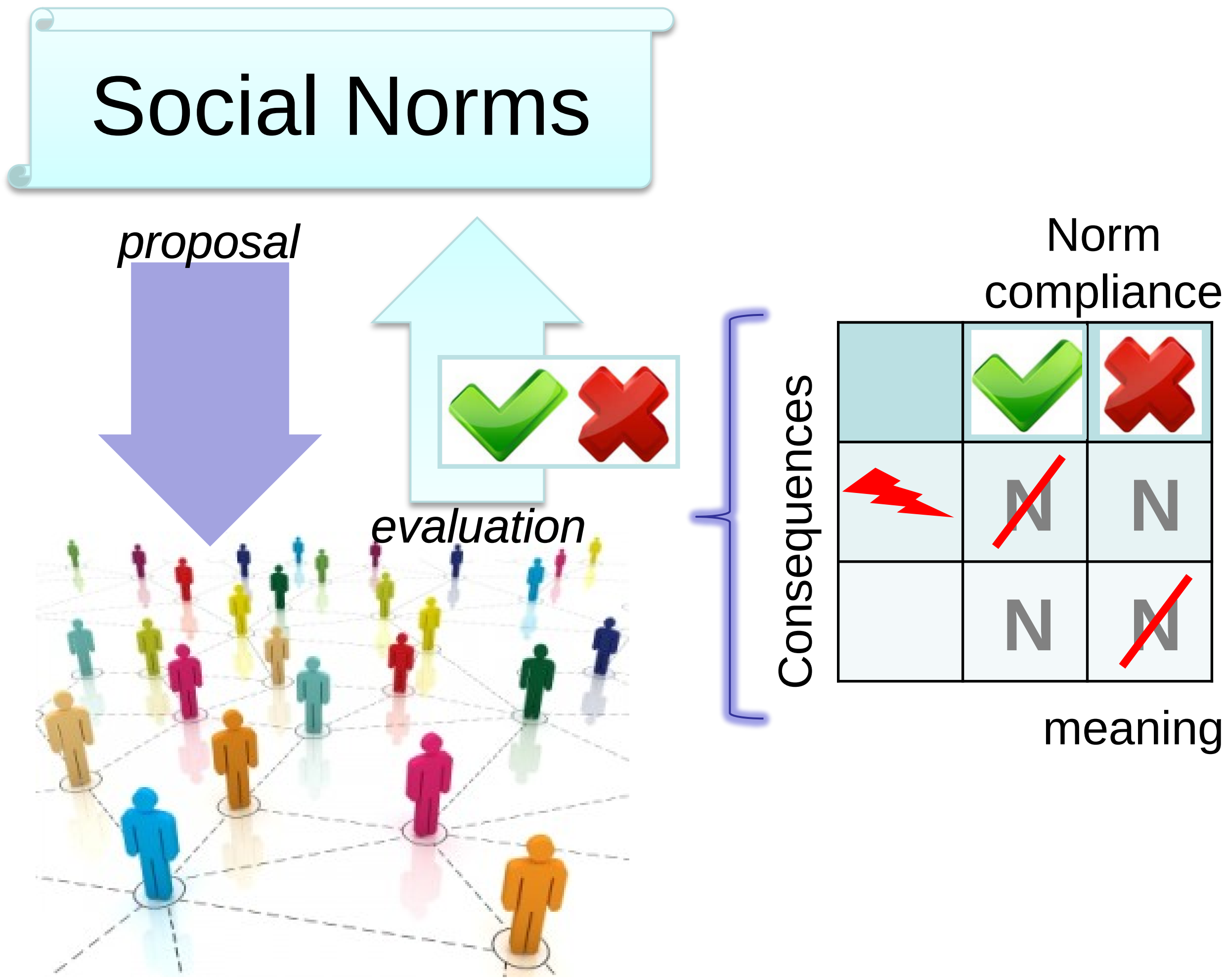
Norm Generation

A proposal



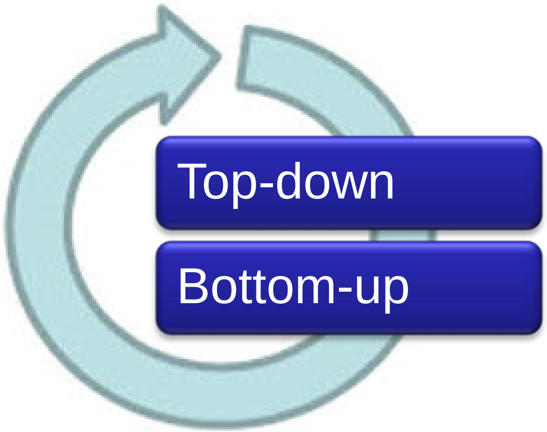
Norm Generation

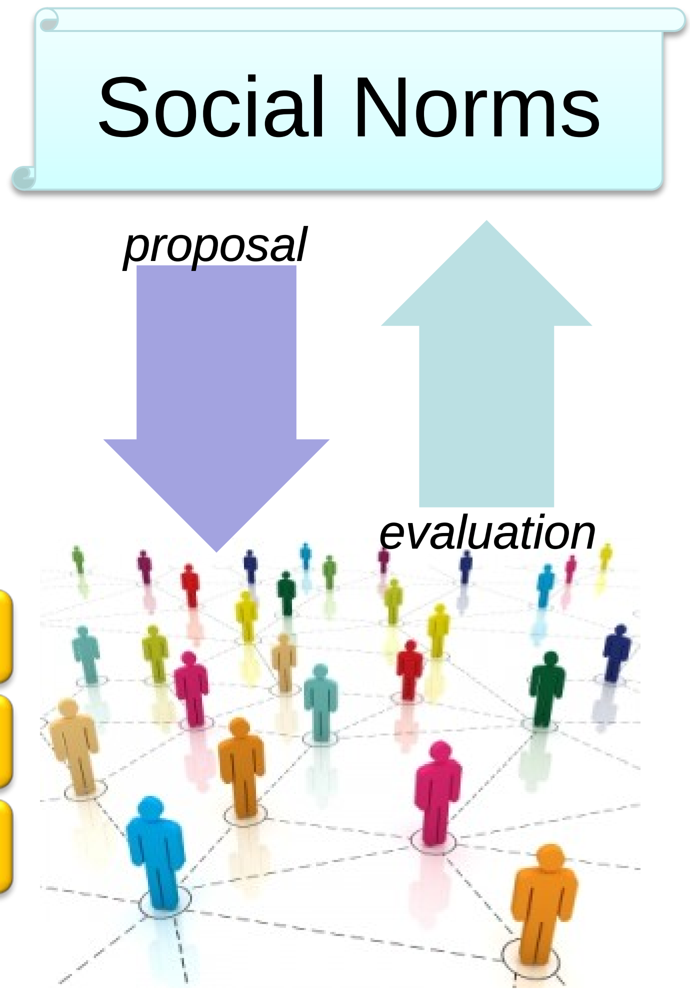
A proposal



Norm Generation

A proposal



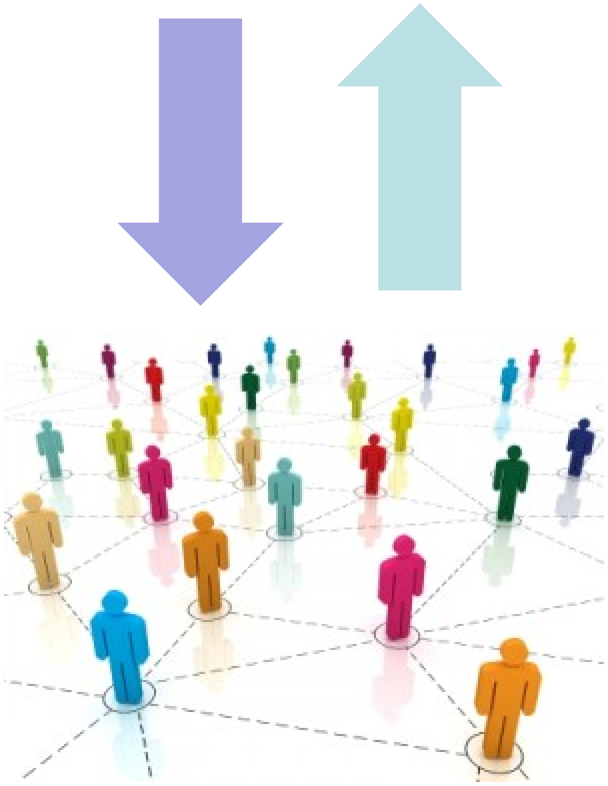


- Goal: conflict avoidance
- Dynamicity
- Division of concerns

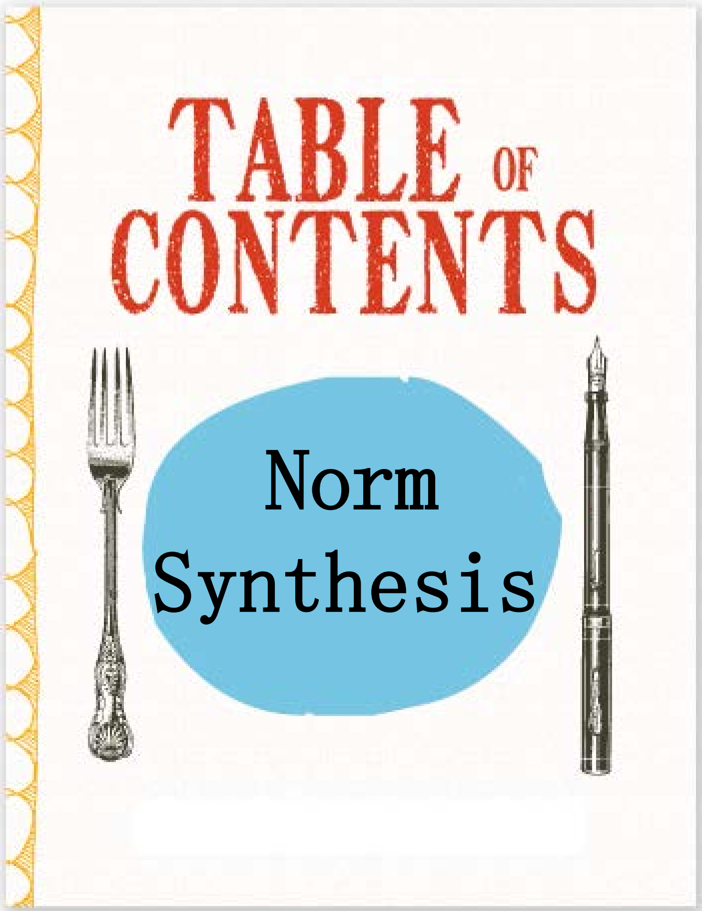
Presentation

Our claim

- Non-intrusive, autonomy preserving, norm generation mechanism
  - Norm quality measured based on
    - System objectives
  - Norm compliance/violation evaluated in terms of
    - System Objectives
    - (No prescribed penalties)
  - General system objective:
    - Avoid conflicts



Tutorial Outline



1. Introduction to Norms and Normative MAS.
2. Overview of approaches to norm synthesis.
3. **On-line automatic norm synthesis.**
   - **AAMAS 2013**.
   - AAMAS 2014
4. Demo and hands-on activity

Automated Synthesis of Normative Systems

Javier Morales, Maite López-Sánchez, Juan A. Rodríguez-Aguilar, Michael Wooldridge, Wamberto Vasconcelos

ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE - IIIA CSIC
UNIVERSITAT DE BARCELONA
UNIVERSITY OF OXFORD
UNIVERSITY OF ABERDEEN

1. Introduction

Individuals within a society
continuously interact →
Conflicts raise naturally

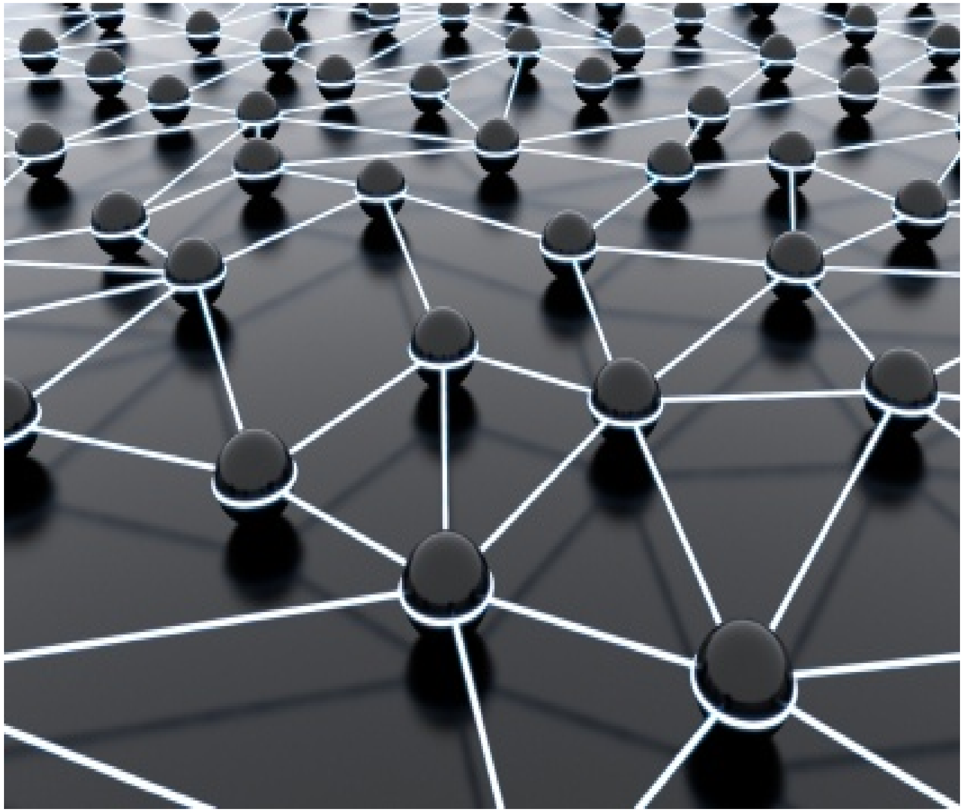
1. Introduction

Individuals within a society continuously interact → Conflicts raise naturally

Human societies avoid undesirable situations (i.e., conflicts) by including regulations.

1. Introduction

Likewise human societies, we can avoid conflicts in a Multi-Agent System (MAS) by including regulations.



1. Introduction

Possible approach: **Off-line** norm design.

1. Generate all system states (off-line).
2. Identify undesired system states.
3. Synthesise norms to avoid undesired states.

Not feasible for Open Multi-agent Systems

1. Large scenario → Impossible to generate all system states.
2. Uncertainty → We do not know agents' behaviour
3. Dynamic systems → System changes along time.

2. Research problem and approach

Research problem: How to synthesise a normative system that helps a MAS to avoid undesirable states (conflicts)?

Assumption: Uncertainty about the MAS composition and agents' behaviour → We cannot design the normative system *off-line*.

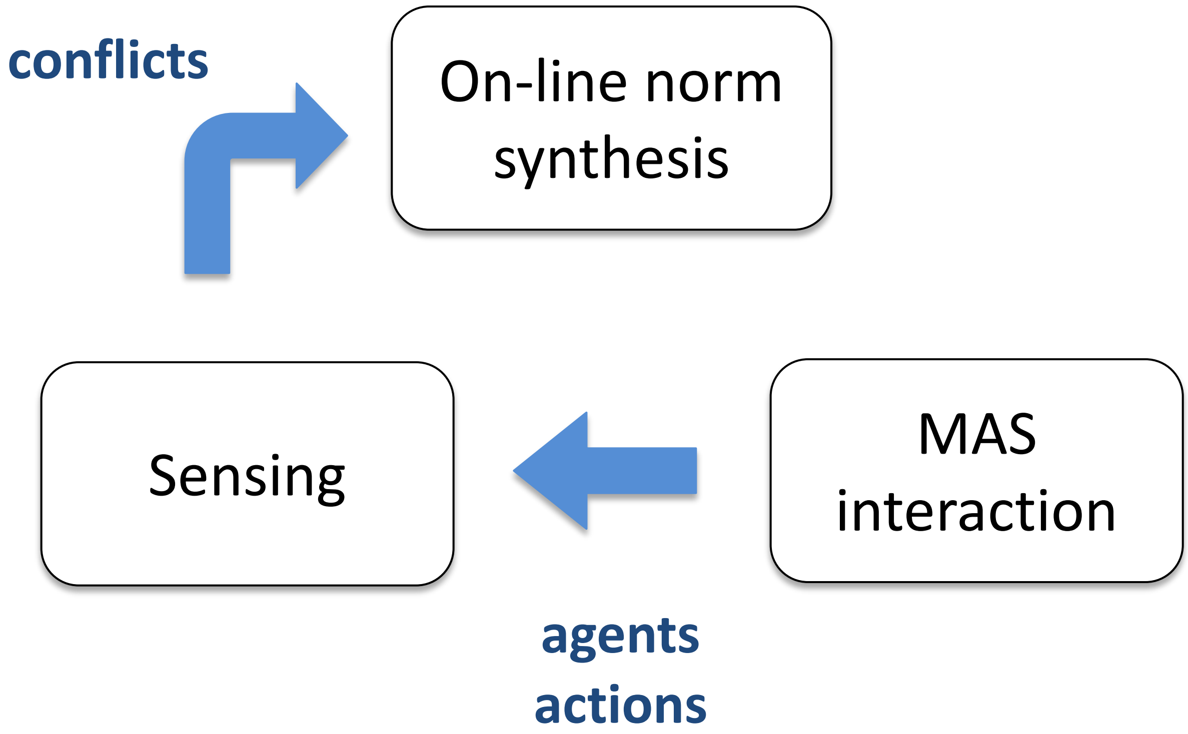
Our approach: An *on-line* mechanism for the automated synthesis of normative systems for MAS.

2. Research problem and approach

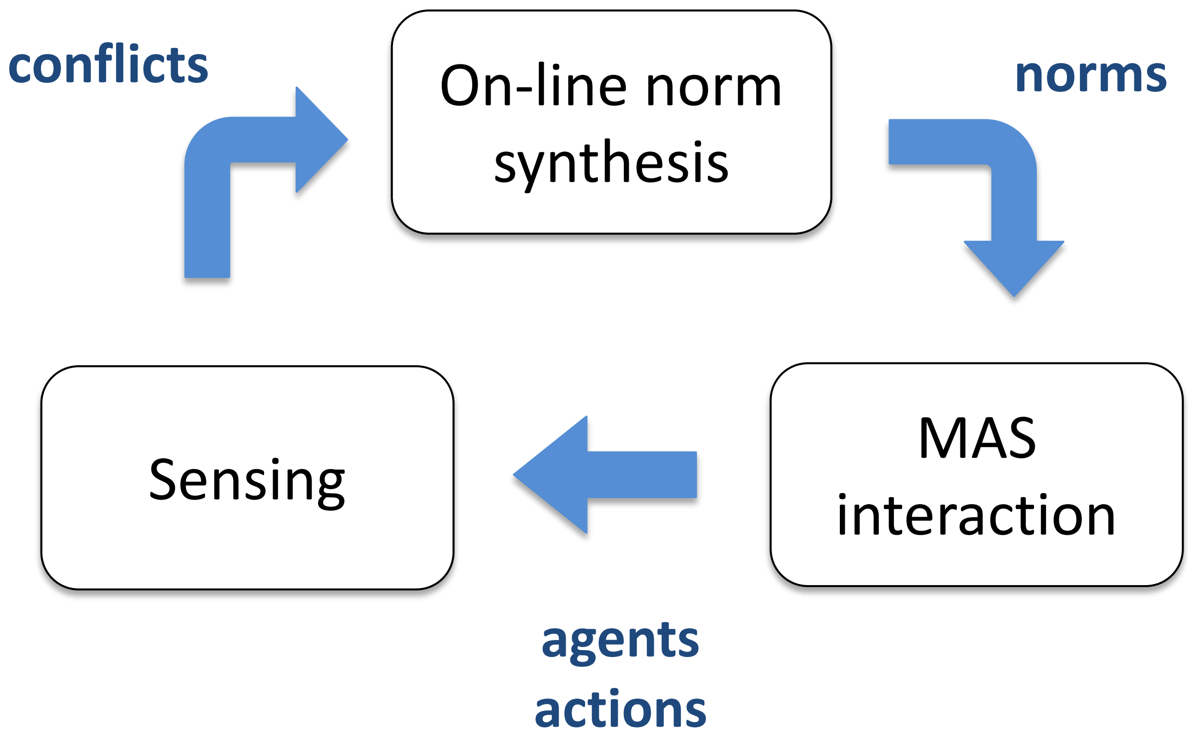
MAS
interaction

2. Research problem and approach

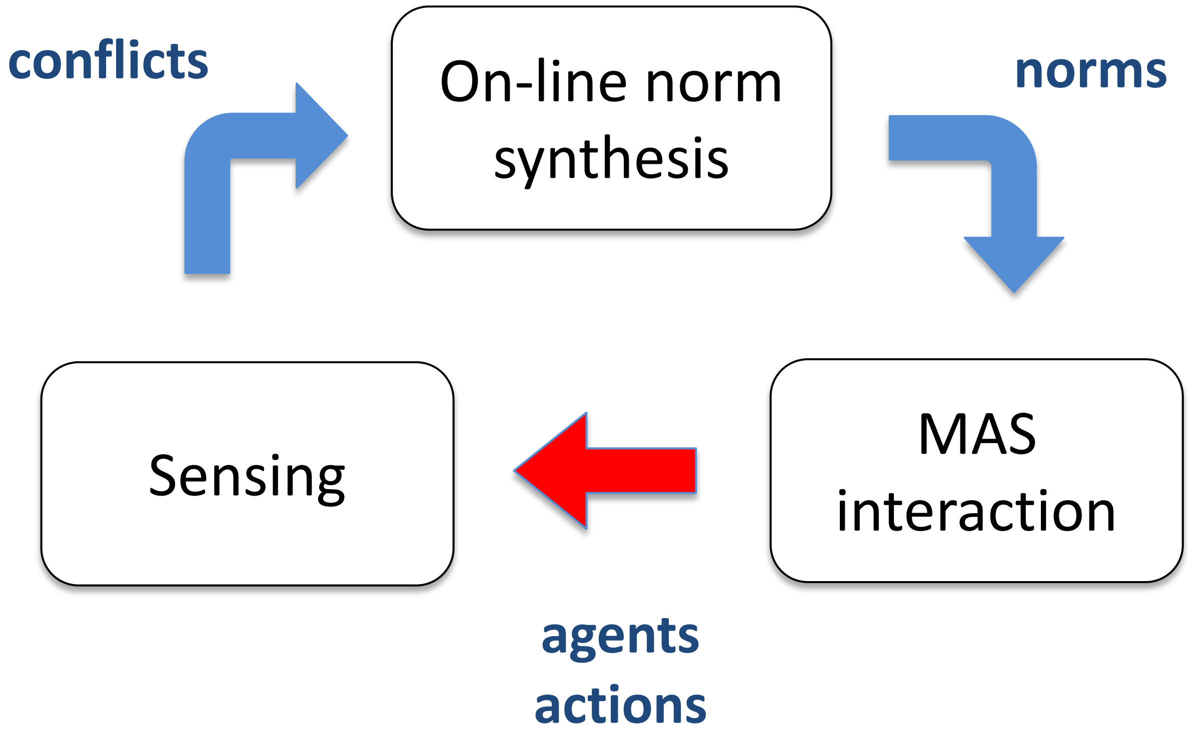
2. Research problem and approach



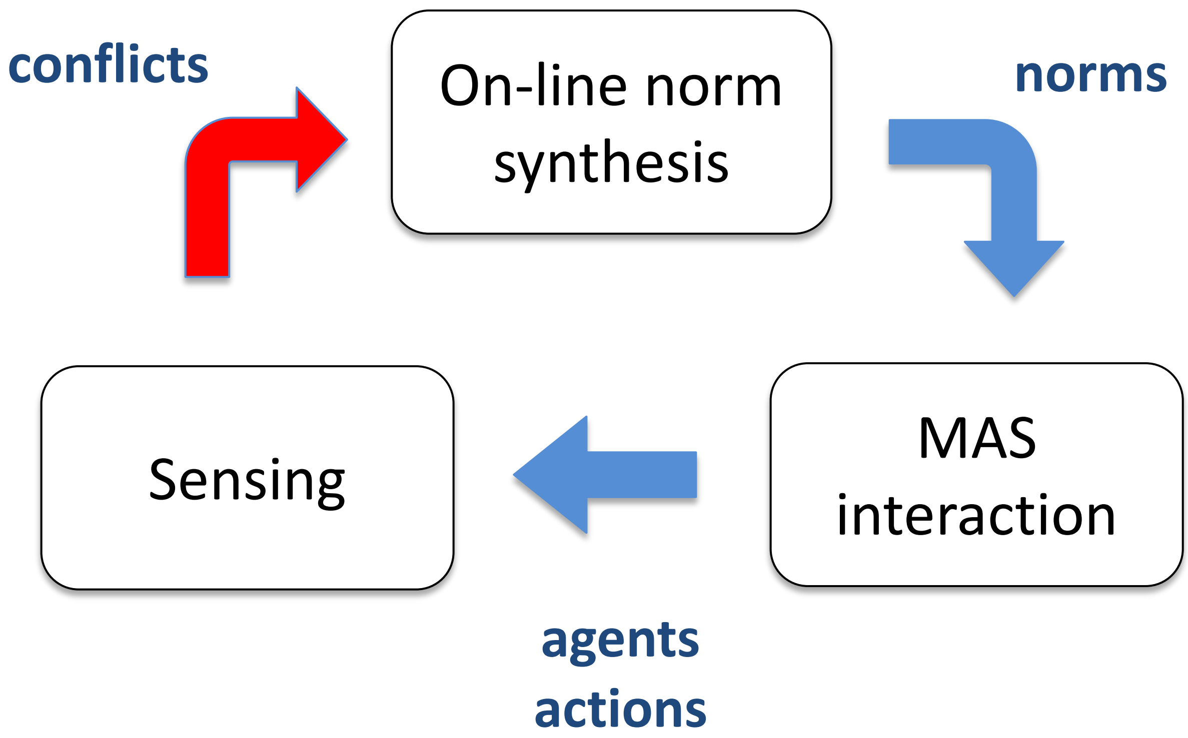
2. Research problem and approach



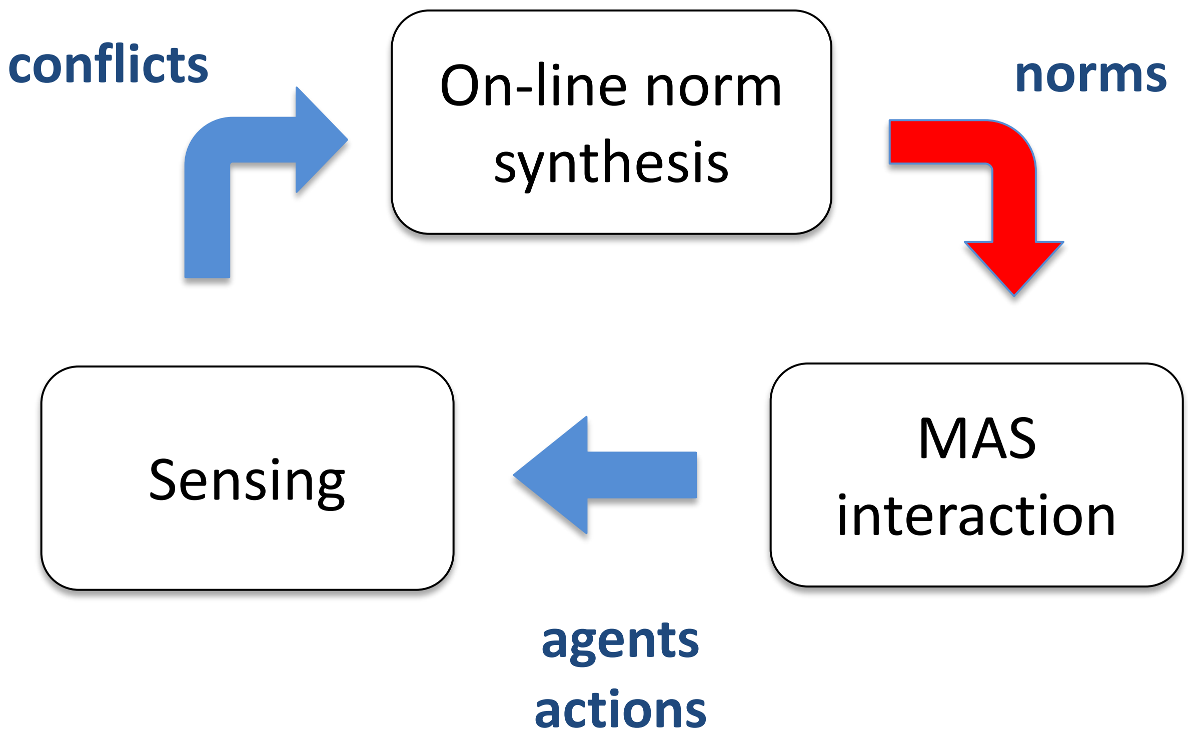
2. Research problem and approach



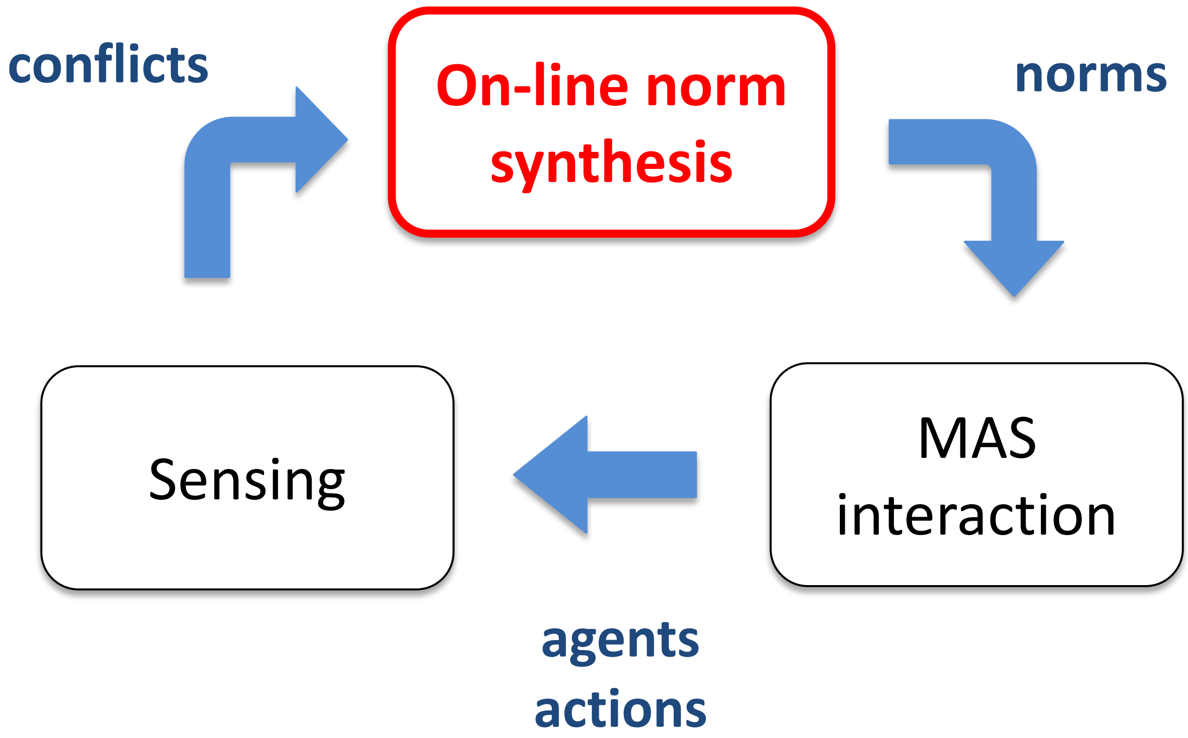
2. Research problem and approach



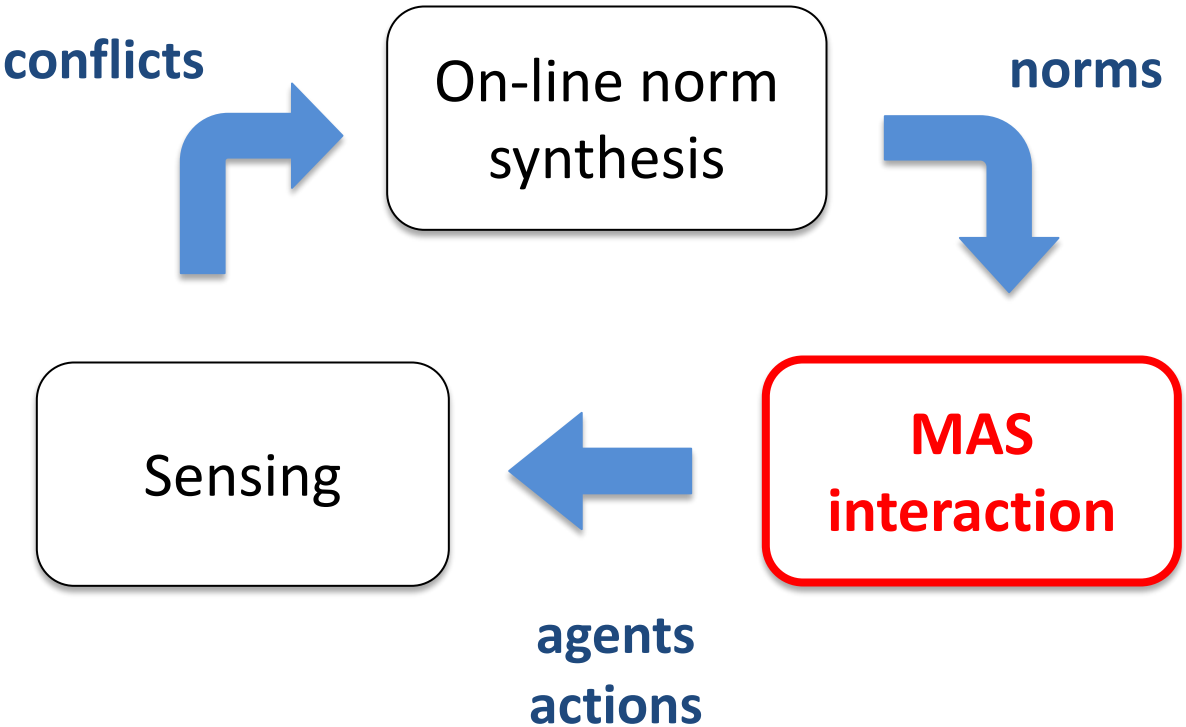
2. Research problem and approach



2. Research problem and approach



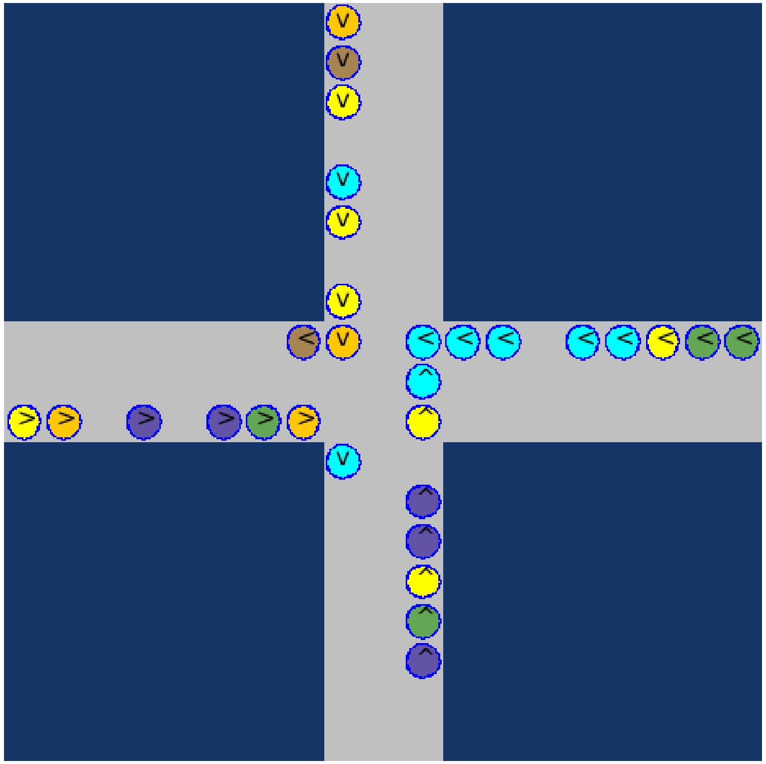
2. Research problem and approach



3. Scenario: The traffic intersection

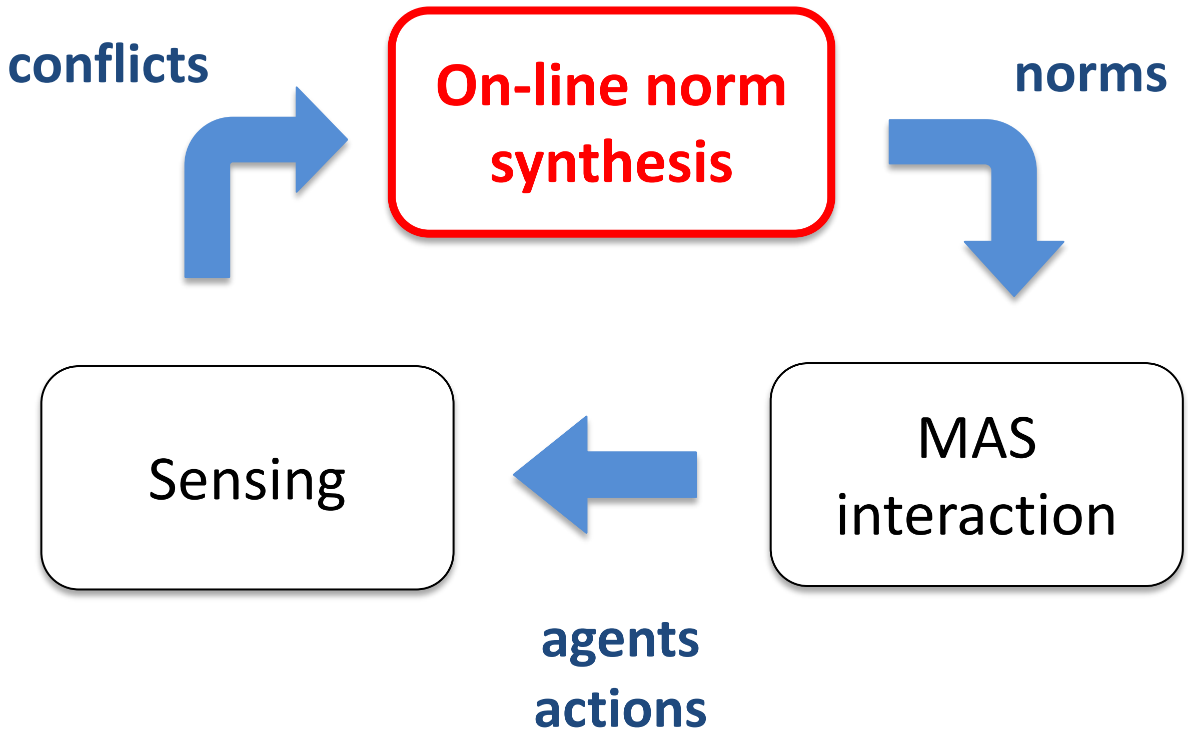
MAS interaction example = **Simulated** discretized traffic intersection:

- Agents are cars.
- Conflicts are collisions among cars.
- MAS goal is to avoid collisions among cars.



Simulated traffic intersection scenario

4. Automated Synthesis of Normative Systems



4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

2. For each detected **conflict** → **Synthesis** of new norms.
   - New norms are aimed to avoid the conflict in the future.

4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

2. For each detected **conflict** → **Synthesis** of new norms.
    - New norms are aimed to avoid the conflict in the future.

But… are synthesised norms good enough for avoiding conflicts?

4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

2. For each detected **conflict** → **Synthesis** of new norms.
   - New norms are aimed to avoid the conflict in the future.

3. **Evaluation** of synthesised norms.
   1. Are synthesised norms **effective**?
   2. Are they really **necessary**?

4. Automated Synthesis of Normative Systems

We evaluate the **performance** of norms in base of their **effectiveness** and **necessity**:

- **Effectiveness:** Do norms **avoid conflicts** when agents comply with them?
- **Necessity:** Do **conflicts arise** when agents do not comply with norms?

Effectiveness:

Consider the following norms…

1. **Give way** to your left.

2. **Never give way**.

4. Automated Synthesis of Normative Systems

Effectiveness:

Consider the following norms…

1. **Give way** to your left.

   IF Agents **apply** it, **NO collisions** arise → **EFFECTIVE** norm

2. **Never give way**.

Effectiveness:

Consider the following norms…

1. **Give way** to your left.

    IF Agents **apply** it, **NO collisions** arise → **EFFECTIVE** norm

2. **Never give way**.

    IF Agents **apply** it, **collisions** arise → **INEFFECTIVE** norm

4. Automated Synthesis of Normative Systems

Necessity:

Consider the following norms…

1. **Give way** to your left.

2. **Stop** if you do not perceive any car.

4. Automated Synthesis of Normative Systems

Necessity:

Consider the following norms…

1. **Give way** to your left.

   IF Agents **violate** it, **collisions** arise → **NECESSARY** norm

2. **Stop** if you do not perceive any car.

4. Automated Synthesis of Normative Systems

Necessity:

Consider the following norms…

1. **Give way** to your left.

   IF Agents **violate** it, **collisions** arise → **NECESSARY** norm

2. **Stop** if you do not perceive any car.

   IF Agents **violate** it, **no collisions** arise → **UNNECESSARY** norm

4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

2. For each detected **conflict** → **Synthesis** of new norms.
   - New norms are aimed to avoid the conflict in the future.

3. **Evaluation** of synthesised norms.
   1. Are synthesised norms **effective**?
   2. Are they really **necessary**?

4. Automated Synthesis of Normative Systems

1. **Conflict detection** by MAS observation.

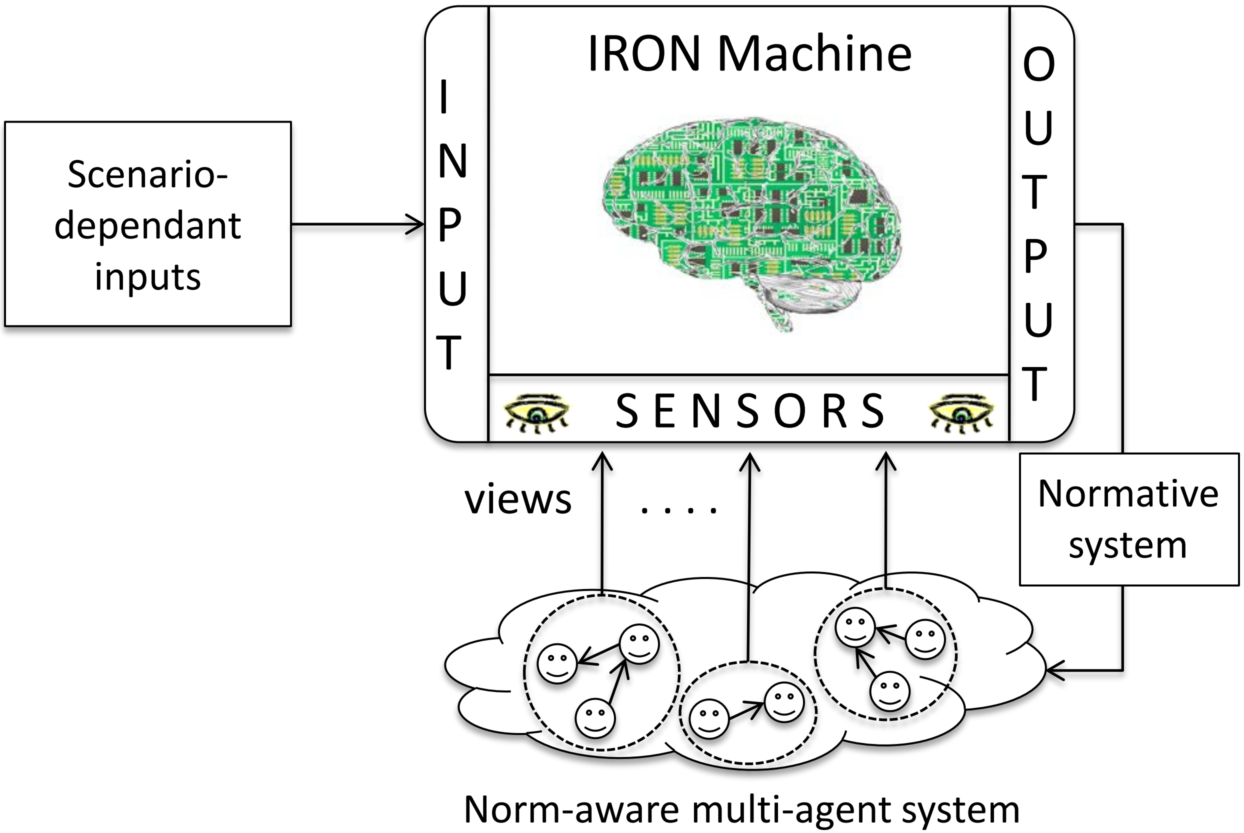
2. For each detected **conflict** → **Synthesis** of new norms.
   - New norms are aimed to avoid the conflict in the future.

3. **Evaluation** of synthesised norms.
   1. Are synthesised norms **effective**?
   2. Are they really **necessary**?

4. **Refinement** of norms. Norms that do not perform well (ineffective and unnecessary norms) are removed.

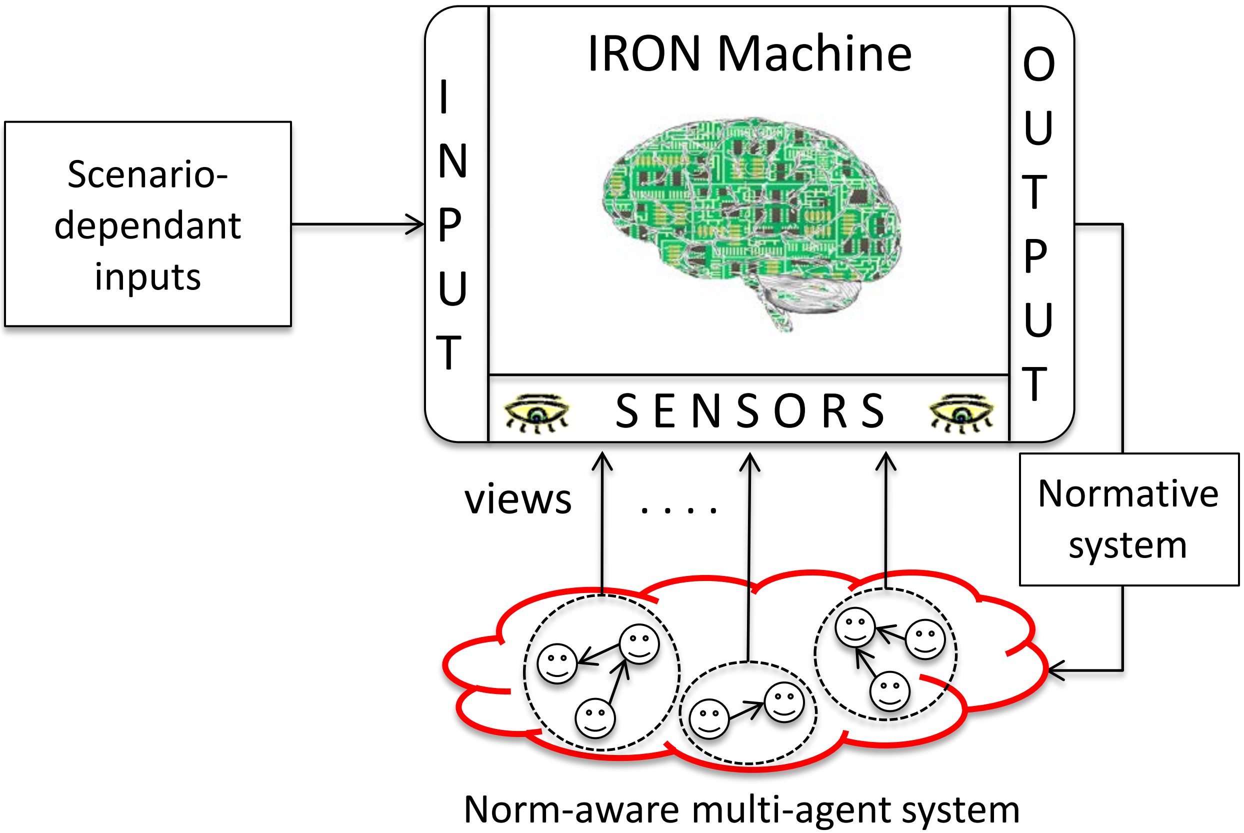
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



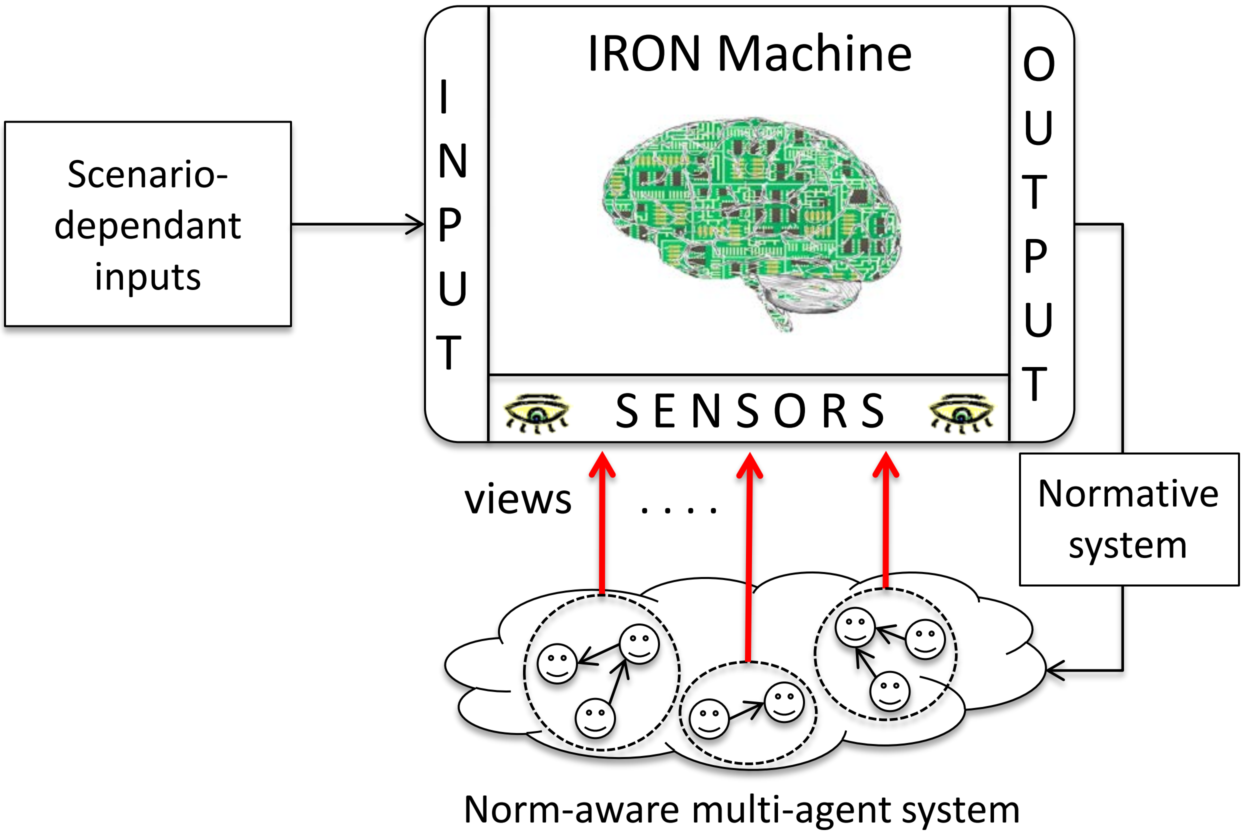
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



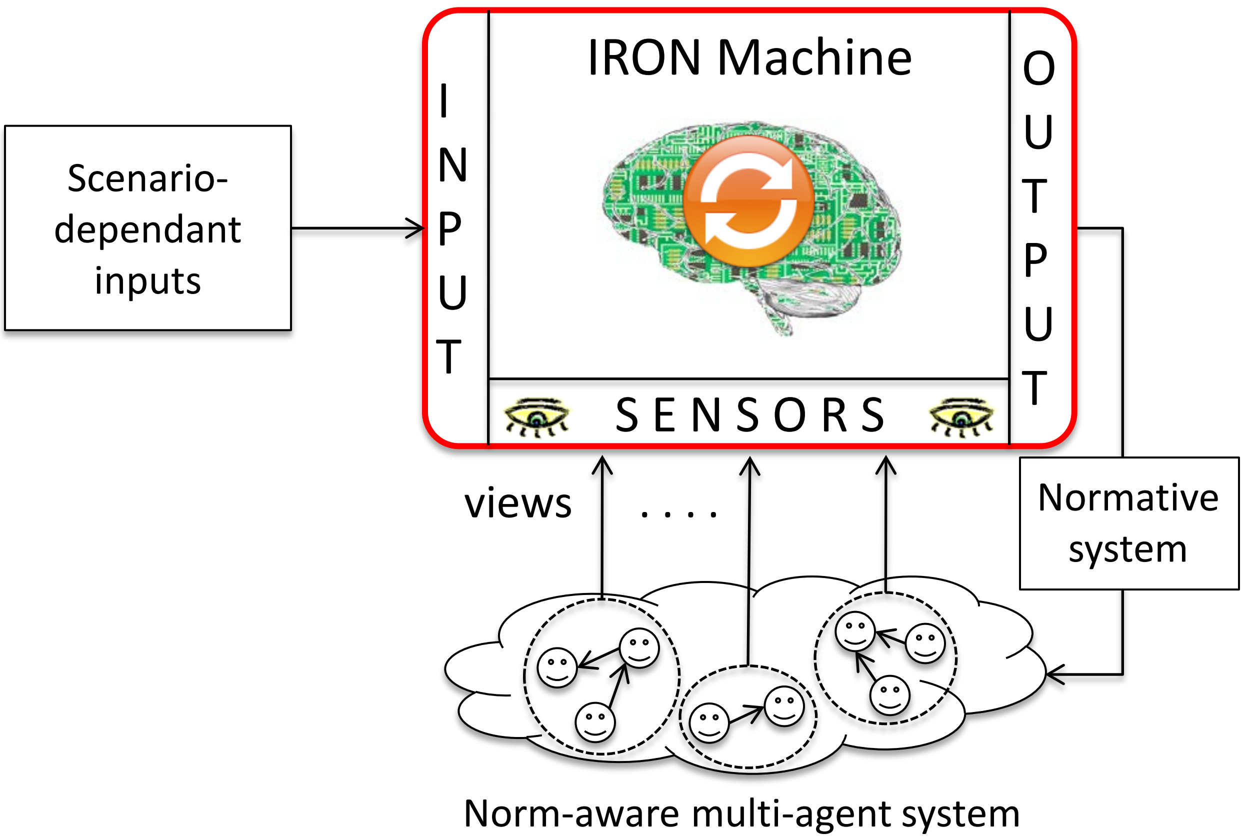
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



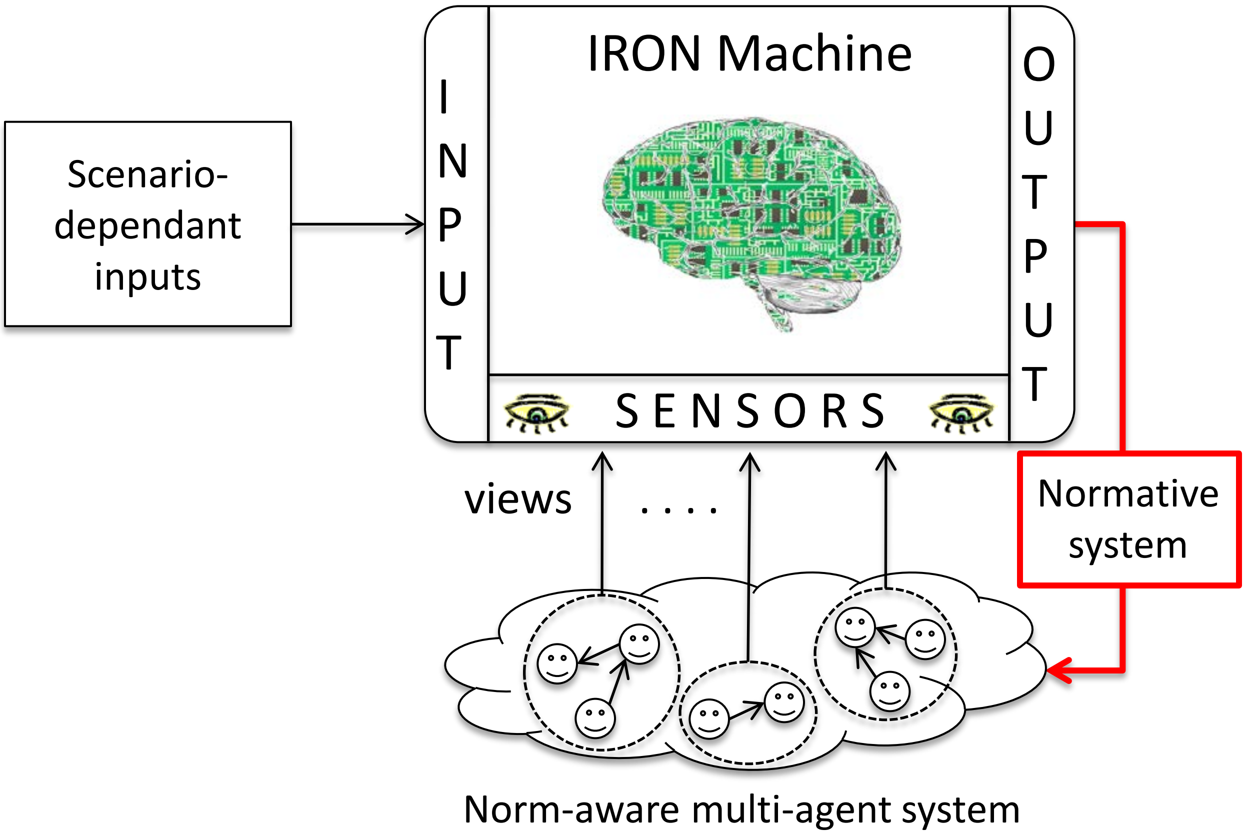
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



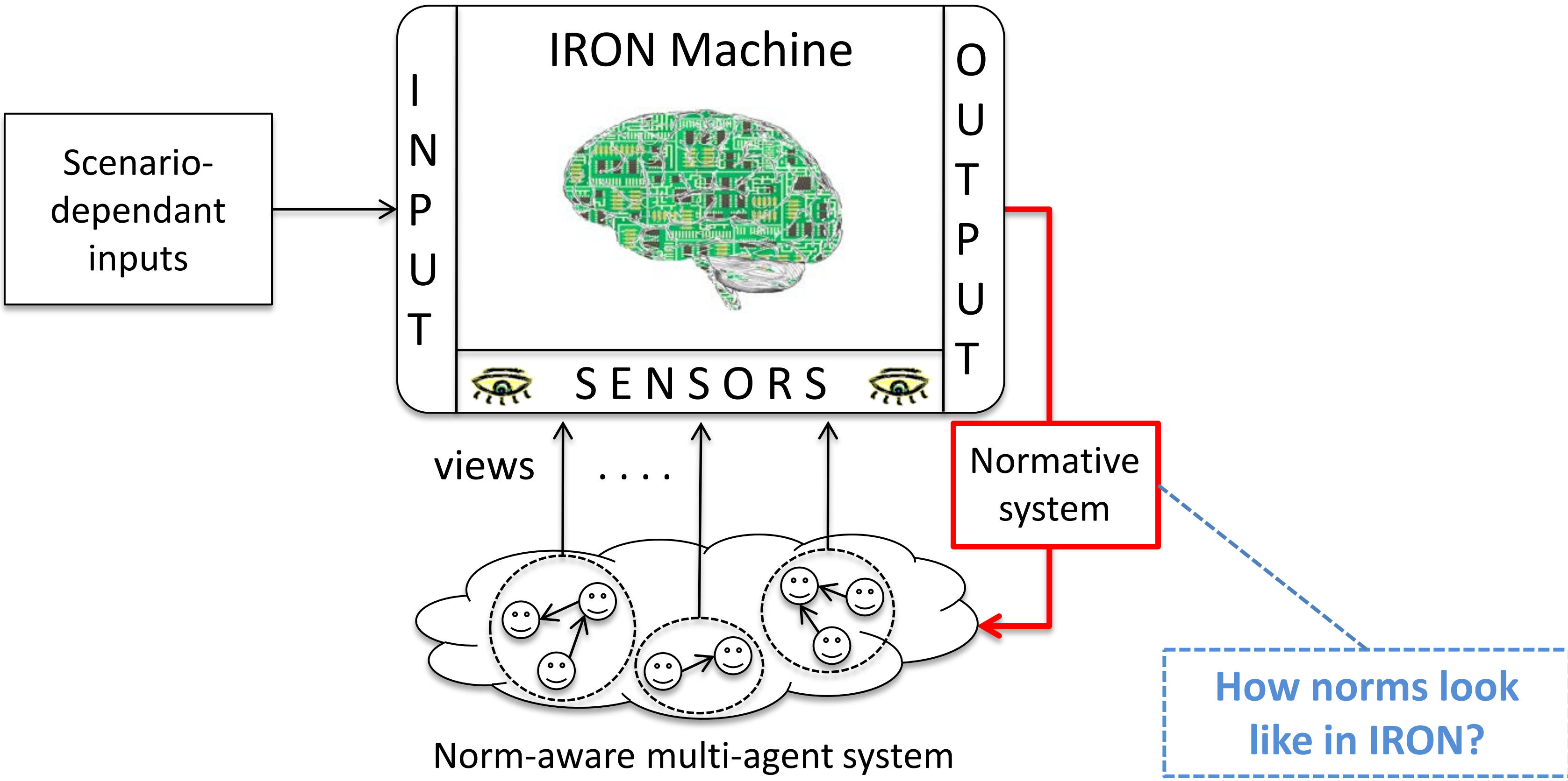
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



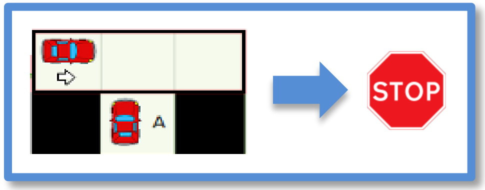
4. IRON: Automated Synthesis of Normative Systems

IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



4. IRON: Automated Synthesis of Normative Systems

A norm is an **IF … THEN …** rule like:



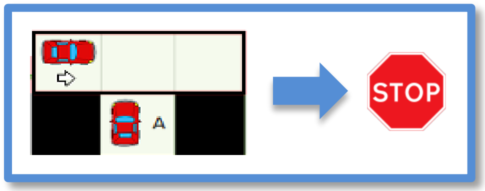
<left(car-to-right)&front(-)&right(-), obl(stop)>

- IRON synthesises norms from the agents'perspective → **Agents can understand norms.**

- Whenever the local perception of a agent satisfies the precondition (IF) of a norm, then the norm **applies to the agent.**

4. IRON: Automated Synthesis of Normative Systems

A norm is an **IF … THEN …** rule like:



<left(car-to-right)&front(-)&right(-), obl(stop)>

Formally, a norm is of the form **$<\varphi, \theta(ac)>$**

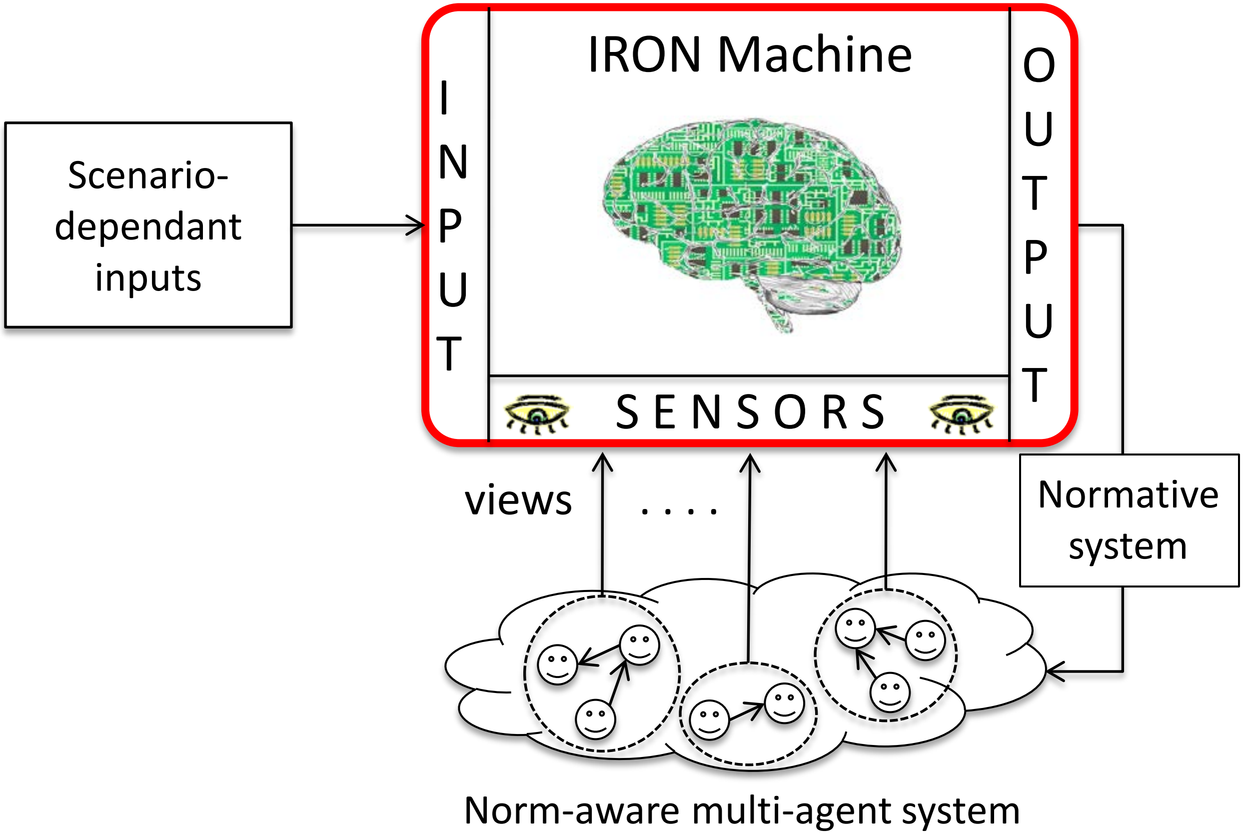
- **φ** is the precondition.
- **ac** is an action available to the agents.
- **θ(ac)** is a deontic operator.

To synthesise norms, IRON uses a BNF grammar:

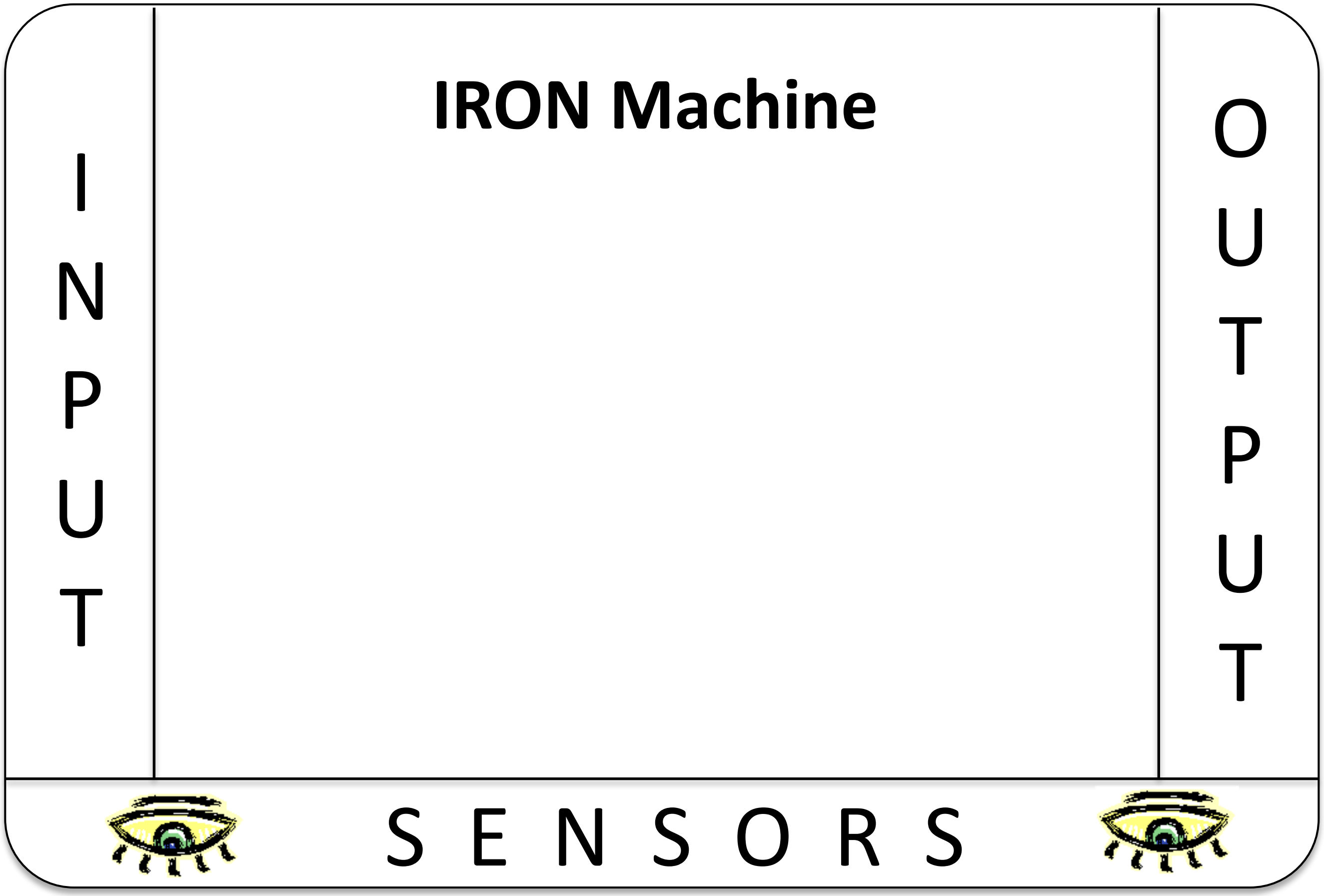
Norm	$::= <\varphi, \theta(ac)>$
φ	$::= <\varphi \,\&\, \varphi> \mid \alpha$
θ	$::=$ obl | perm | prh
Ac	$::= ac_1 \mid ac_2 \mid \ldots \mid ac_n$
α	$::= p^n(\tau_1, \ldots, \tau_n)$

4. IRON: Automated Synthesis of Normative Systems

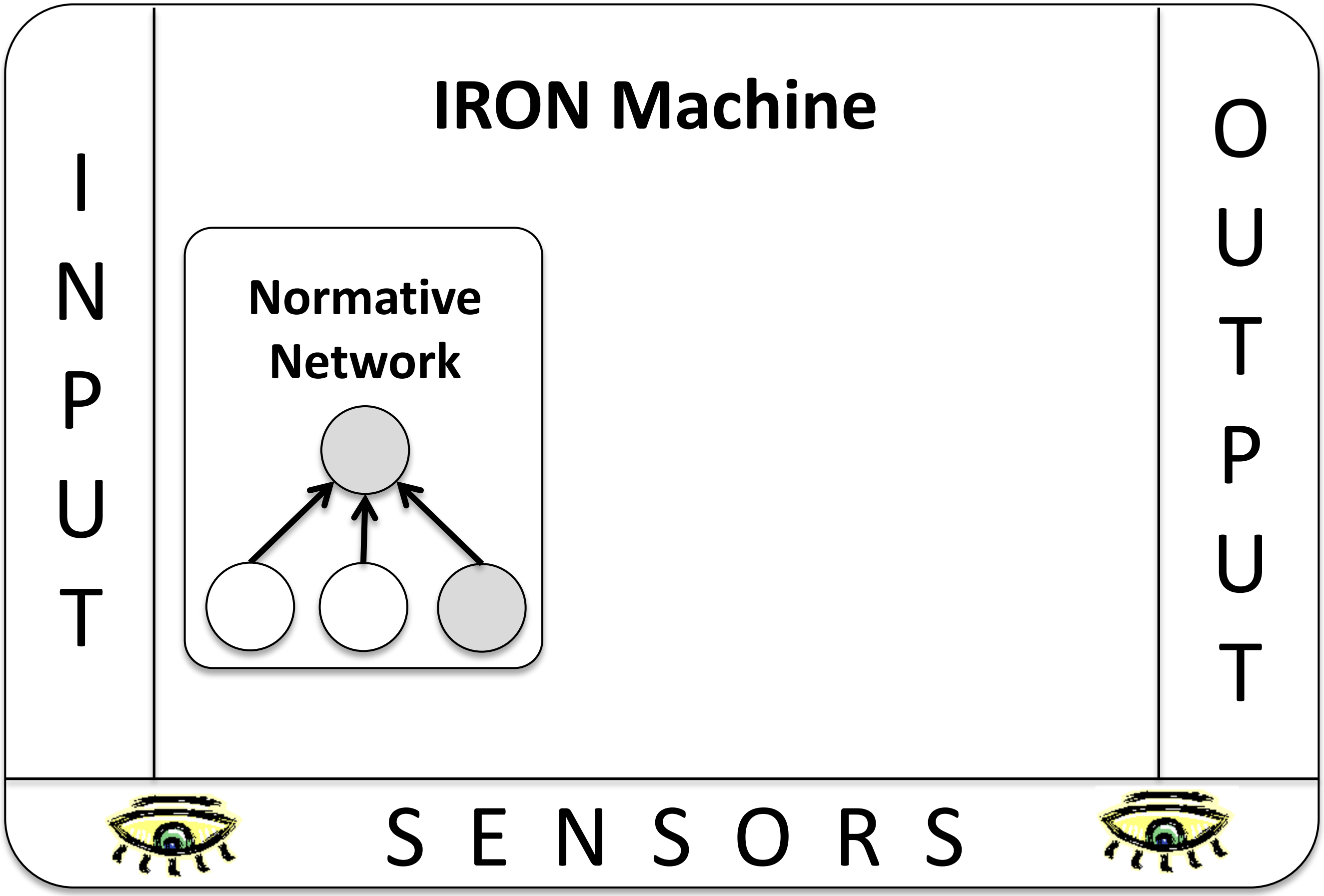
IRON *(Intelligent Robust On-line Norm synthesis mechanism)* solves the on-line automated norm synthesis problem.



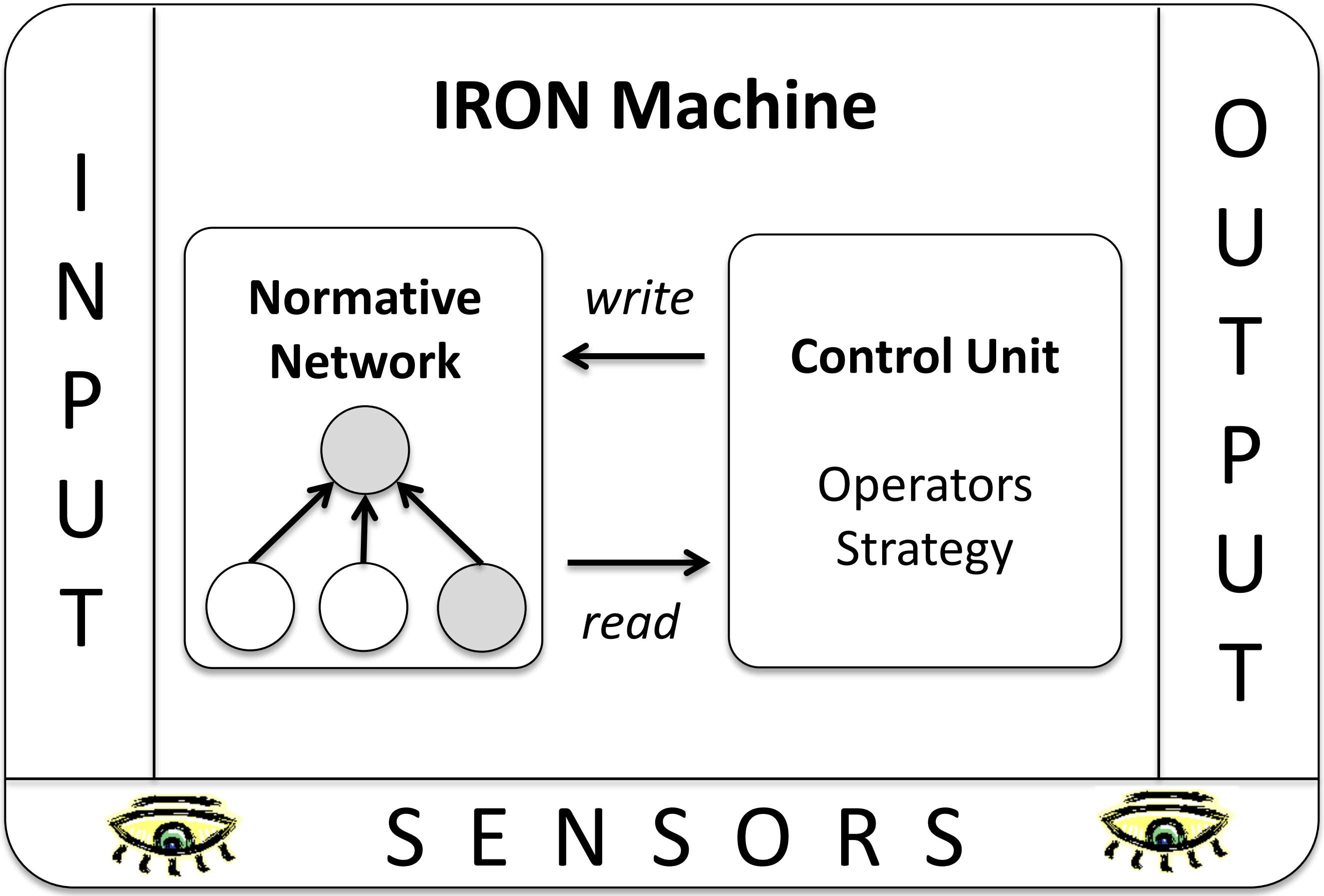
4. IRON: Automated Synthesis of Normative Systems



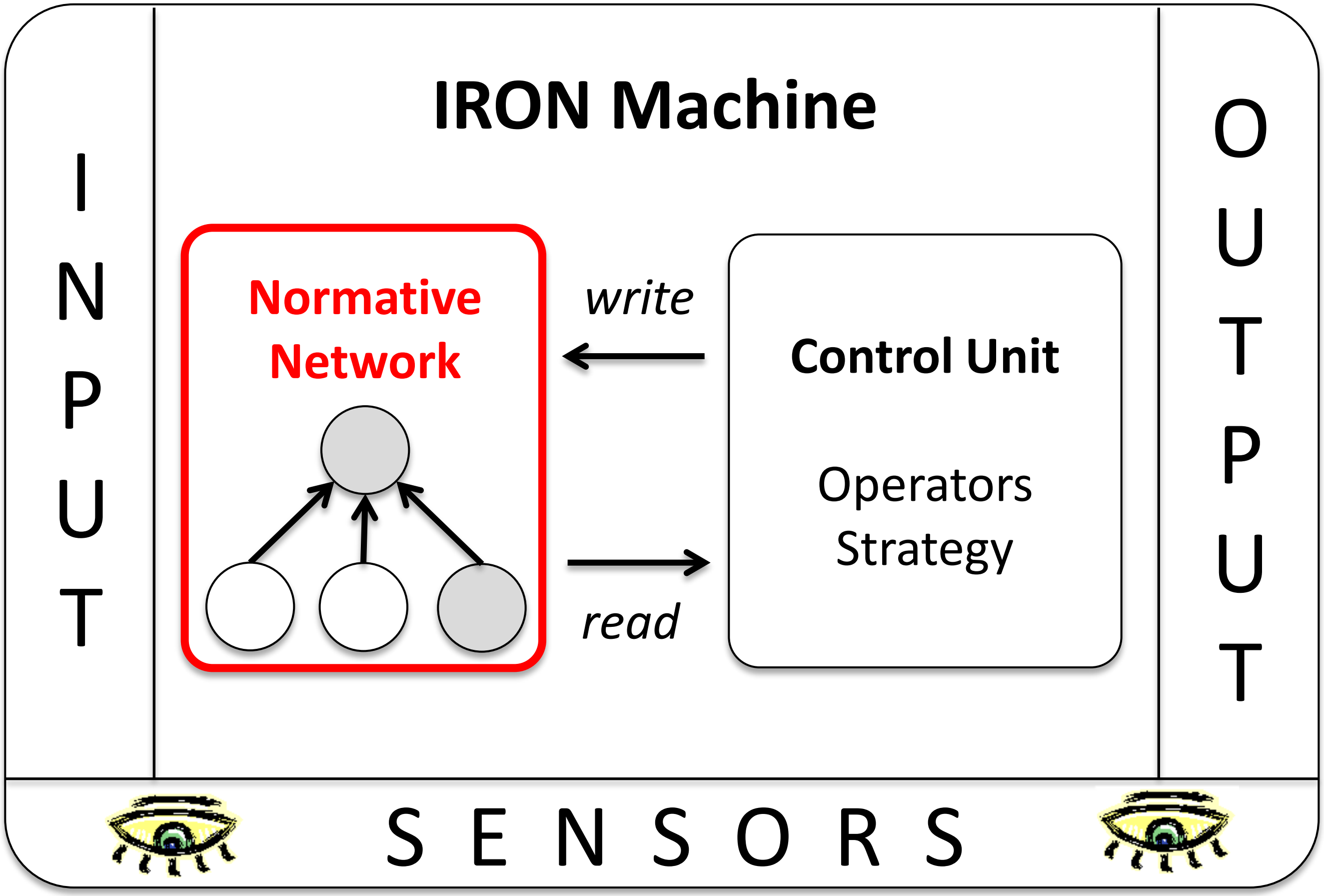
4. IRON: Automated Synthesis of Normative Systems



4. IRON: Automated Synthesis of Normative Systems

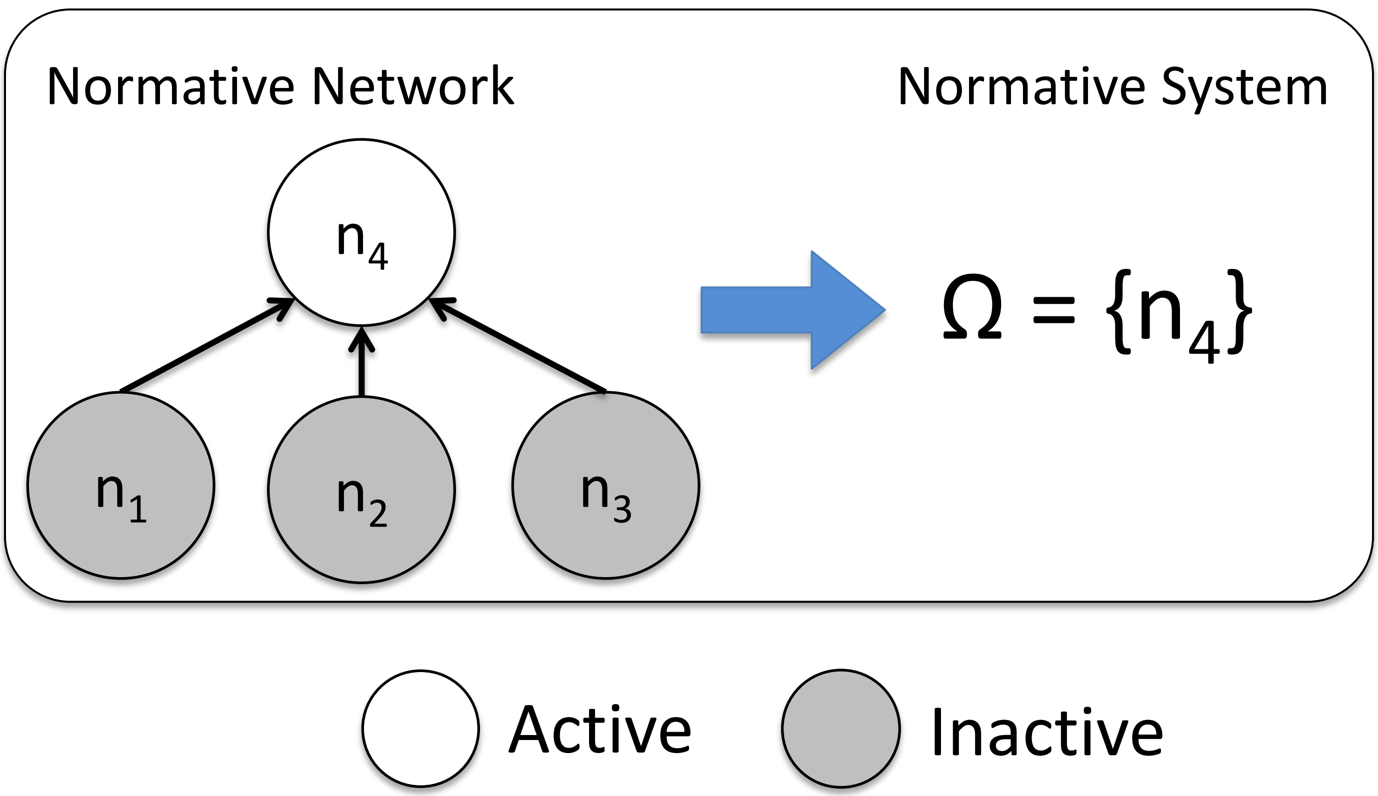


4. IRON: Automated Synthesis of Normative Systems

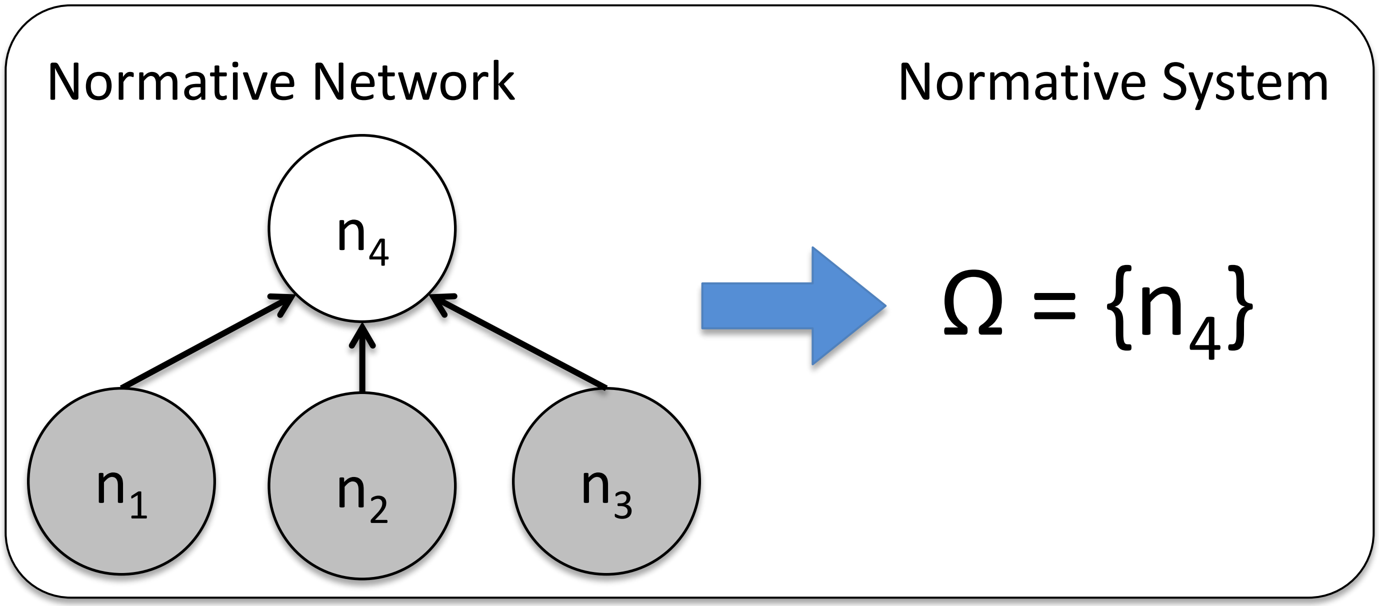


4. IRON: The Normative Network

- **Data structure** to represent explored norms.
- **Nodes** stand for norms.
- **Edges** stand for **generalisation relationships** between norms.
- A normative network represents a normative system **Ω** as its active norms.



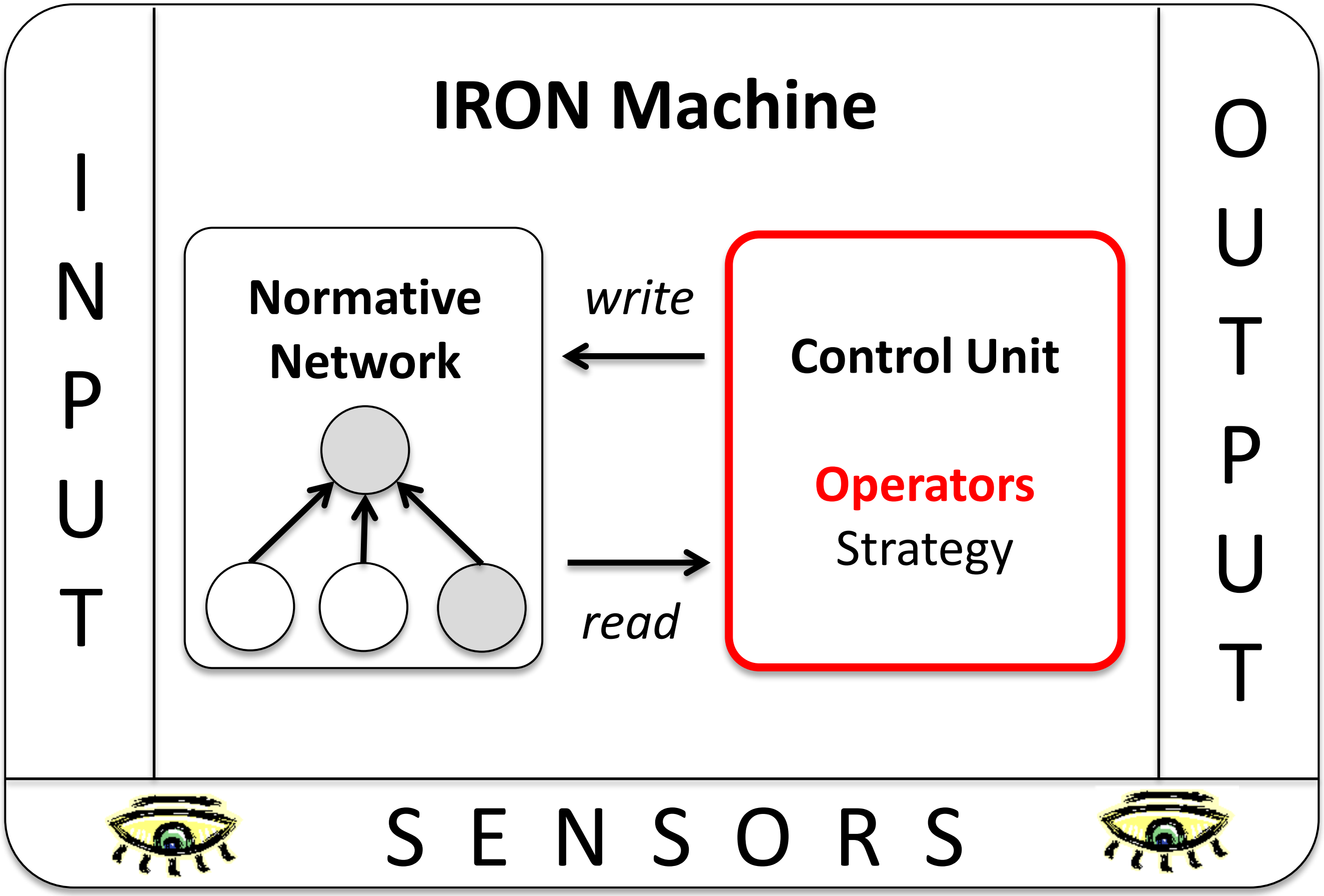
4. IRON: The Normative Network



n_4: Give way to emergency vehicles

n_1: Give way to police cars **n_2: Give way to fire-trucks** **n_3: Give way to ambulances**

4. IRON: Automated Synthesis of Normative Systems



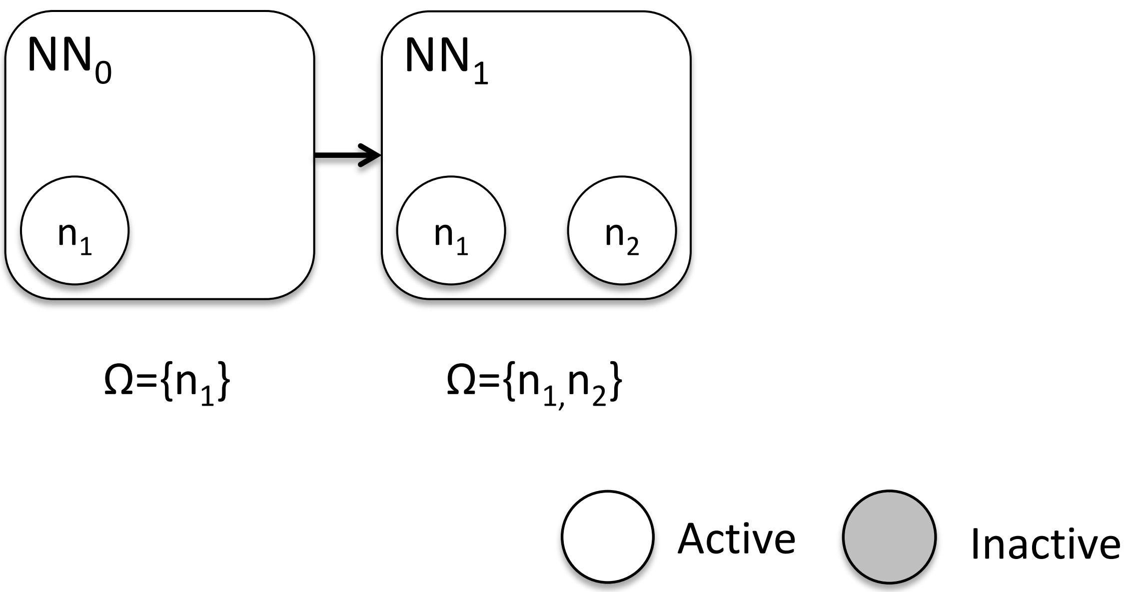
4. IRON: Operators

Operators apply changes to the Normative Network → Transitions from one **normative system** to another.

4. IRON: Operators

Operators apply changes to the Normative Network → Transitions from one **normative system** to another.

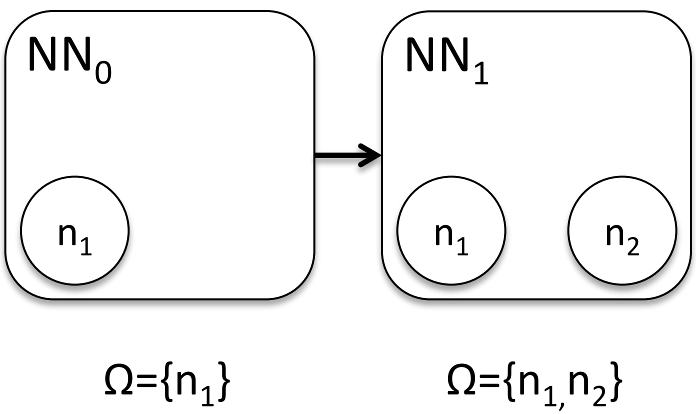
create: Synthesises a norm and adds it to the normative network



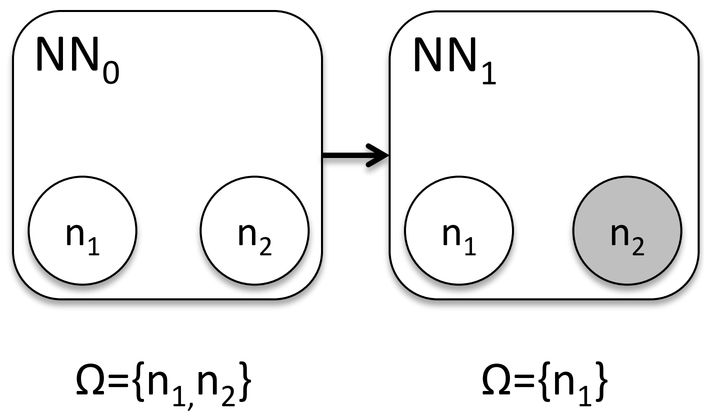
4. IRON: Operators

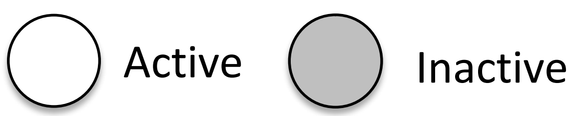
Operators apply changes to the Normative Network → Transitions from one **normative system** to another.

create: Synthesises a norm and adds it to the normative network



deactivate: Deactivates a norm in the normative network

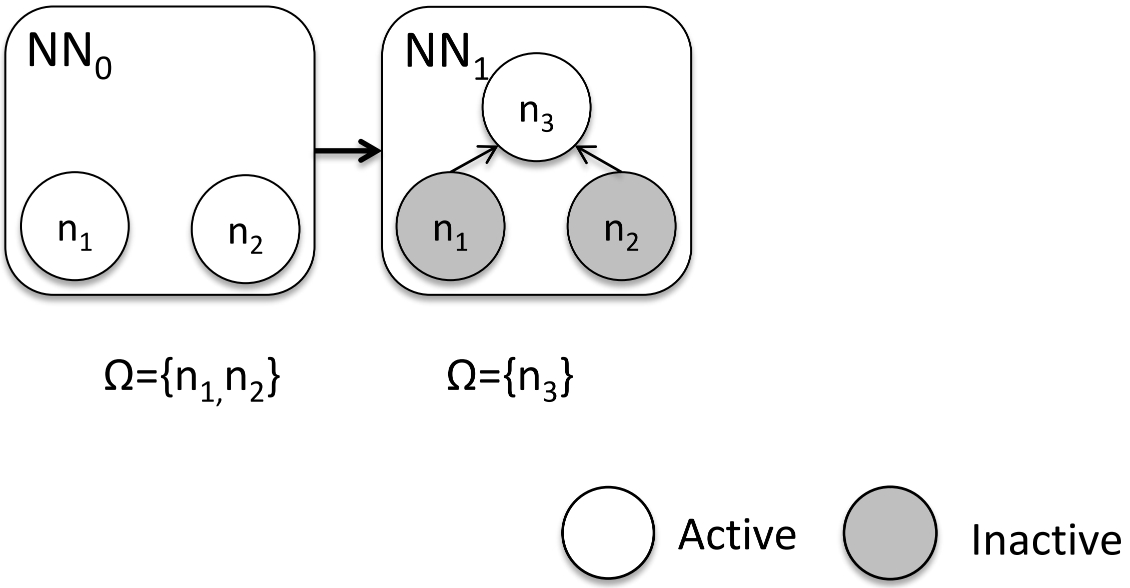




4. IRON: Operators

Operators apply changes to the Normative Network → Transitions from one **normative system** to another.

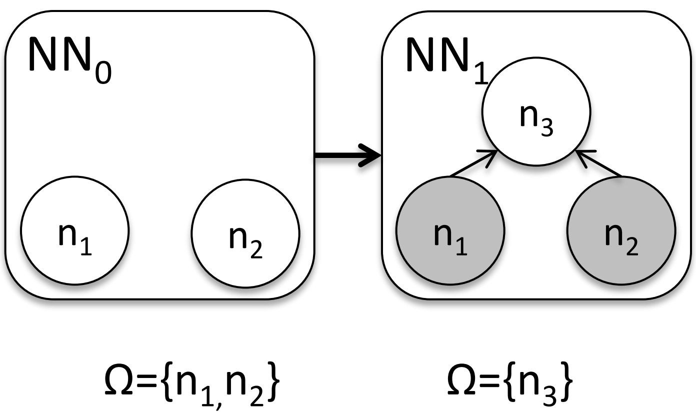
generalise: Generalises a set of norms into a parent norm



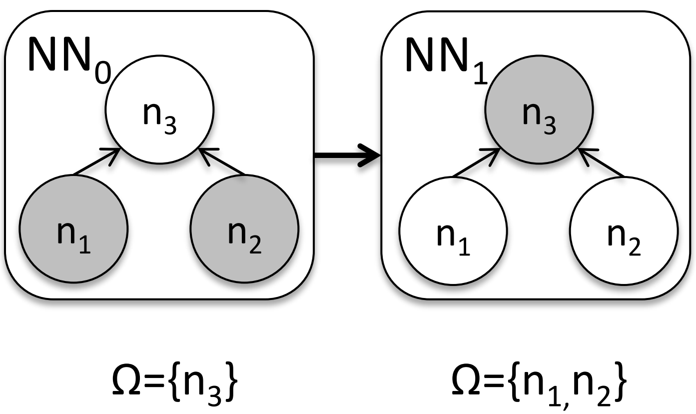
4. IRON: Operators

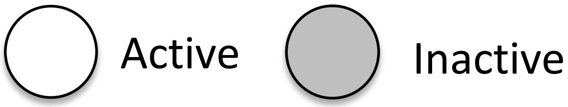
Operators apply changes to the Normative Network → Transitions from one **normative system** to another.

generalise: Generalises a set of norms into a parent norm

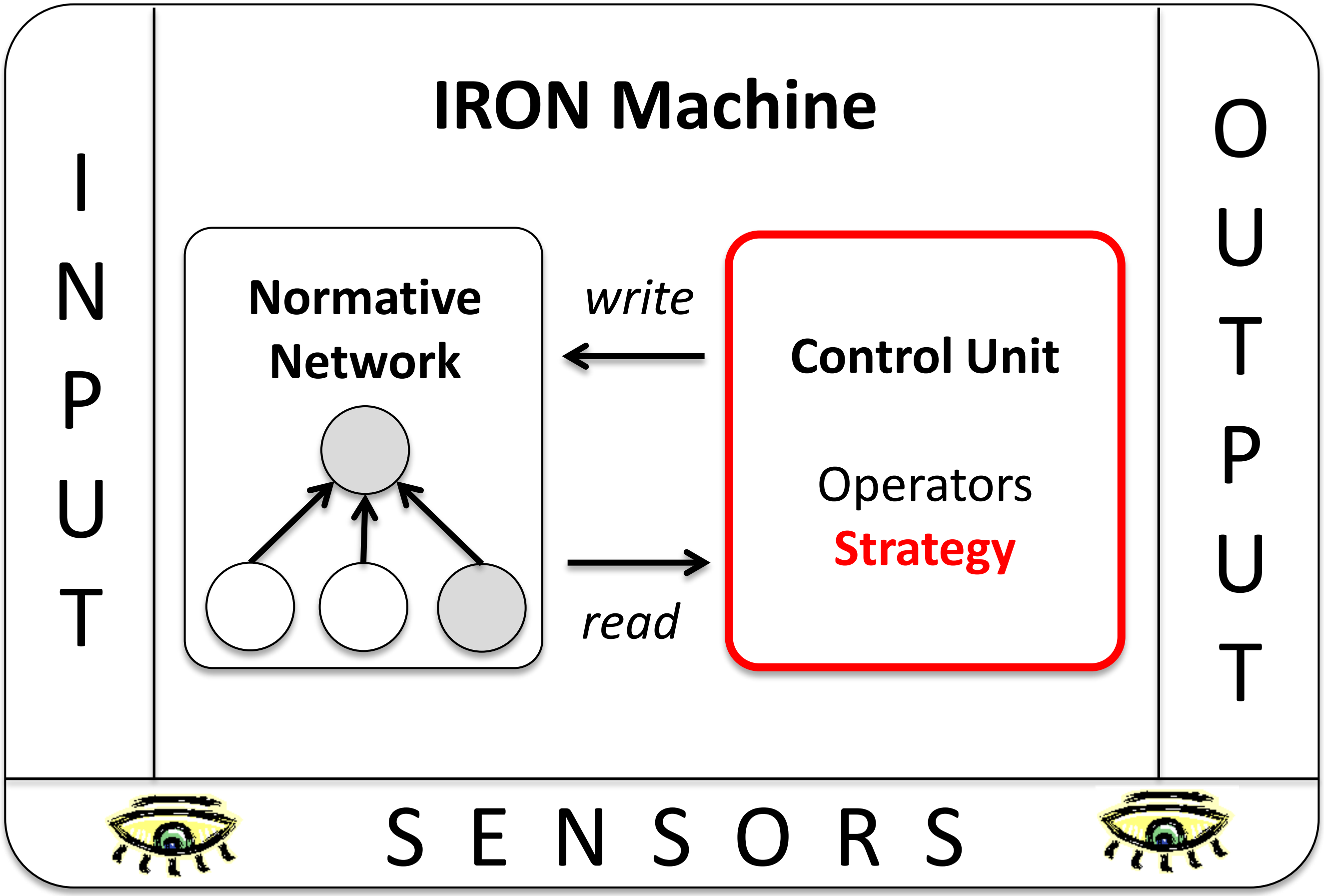


specialises: Undoes a norm generalisation





4. IRON: Automated Synthesis of Normative Systems



4. IRON: Strategy

1. **Conflict detection** from perceptions of the MAS.

2. **Norm Synthesis:** For each detected conflict, it uses operator **create** to synthesise norms.

3. **Norm evaluation:** Evaluates norms effectiveness and necessity based on their outcomes in the MAS.

4. **Norm refinement:** Deactivates ineffective and unnecessary norms by means of operator **deactivate**. Generalises and specialises norms using operators **generalise** and **specialise**.

4. IRON: Strategy

1. **Conflict detection** from perceptions of the MAS.

2. **Norm Synthesis:** For each detected conflict, it uses operator **create** to synthesise norms.

3. **Norm evaluation:** Evaluates norms effectiveness and necessity based on their outcomes in the MAS.

4. **Norm refinement:** Deactivates ineffective and unnecessary norms by means of operator **deactivate**. Generalises and specialises norms using operators **generalise** and **specialise**.

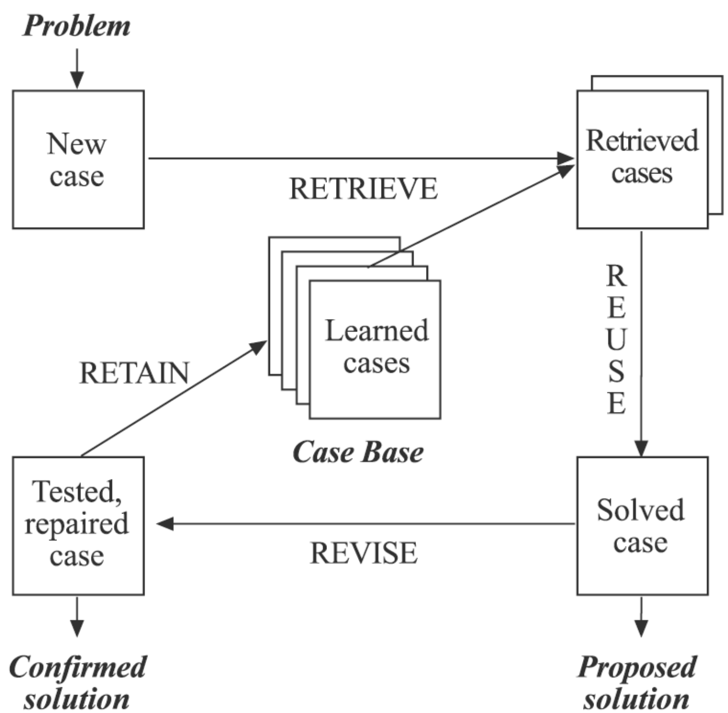
Creation of new norms → Based on **experience**

We require an AI technique to store experiences and their solutions, and learn from them.

4. IRON: Strategy. Norm synthesis

Case Based Reasoning (CBR). Solving problems based on the following principle:

- Similar problems have similar solutions.



Case base: Experience is stored in the formof cases, each case with its solution.

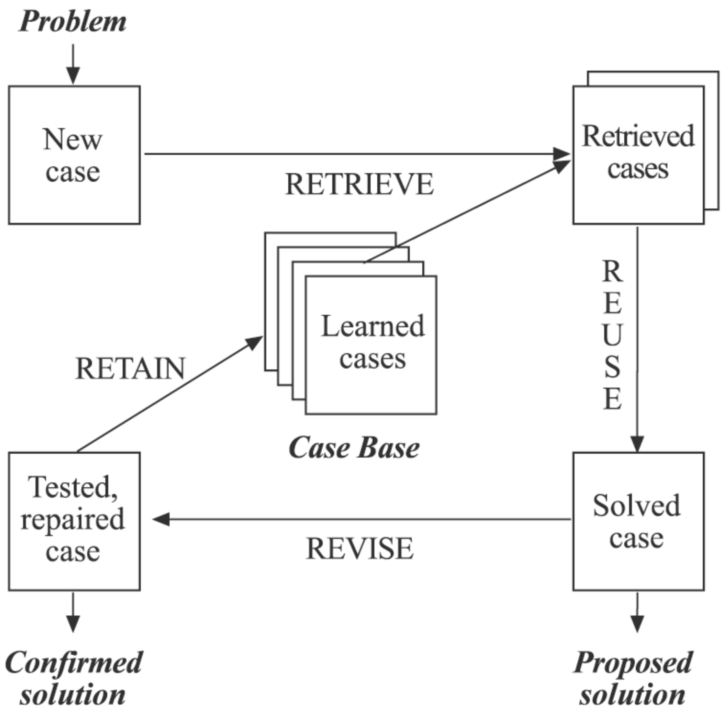
Whenever we want to solve a new problem (case):
Quan volem resoldre un nou problema (cas):

1. Build new description of the case
2. Search into the case base for the most similar problem.
3. Adapt its solution to the new case.
4. Revise the solution: Does it work?
5. If the new case is relevant and its solution works, store new case.

4. IRON: Strategy. Norm synthesis

Case Based Reasoning (CBR). Solving problems based on the following principle:

- Similar problems have similar solutions.



Case base: Experience is stored in the formof cases, each case with its solution.

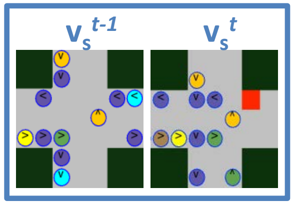
Whenever we want to solve a new problem (case):
Quan volem resoldre un nou problema (cas):

1. Build new description of the case
2. Search into the case base for the most similar problem.
3. Adapt its solution to the new case.
4. Revise the solution: Does it work?
5. If the new case is relevant and its solution works, store new case.

Problem: CBR requires a human to revise solutions and evaluate how good they are. In our case... **Unsupervised CBR.**

4. IRON: Strategy. Norm synthesis

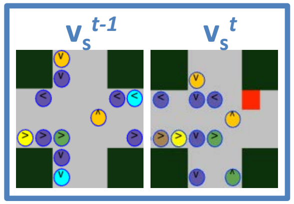
Initially empty
Case Base



New Conflict description arrives

4. IRON: Strategy. Norm synthesis

Initially empty
Case Base

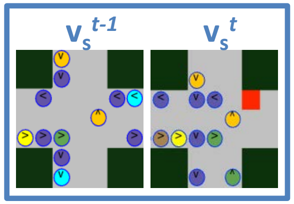


New Conflict description arrives

Search for a similar situation

4. IRON: Strategy. Norm synthesis

Initially empty
Case Base



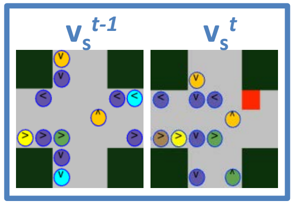
New Conflict description arrives

Search for a similar situation

No similar case is found

4. IRON: Strategy. Norm synthesis

Initially empty
Case Base



New Conflict description arrives

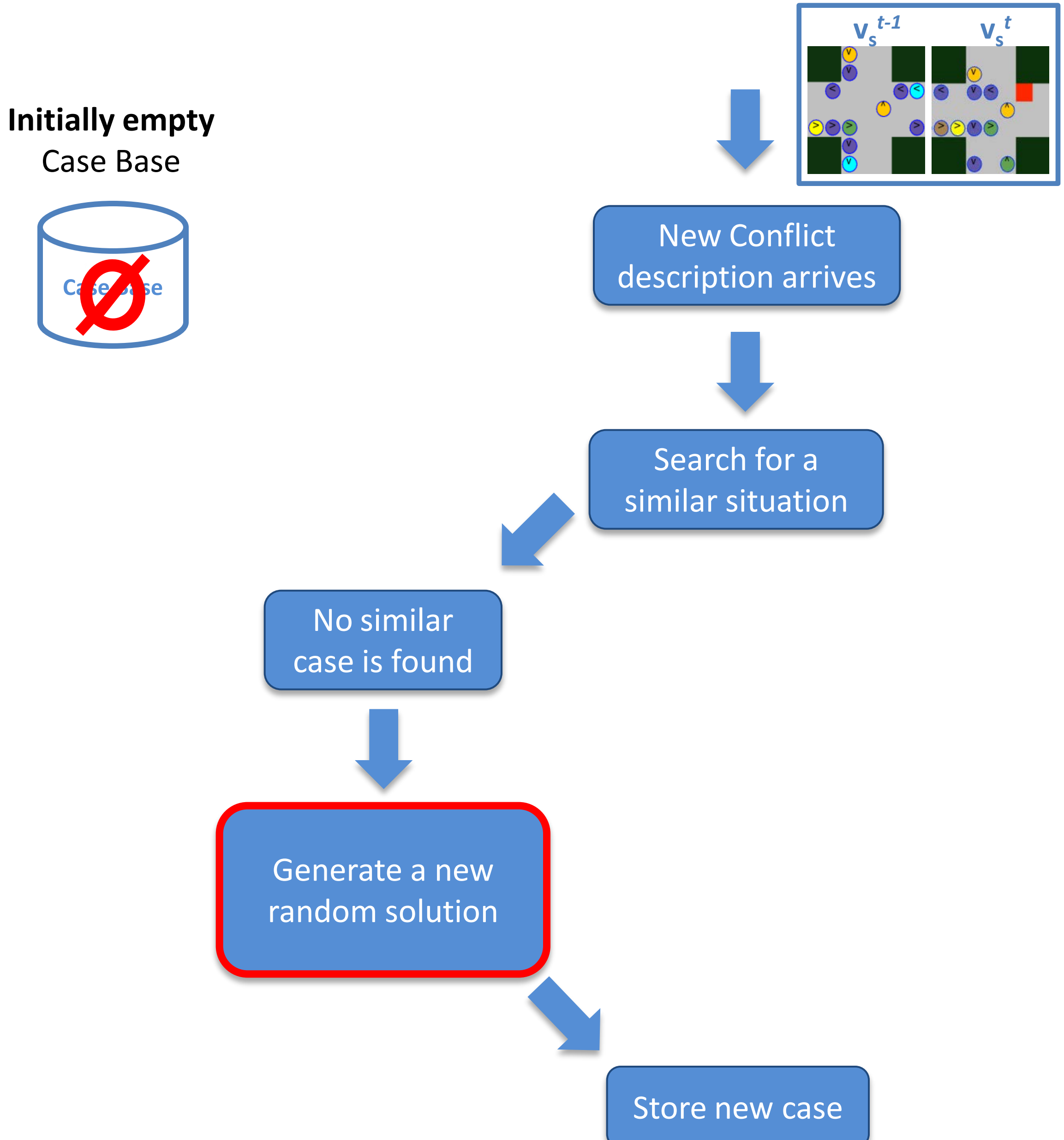
Search for a similar situation

No similar case is found

Generate a new random solution

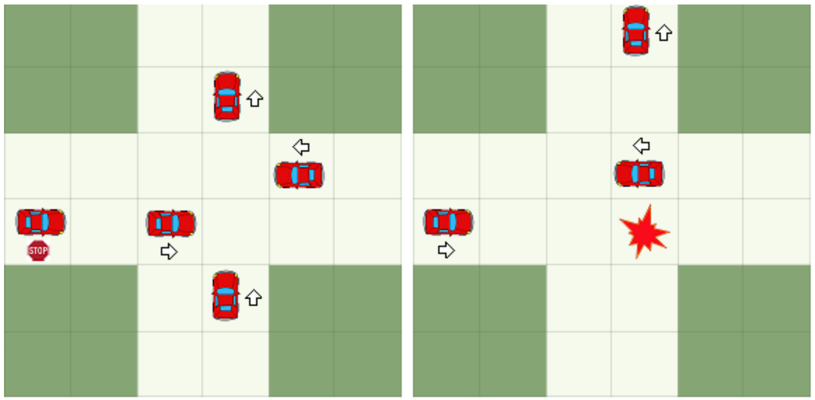
Store new case

4. IRON: Strategy. Norm synthesis



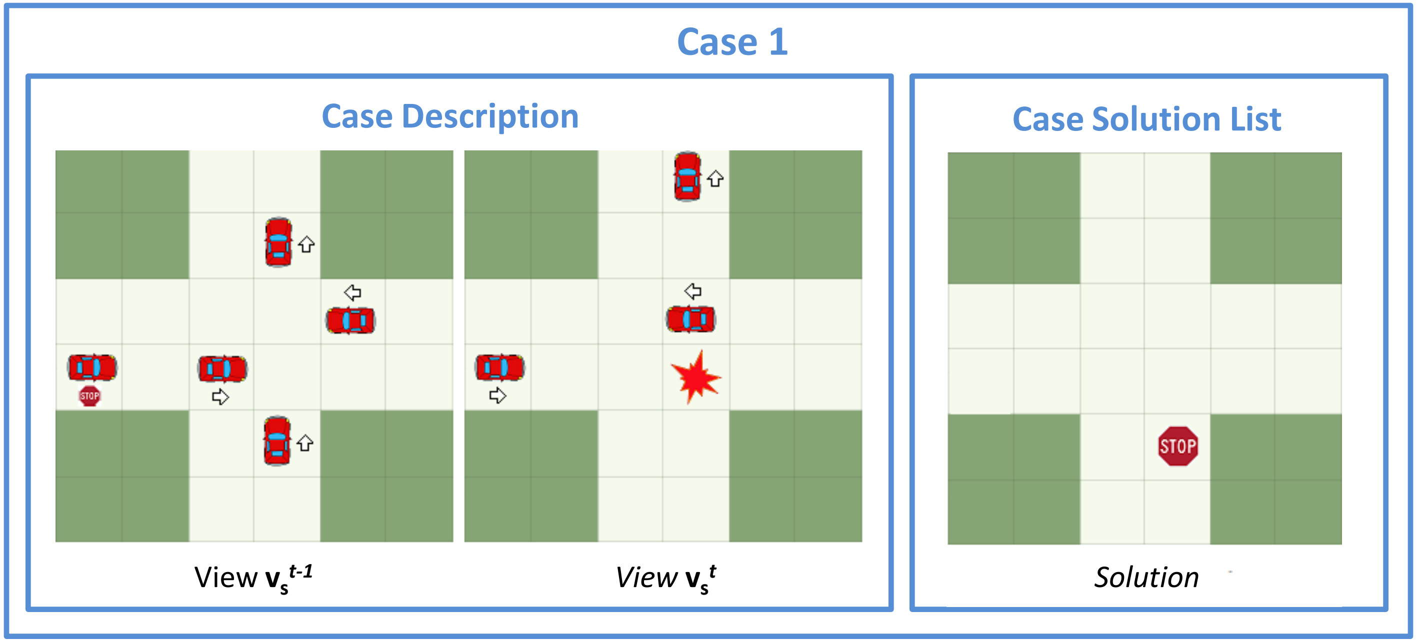
4. IRON: Strategy. Norm synthesis

Conflict Description

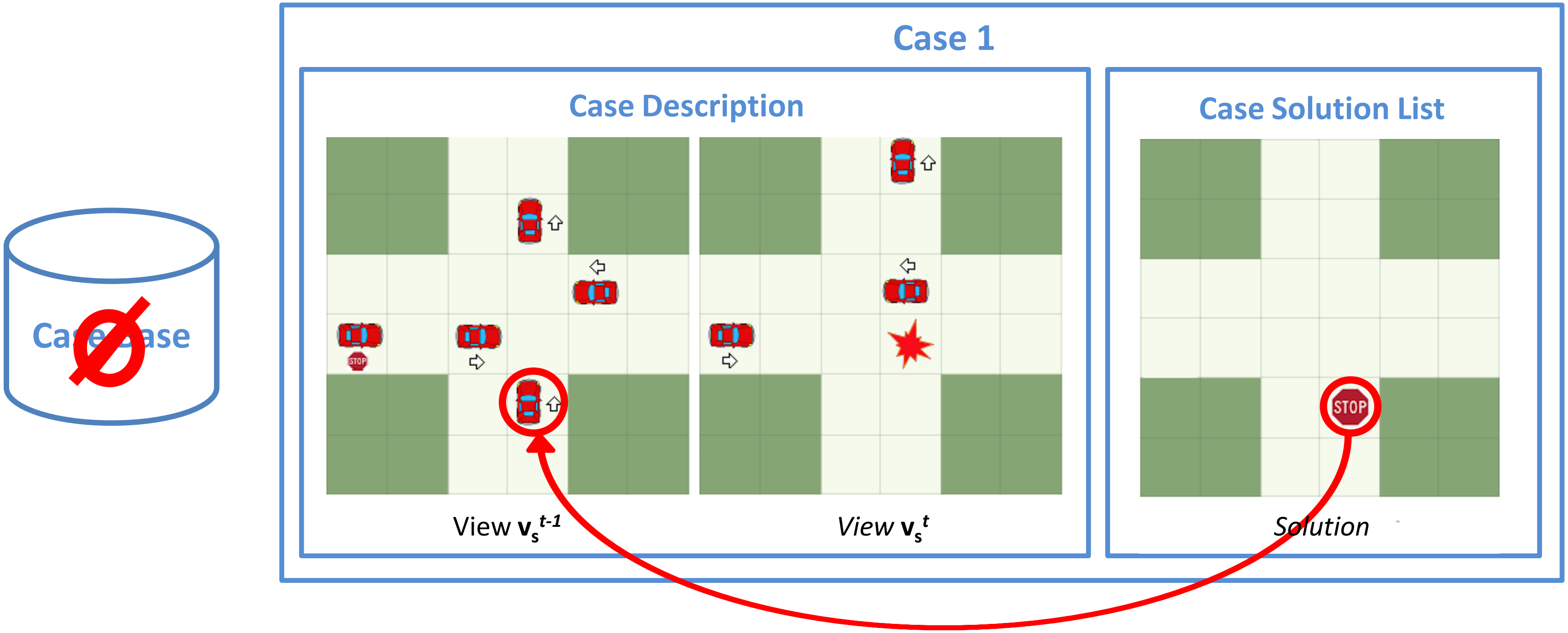


View $\mathbf{v}_s^{t\text{-}1}$ *View* $\mathbf{v}_s^{t}$

4. IRON: Strategy. Norm synthesis

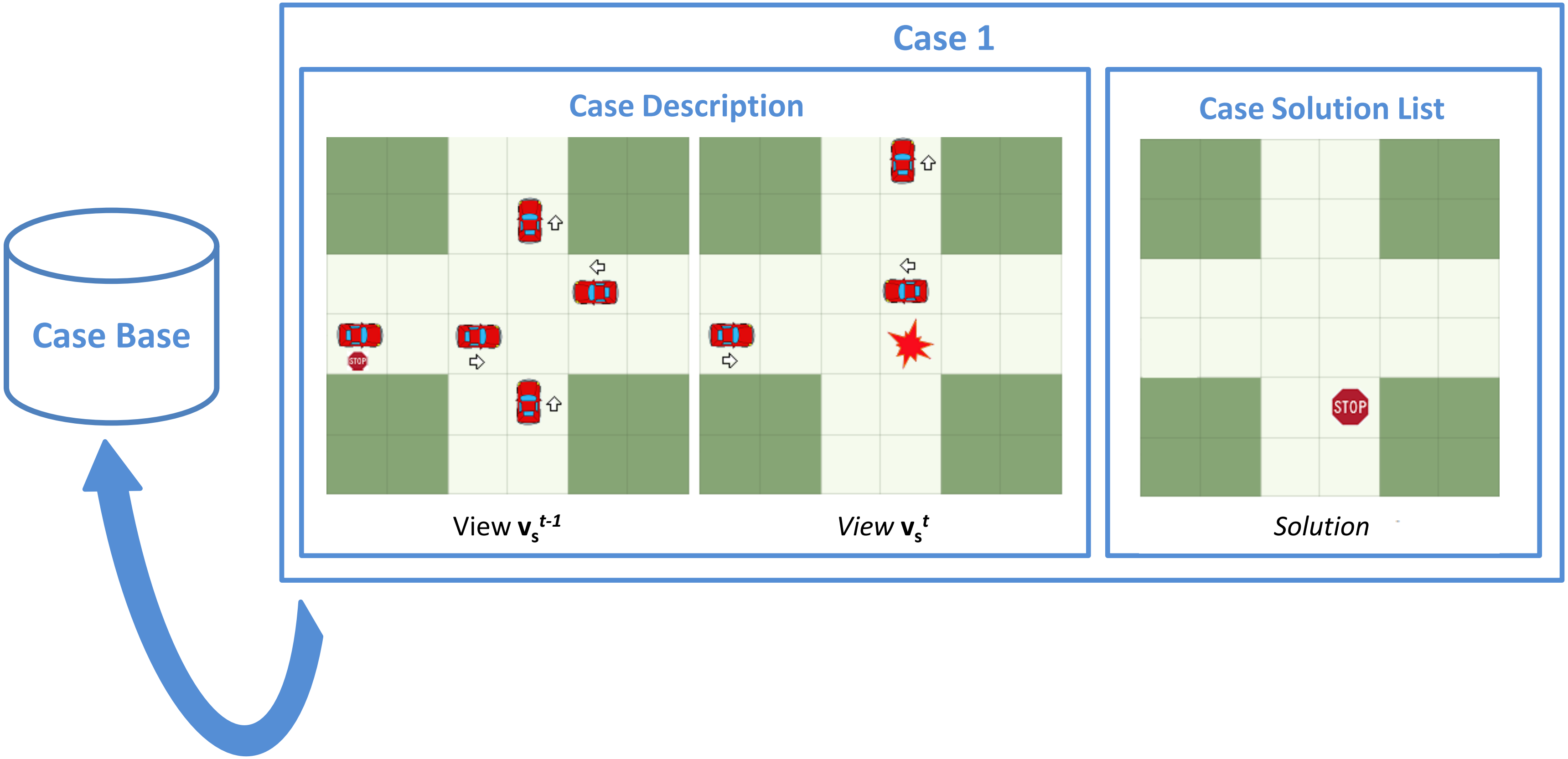


4. IRON: Strategy. Norm synthesis



STOP obligation to ONE of collided cars

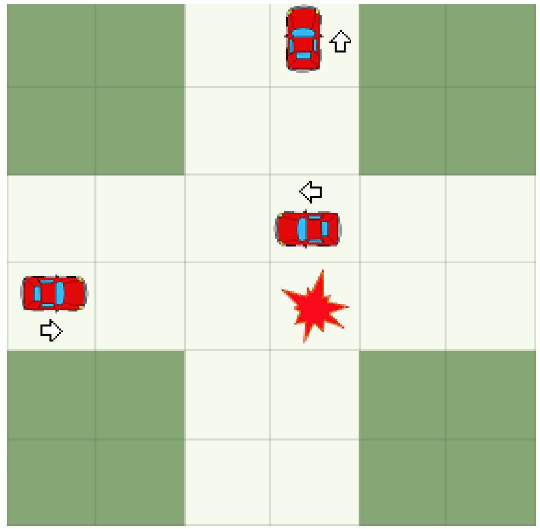
4. IRON: Strategy. Norm synthesis



4. IRON: Strategy. Norm synthesis

Consider the following conflict:

View at time *t-1*



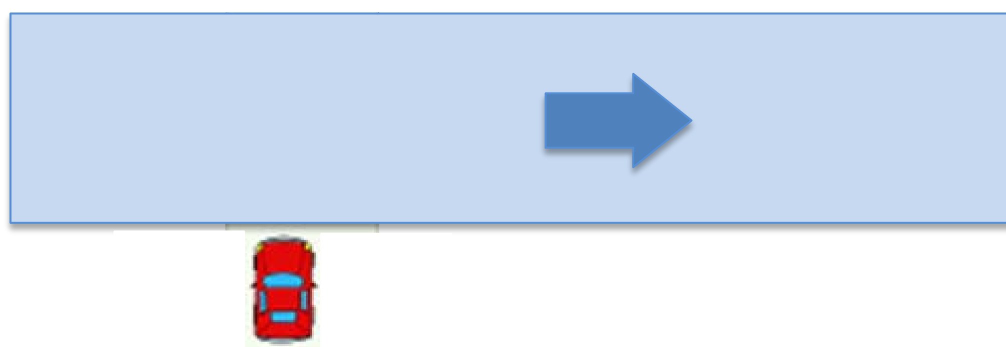
View at time *t*

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

Agent actions ($t\text{-}1 \rightarrow t$):

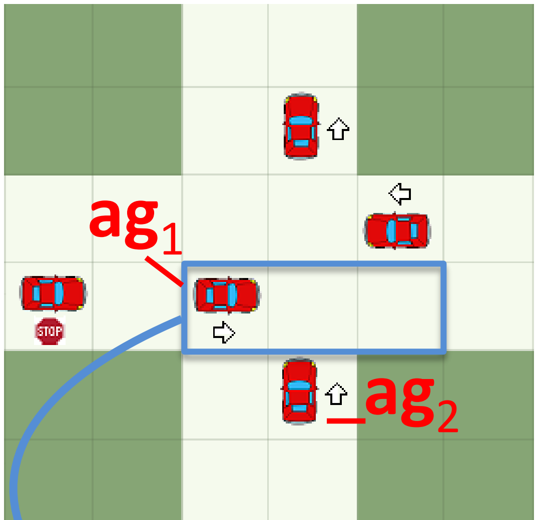
$\{ag_1\text{: Go}, ag_2\text{: Go}\}$

New norm

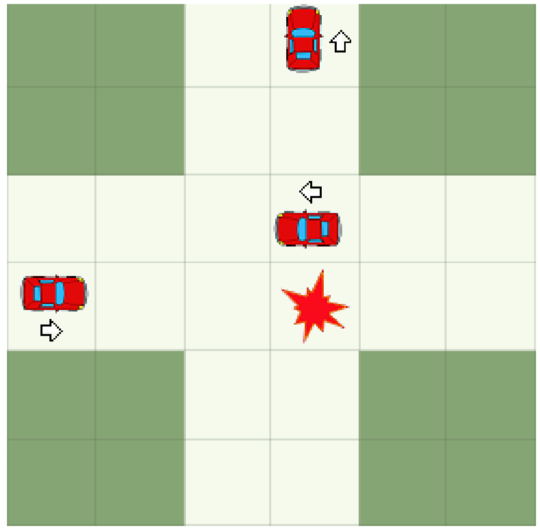


4. IRON: Strategy. Norm synthesis

Consider the following conflict:



View at time *t-1*



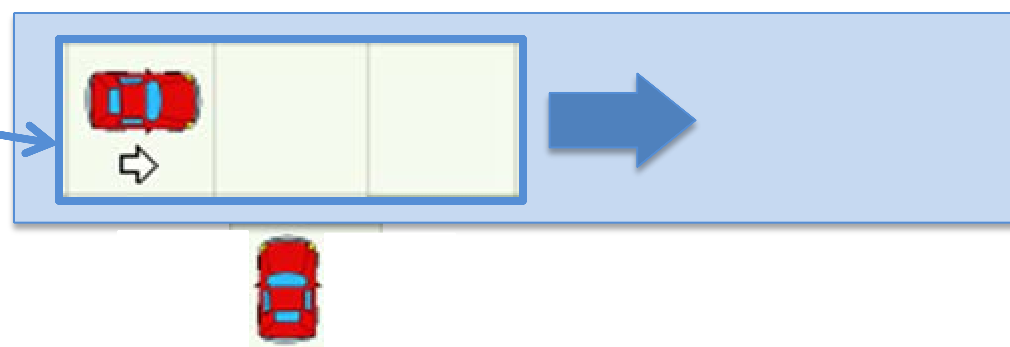
View at time *t*

Conflicting agents: {ag$_1$, ag$_2$}

Agent actions (*t-1* → *t*):

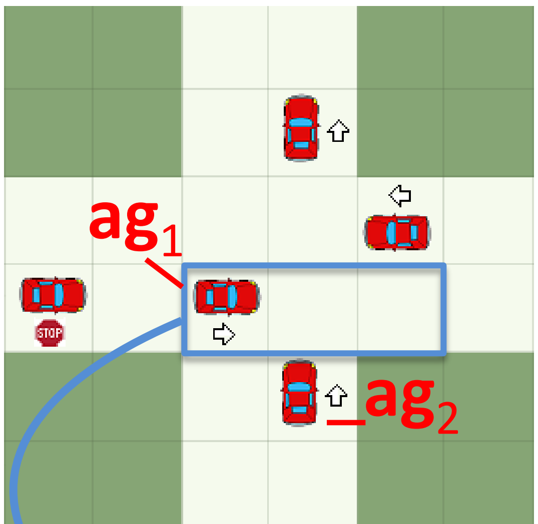
{ag$_1$: Go, ag$_2$: Go}

New norm

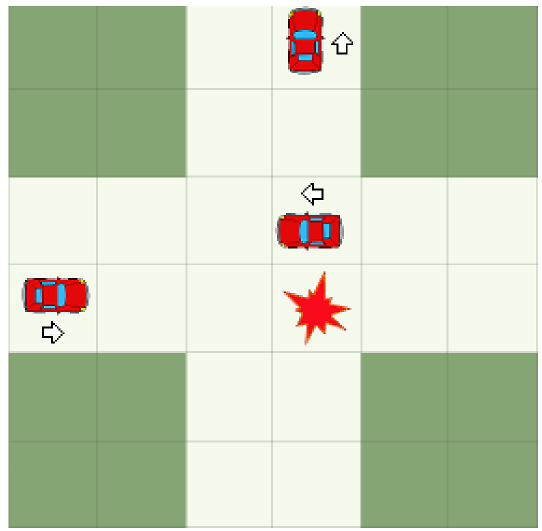


4. IRON: Strategy. Norm synthesis

Consider the following conflict:



View at time *t-1*



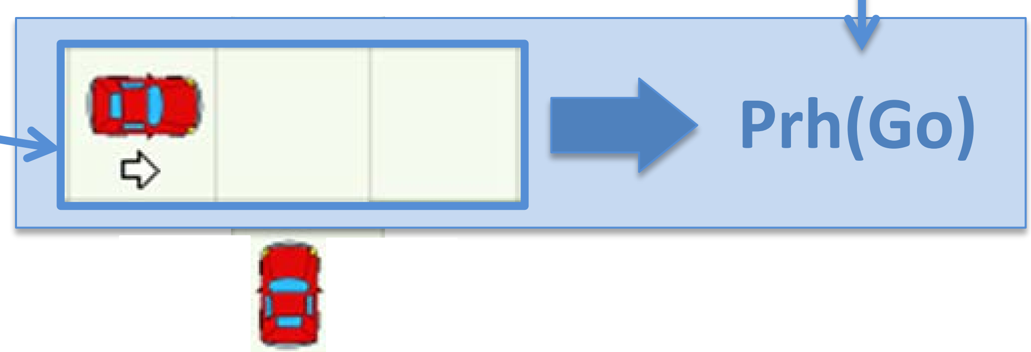
View at time *t*

Conflicting agents: {ag_1, ag_2}

Agent actions (t-1 → t):

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

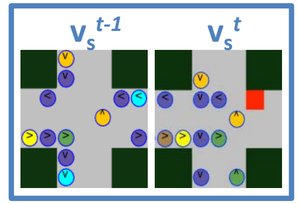
New norm



4. IRON: Strategy. Norm synthesis

Initially empty
Case Base

Case Base ∅



New Conflict description arrives

Search for a similar situation

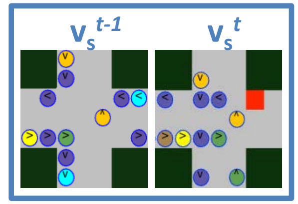
No similar case is found

Generate a new random solution

Store new case

4. IRON: Strategy. Norm synthesis

Initially empty
Case Base



New Conflict description arrives

Search for a similar situation

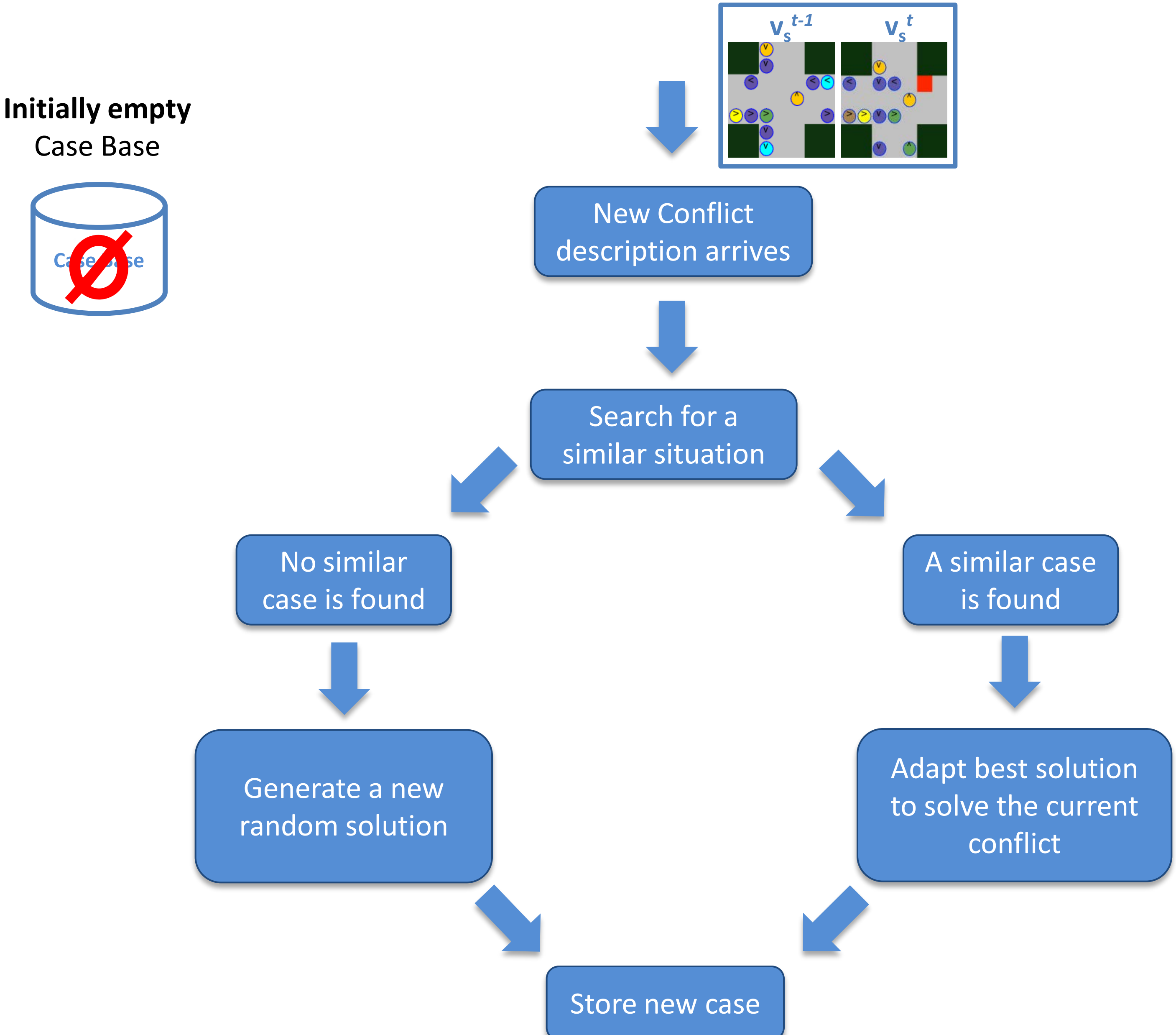
No similar case is found

A similar case is found

Generate a new random solution

Store new case

4. IRON: Strategy. Norm synthesis



4. IRON: Strategy

1. **Conflict detection** from perceptions of the MAS.

2. **Norm Synthesis:** For each detected conflict, it uses operator **create** to synthesise norms.

3. **Norm evaluation:** Evaluates norms effectiveness and necessity based on their outcomes in the MAS.

4. **Norm refinement:** Deactivates ineffective and unnecessary norms by means of operator **deactivate**. Generalises and specialises norms using operators **generalise** and **specialise**.

4. IRON: Norm evaluation

Norms are evaluated based on the conflicts that arise after agents apply/violate them.

Consider a car $\mathbf{c_1}$ and a norm $\mathbf{n_1}$ to avoid collisions

a. If $\mathbf{c_1}$ applies $\mathbf{n_1}$ and **collides** → *Ineffective norm.*
b. If $\mathbf{c_1}$ applies $\mathbf{n_1}$ and **does not** collide → *Effective norm.*
c. If $\mathbf{c_1}$ violates $\mathbf{n_1}$ and **does not** collide → *Unnecessary norm.*
d. If $\mathbf{c_1}$ violates $\mathbf{n_1}$ and **collides** → *Necessary norm.*

Effectiveness

$$\mu_{eff}(n,t) = (1-\alpha)\cdot\mu_{eff}(n,t-1) + \alpha\cdot r_{eff}(n,t)$$

$$r_{eff}(n,t)\frac{\omega_{A\bar{C}}\cdot m_{A\bar{C}}(n)}{\omega_{A\bar{C}}\cdot m_{A\bar{C}}(n)+\omega_{AC}\cdot m_{AC}(n)}$$

Necessity

$$\mu_{nec}(n,t) = (1-\alpha)\cdot\mu_{nec}(n,t-1) + \alpha\cdot r_{nec}(n,t)$$

$$r_{nec}(n,t)\frac{\omega_{VC}\cdot m_{VC}(n)}{\omega_{VC}\cdot m_{VC}(n)+\omega_{V\bar{C}}\cdot m_{V\bar{C}}(n)}$$

4. IRON: Strategy

1. **Conflict detection** from perceptions of the MAS.

2. **Norm Synthesis:** For each detected conflict, it uses operator **create** to synthesise norms.

3. **Norm evaluation:** Evaluates norms effectiveness and necessity based on their outcomes in the MAS.

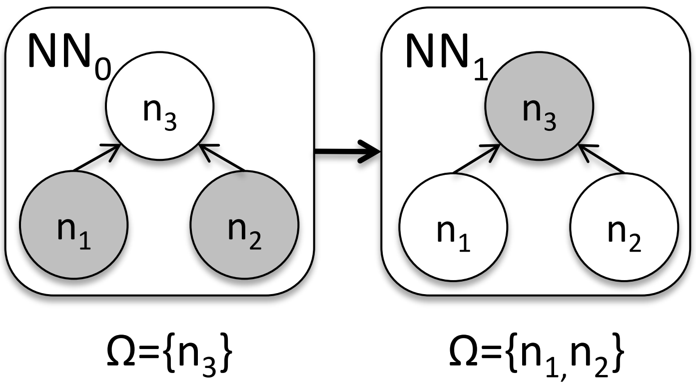
4. **Norm refinement:** Deactivates ineffective and unnecessary norms by means of operator **deactivate**. Generalises and specialises norms using operators **generalise** and **specialise**.

4. IRON: Strategy. Norm refinement

For each norm:

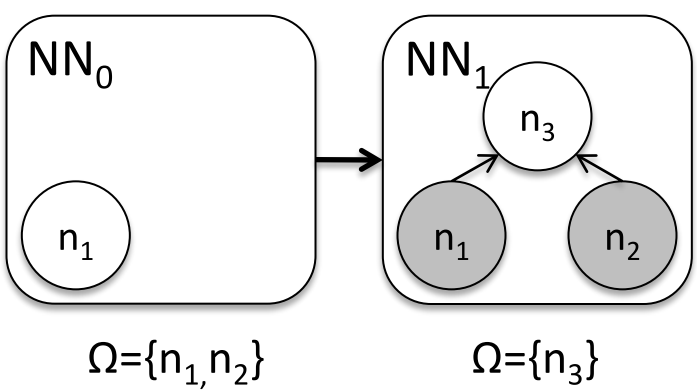
1. **IF** its effectiveness **OR** necessity is under a deactivation threshold, then **specialise** the norm if it is general, or **deactivate** it if it's a leave.

specialise

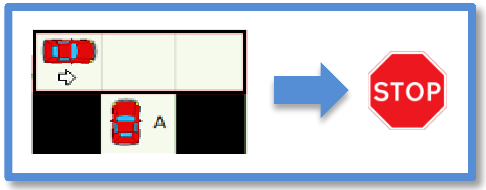


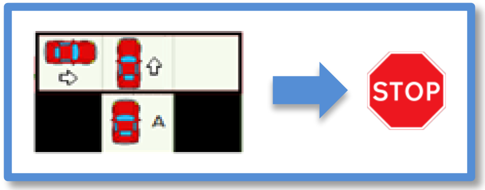
2. **IF** its effectiveness **AND** necessity is above a generalisation threshold, then try to **generalise**.

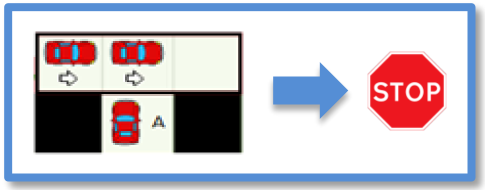
generalise



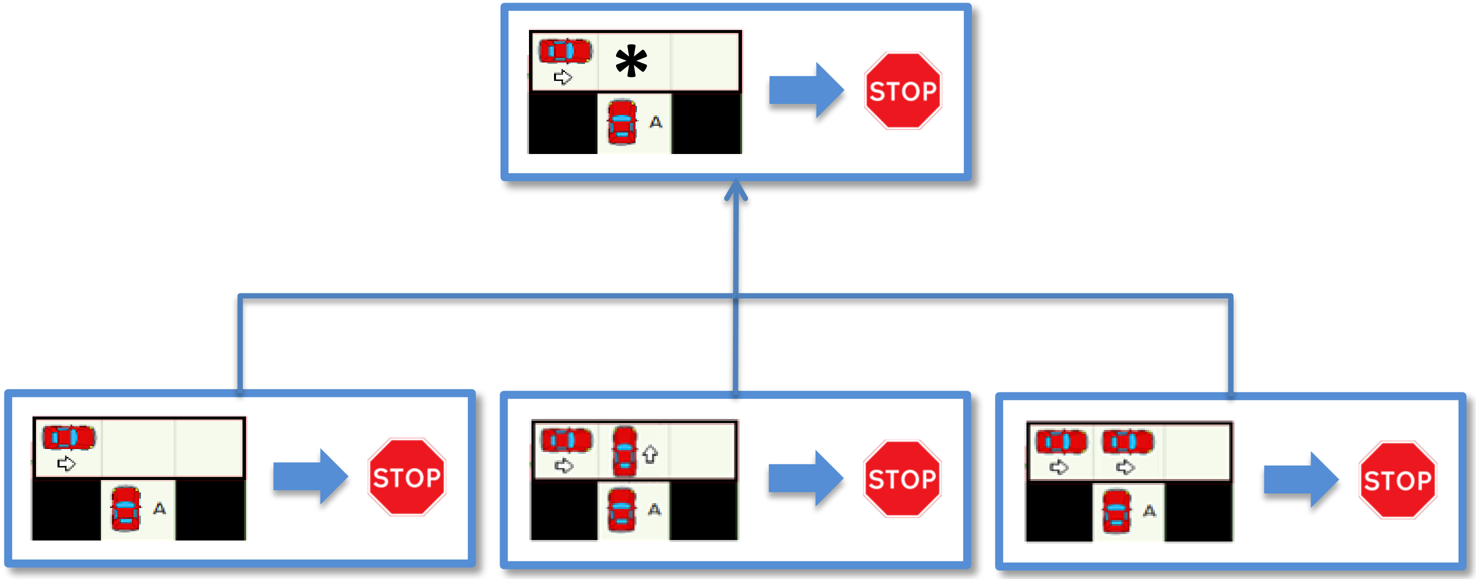
4. IRON: Strategy. Norm generalisation



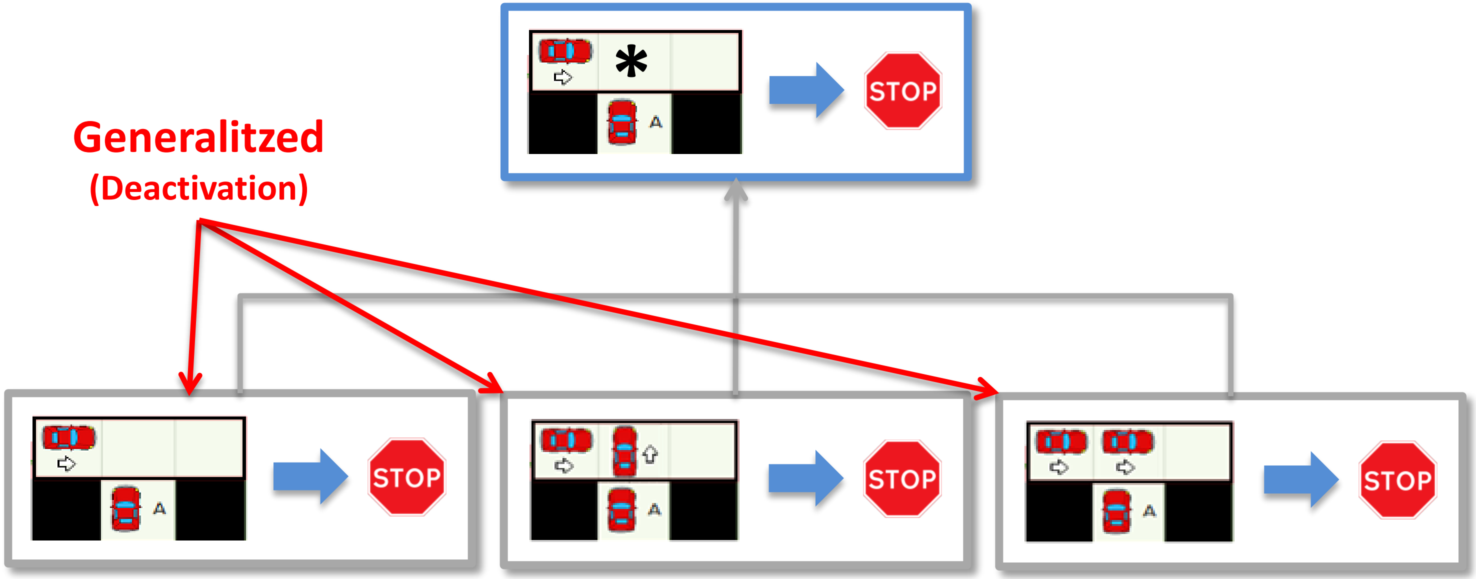




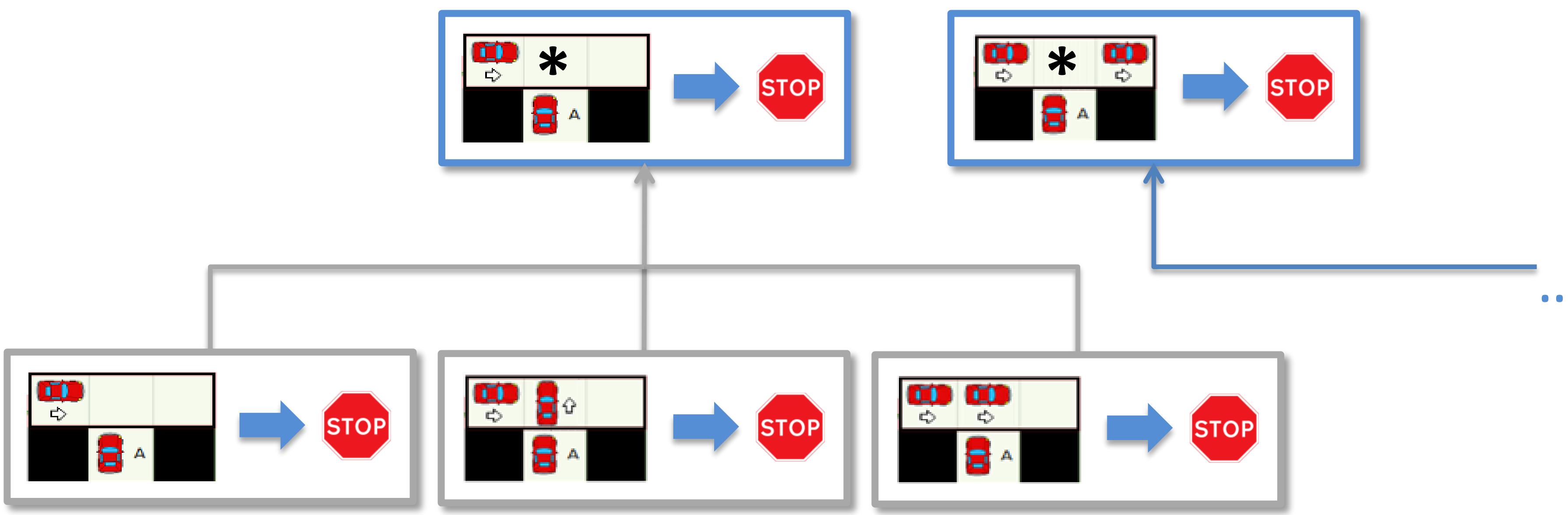
4. IRON: Strategy. Norm generalisation example



4. IRON: Strategy. Norm generalisation example

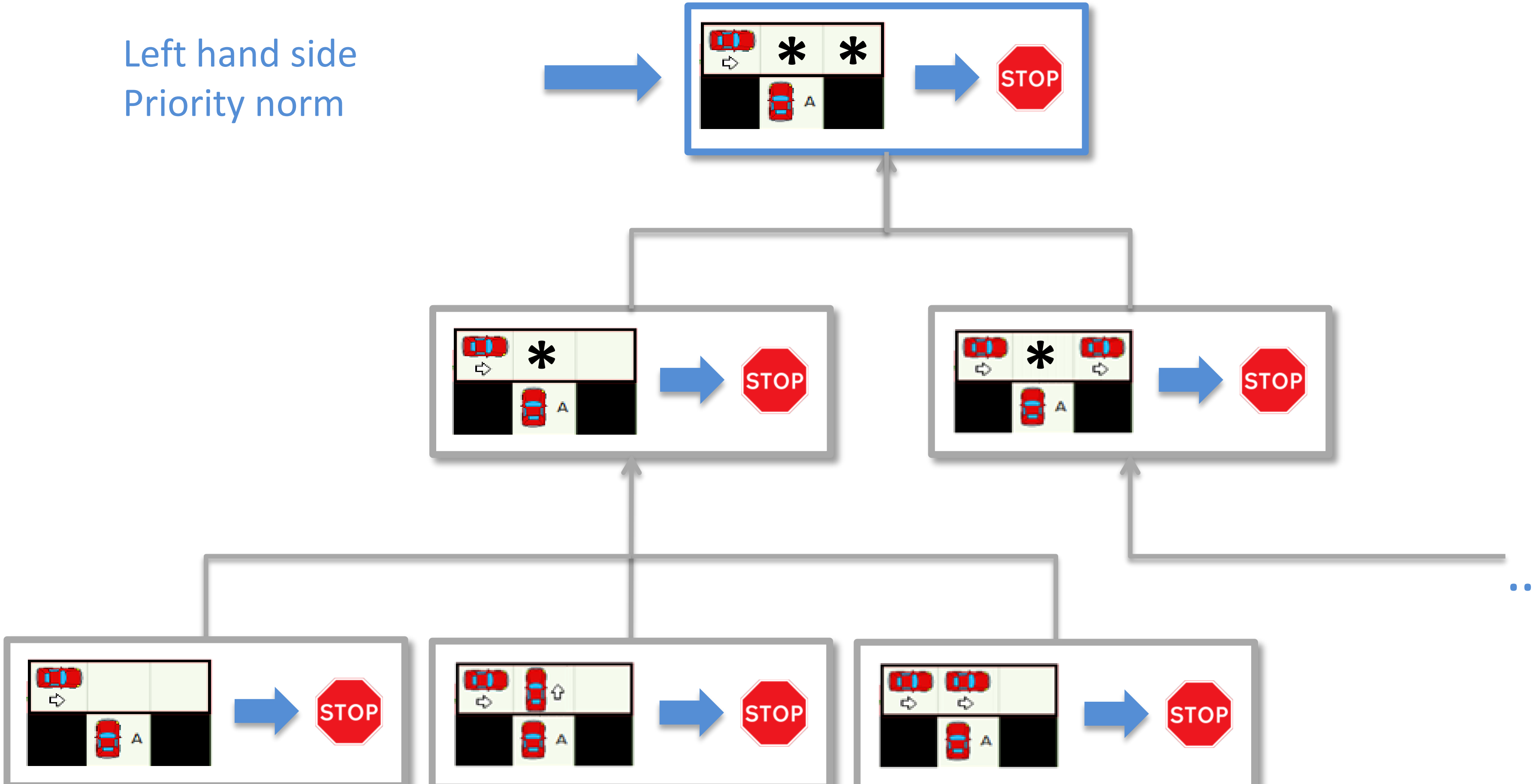


4. IRON: Strategy. Norm generalisation example

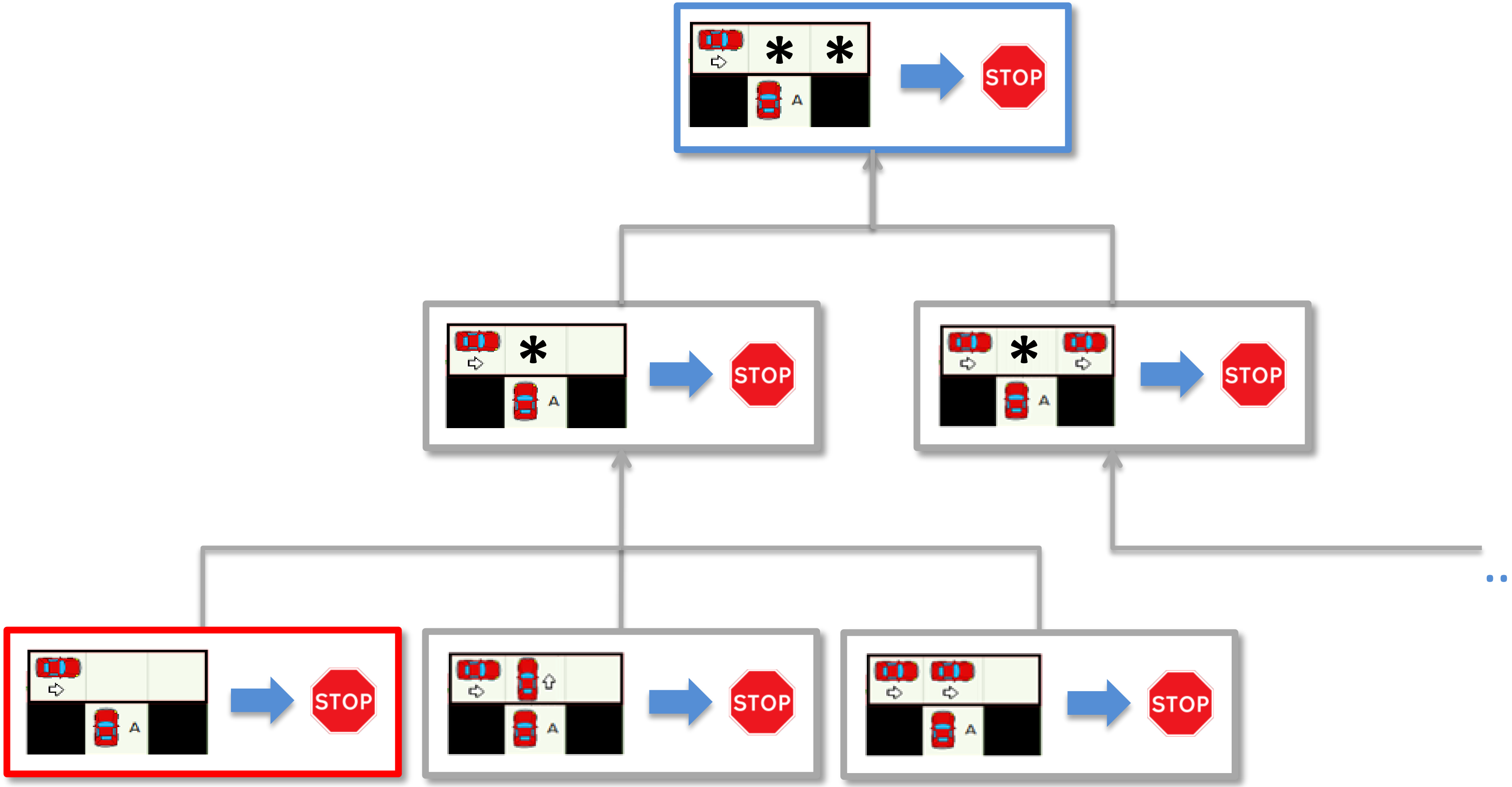


4. IRON: Strategy. Norm generalisation example

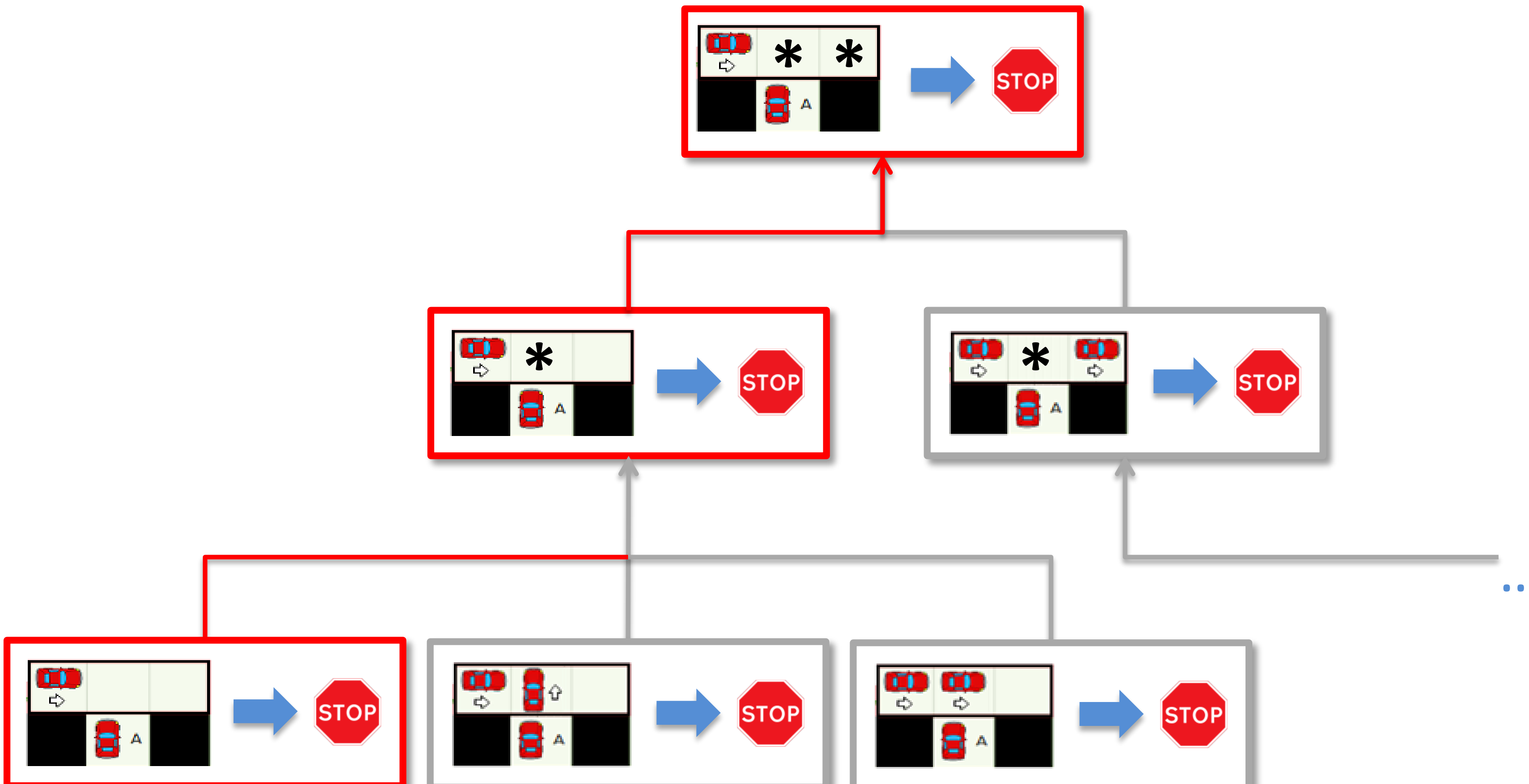
Left hand side
Priority norm



4. IRON: Strategy. Norm specialisation example



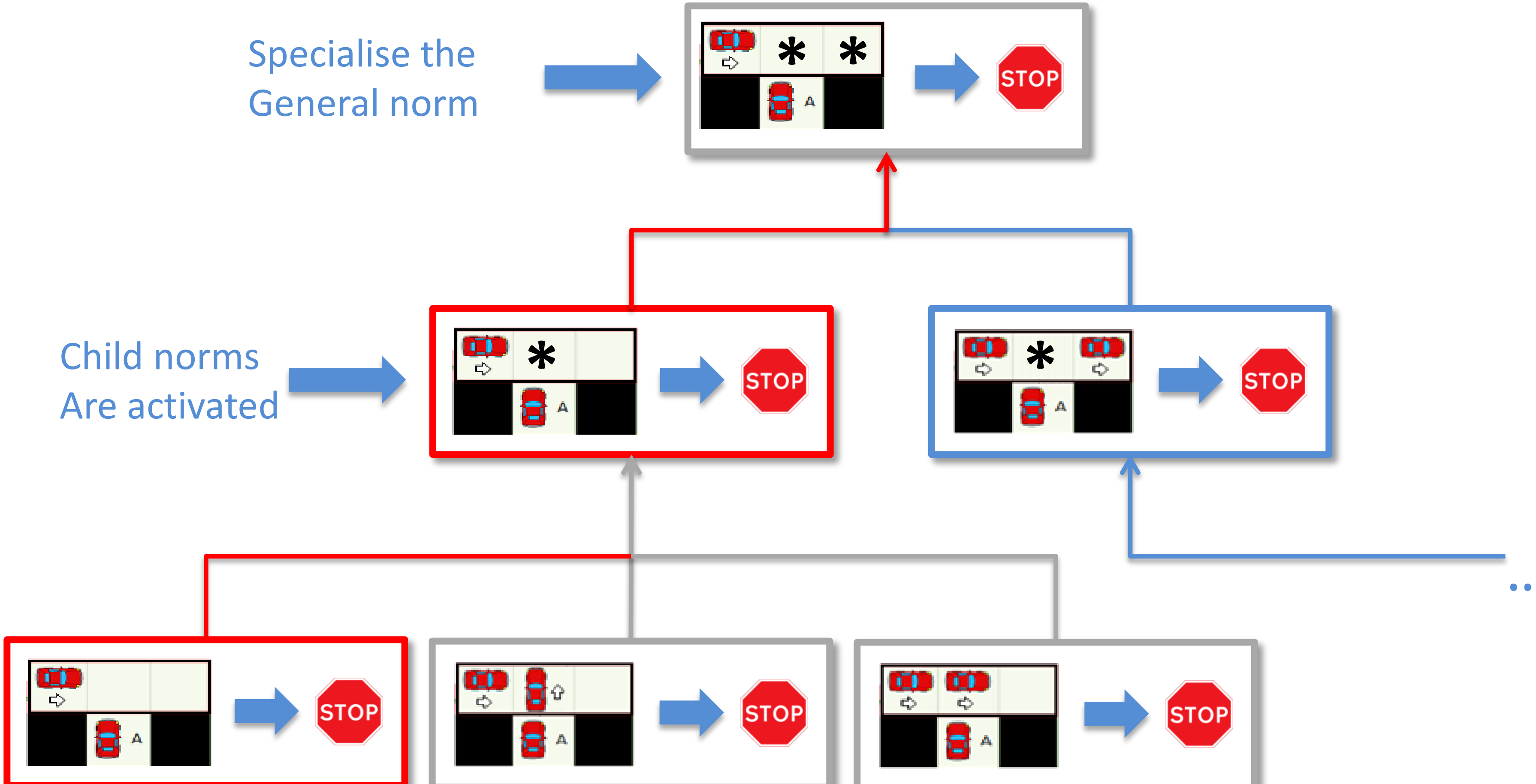
4. IRON: Strategy. Norm specialisation example



4. IRON: Strategy. Norm specialisation example

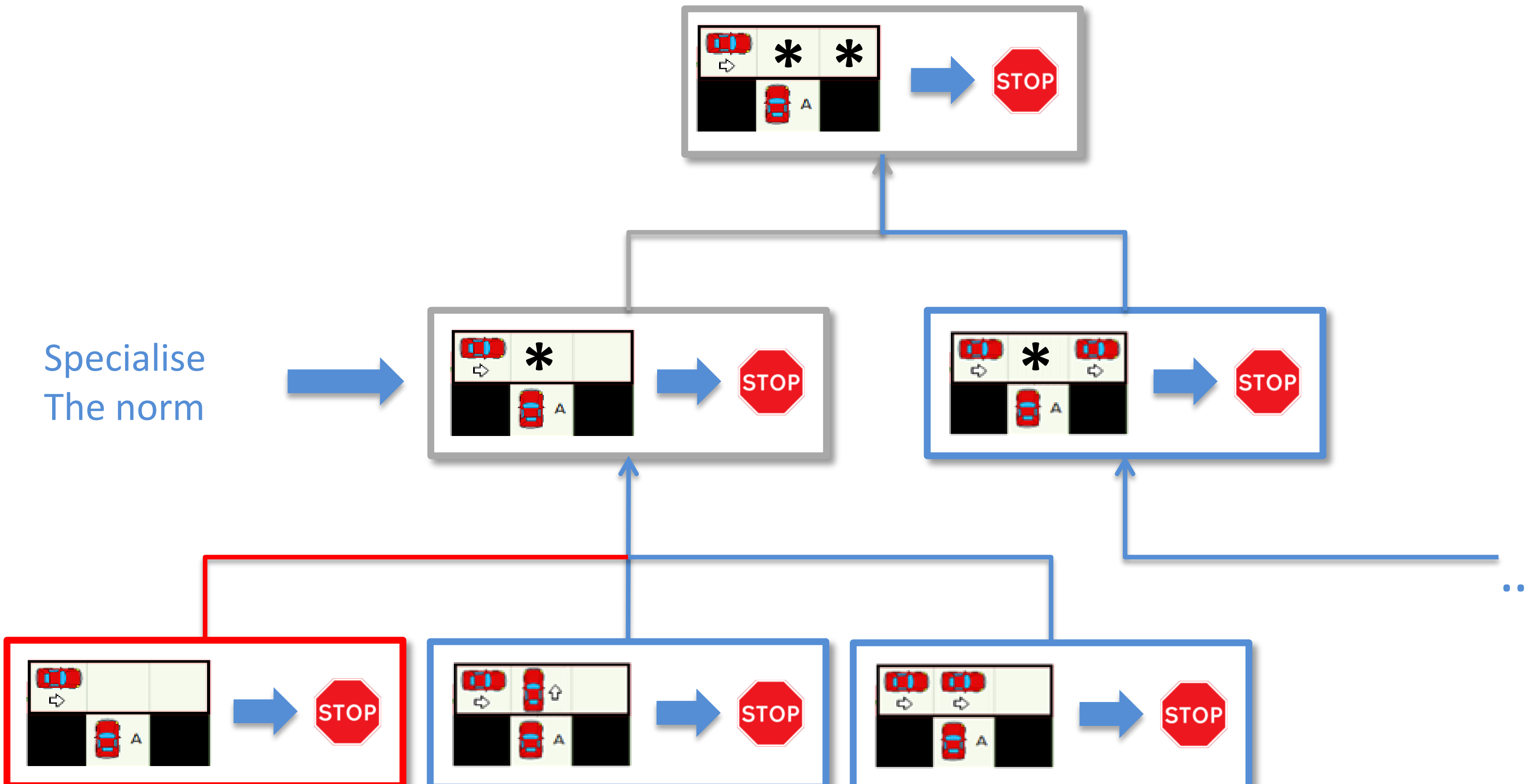
Specialise the
General norm

Child norms
Are activated

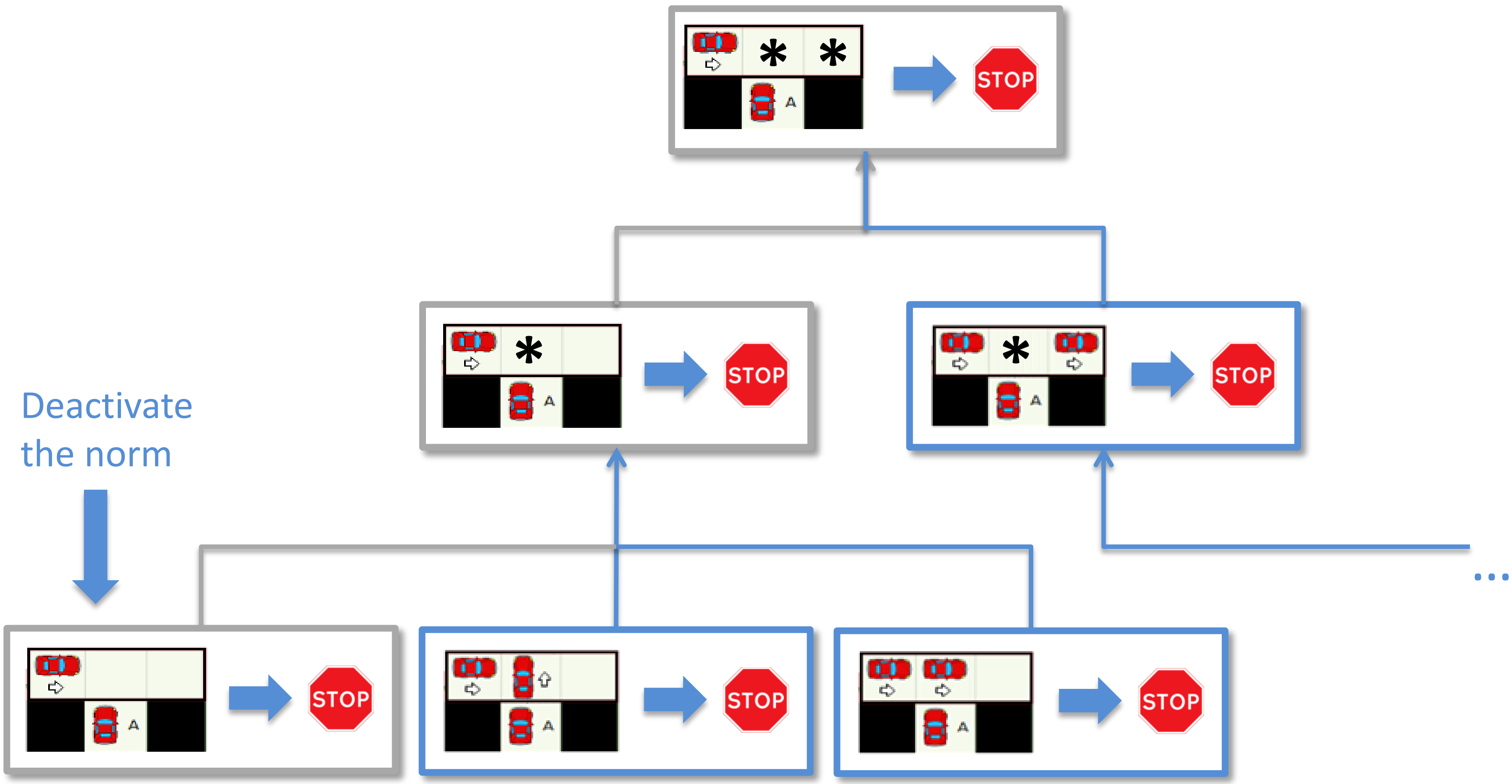


4. IRON: Strategy. Norm specialisation example

Specialise
The norm

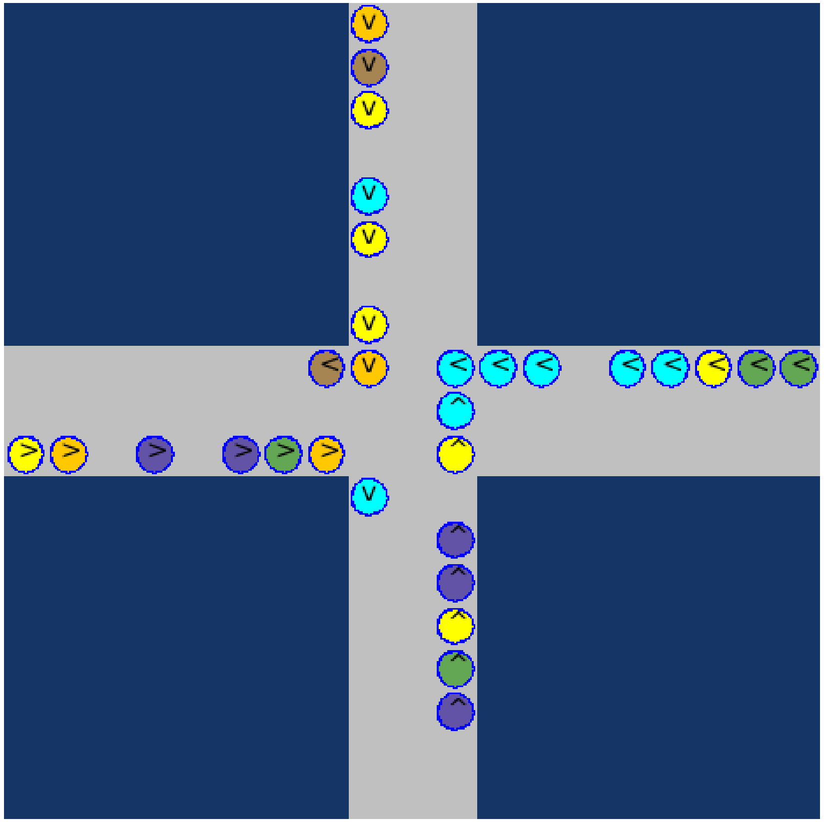


4. IRON: Strategy. Norm specialisation



5. Evaluation: The traffic intersection scenario

In this simple scenario we may synthesise many candidate norms…

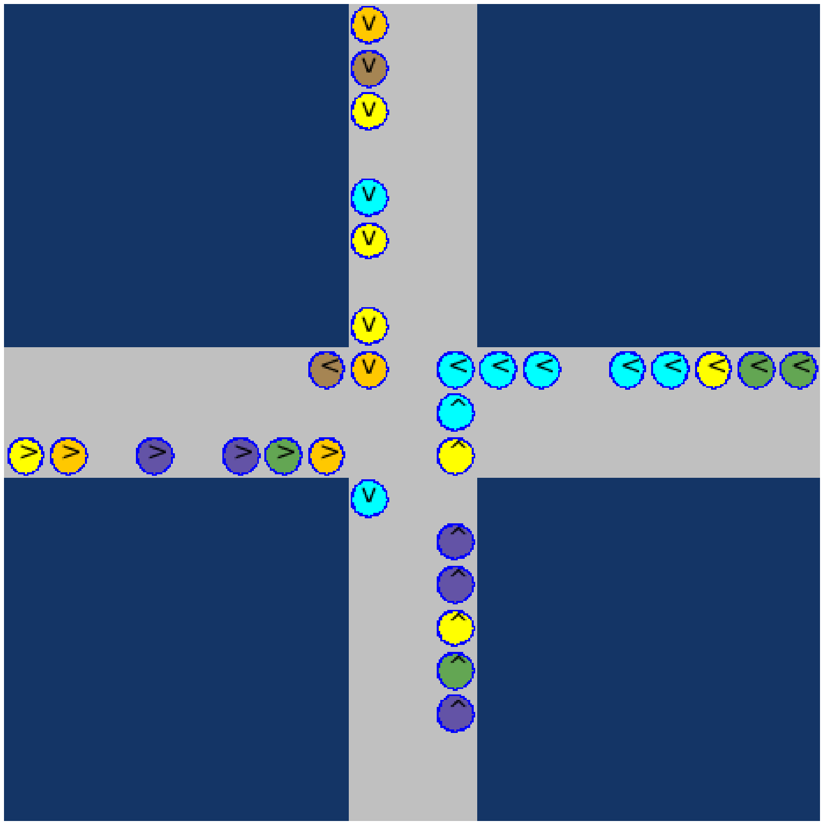


1. Give way to left.
2. Give way to right.
3. Keep security distance.
4. Stop always.
5. Never stop.
6. Stop when you perceive a car behind you.
7. …

What **combination** of candidate norms (i.e., normative system) may achieve MAS goals?

5. Evaluation: The traffic intersection scenario

In this simple scenario we may synthesise many candidate norms…



1. Give way to left.
2. Give way to right.
3. Keep security distance.
4. Stop always.
5. Never stop.
6. Stop when you perceive a car behind you.
7. …

In this scenario: **216** candidate norms → 2^{216} = $\mathbf{10^{65}}$ candidate normative systems.

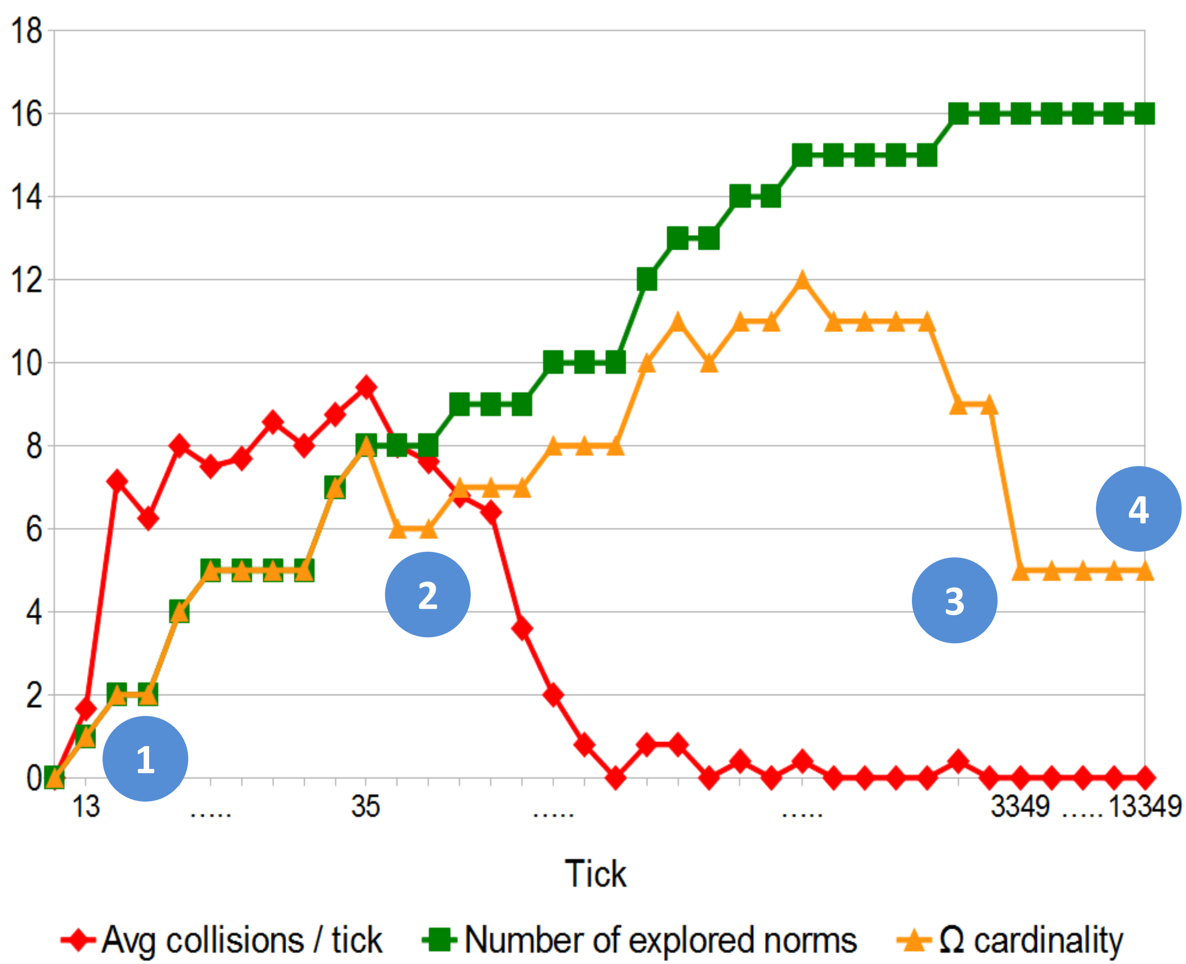
5. Empirical evaluation

1. A **typical execution** of the norm synthesis process.
   - **IRON successfully synthesises normative systems that avoids collisions.**

2. A **robustness analysis** that the tolerance of IRON to non-compliant behaviour (norm violations).
   - **IRON synthesises normative systems even for high norm violation rates.**

5. Empirical evaluation

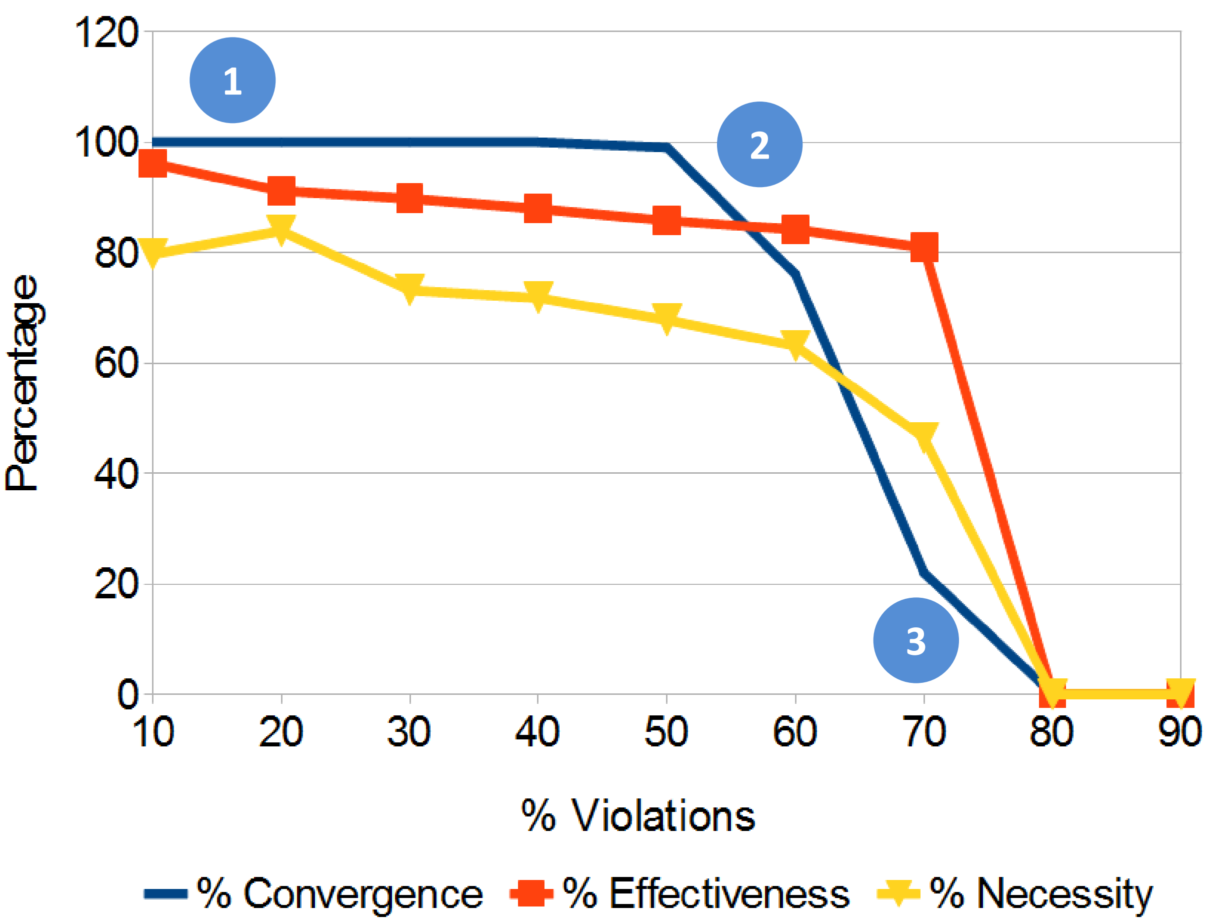
Prototype Execution



1. Tick **13:** first collisions arise and IRON synthesises first norms.
2. Tick **35:** IRON **generalises** norms.
3. Tick **3349**: Cardinality of the normative system reduced to 5 norms. Collisions are avoided.
4. Tick **13349**: Simulation stops because of convergence.

5. Empirical evaluation

Robustness Analysis



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

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

3 **Very high violation rates (70%-90%)** IRON converges for 20% of the simulation runs despite a 70% violation rate. Norms cannot be synthesised beyond 80% violation rate.

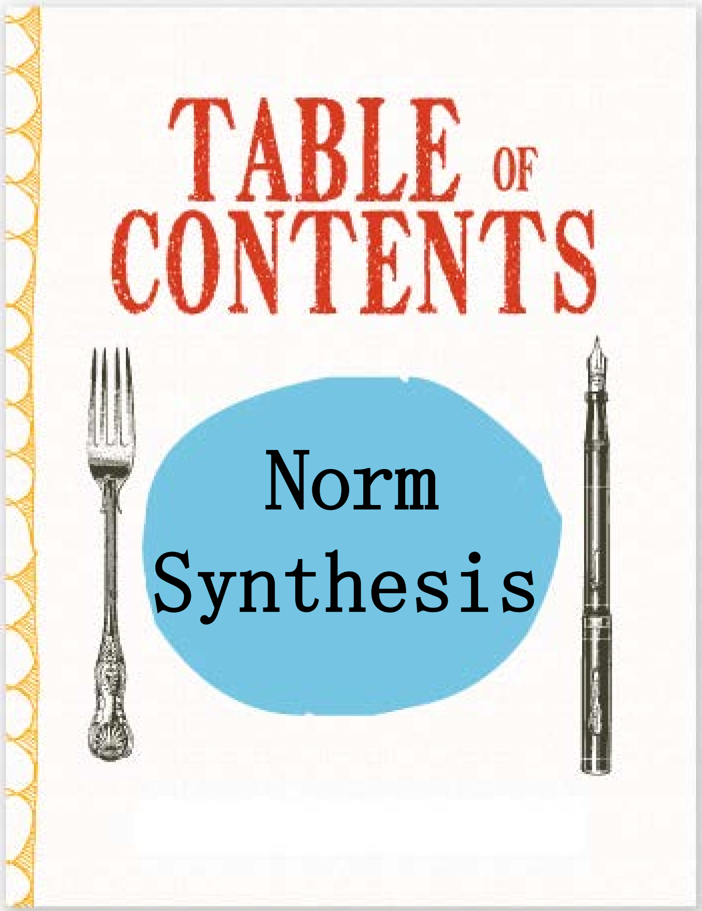
6. Conclusions

- We have contributed to the automated synthesis of normative systems.
- Our norm synthesis mechanism is based on:
  - A set of core synthesis operators.
  - Effectiveness and necessity as the means of evaluating norms.
- Empirical evaluation shows that IRON successfully synthesises norms even in presence of non-compliant behaviour.

Future work

- To investigate further relationships between norms in the normative network.
- Extend the norm synthesis process to create norms with sequences of views (t-n, …, t) instead of (t-1, t).

Tutorial Outline



1. Introduction to Norms and Normative MAS.
2. Overview of approaches to norm synthesis.
3. **On-line automatic norm synthesis.**
   – AAMAS 2013.
   – **AAMAS 2014**
4. Demo and hands-on activity

Minimality and Simplicity in the On-line Automated Synthesis of Normative Systems

Javier Morales, Maite López-Sánchez, Juan A. Rodríguez-Aguilar, Michael Wooldridge, Wamberto Vasconcelos

Introduction

- Individuals interacting cause **conflicts (undesired states).**
- **Norms** are enacted to avoid conflicts.
- **Running example:** Road traffic.
- Norms can be employed to avoid undesirable states (i.e., conflicts) in Multi-Agent Systems.

Research problem

How to synthesise a normative system that is
good enough to regulate a **dynamic** Multi-Agent System?

Research problem

How to synthesise a normative system that is ***good enough*** to regulate a **dynamic** Multi-Agent System?

1

Avoids **conflicts**

2

Avoids
over regulation

3

***Easy* to reason**
about

Research problem and approach

An **on-line** norm synthesis strategy to synthesise **conflict-free** and **compact** normative systems.

The **compactness** of a normative system is measured by:

- **Minimality**: Size of the normative system.
- **Simplicity**: Size of its individual norms.

1

Avoids **conflicts**

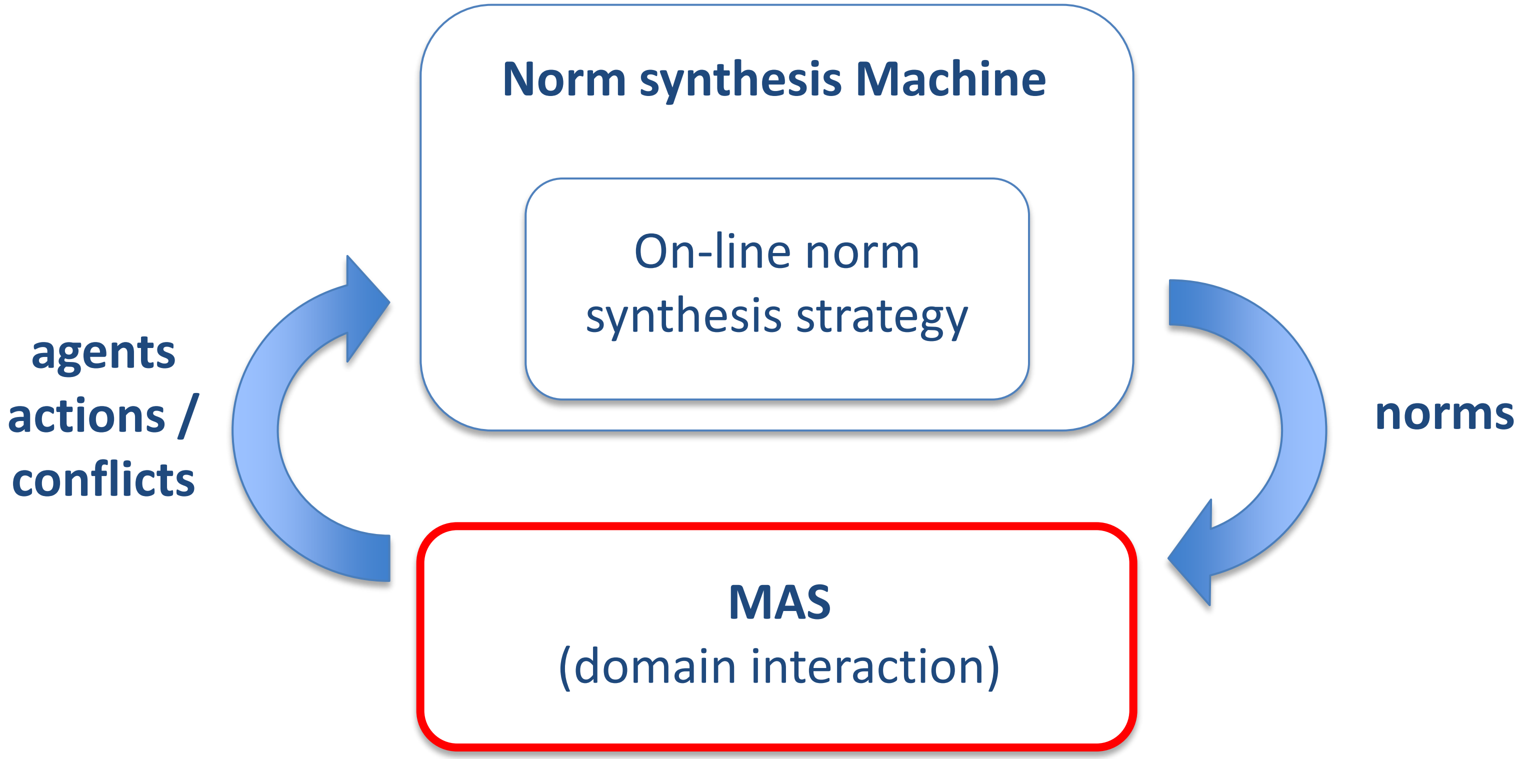
2

Avoids
over regulation

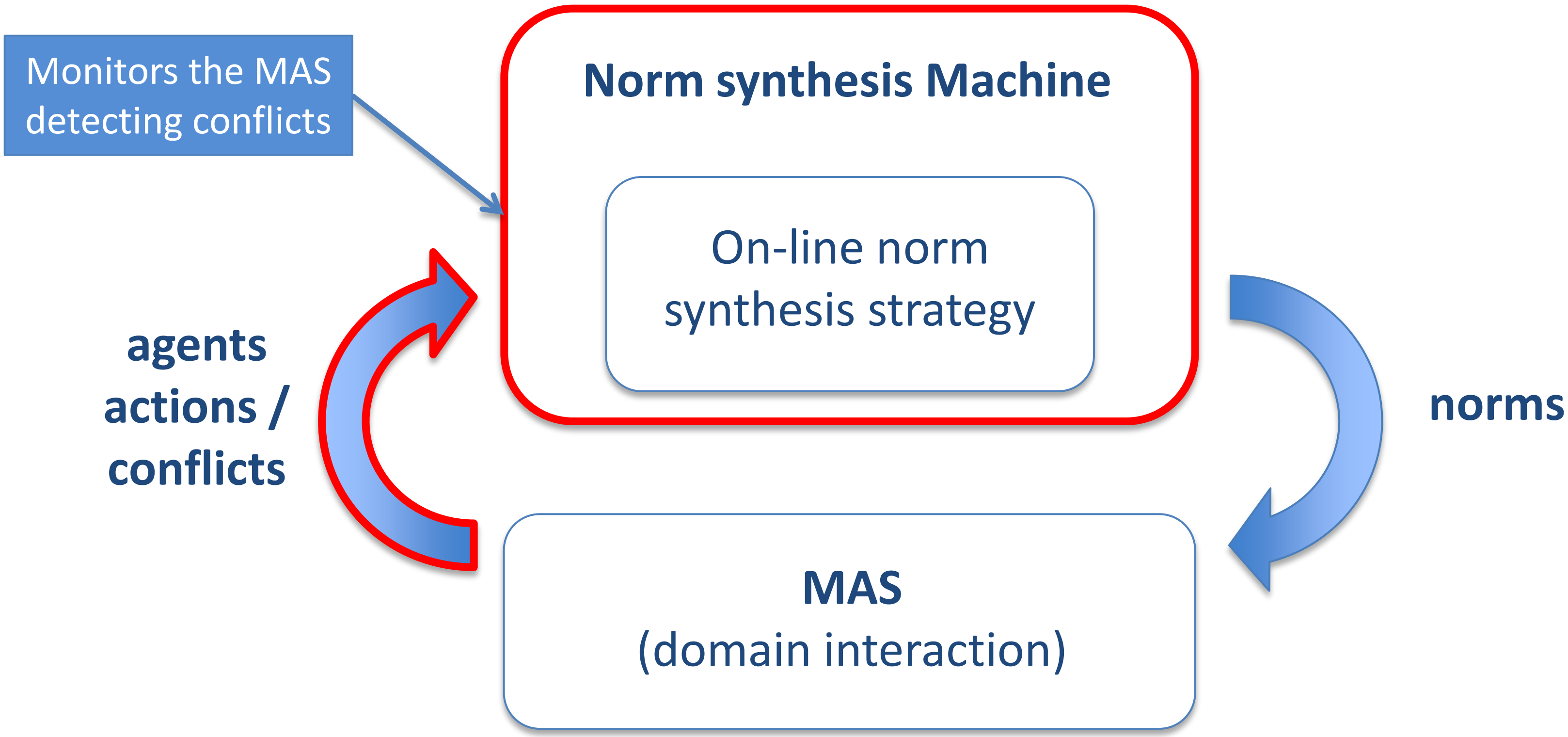
3

***Easy* to reason**
about

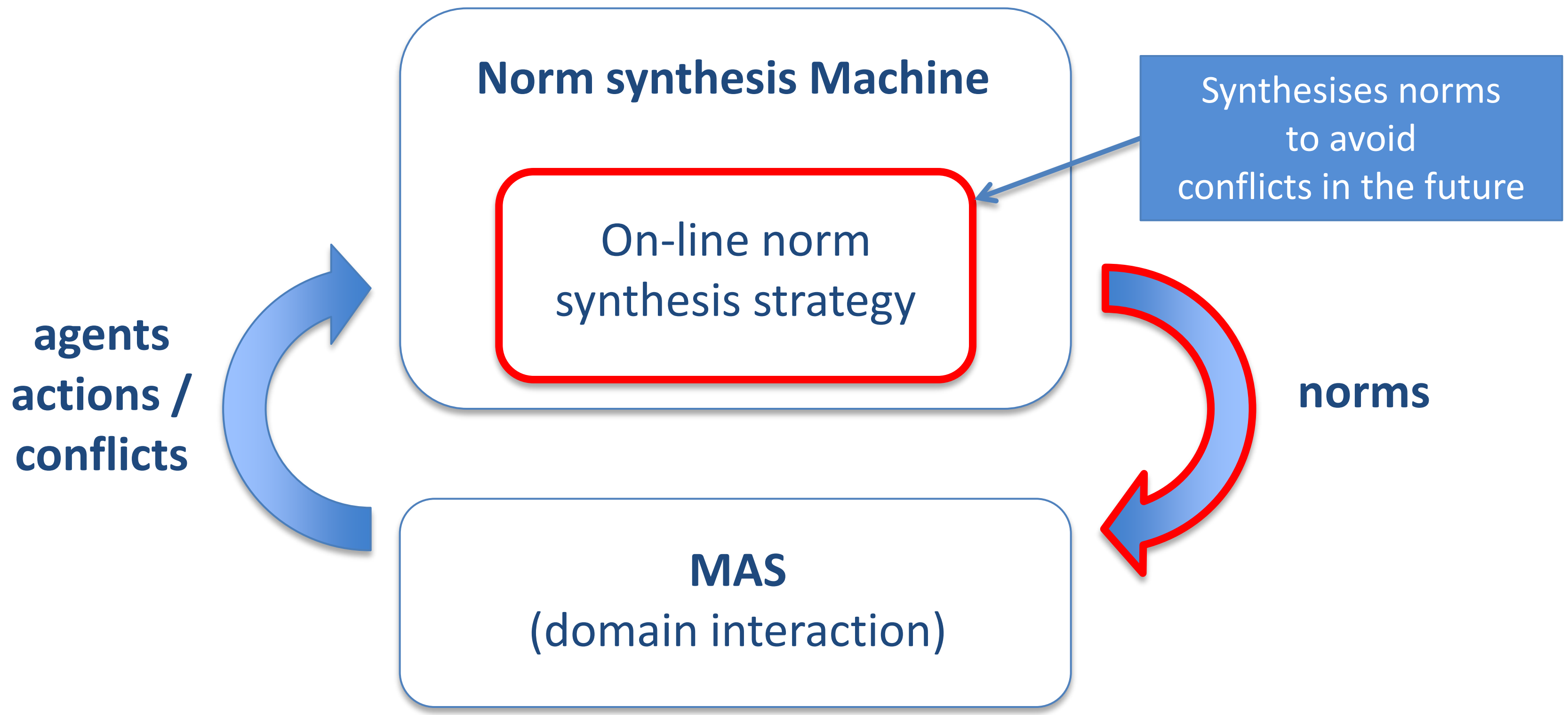
Research problem and approach



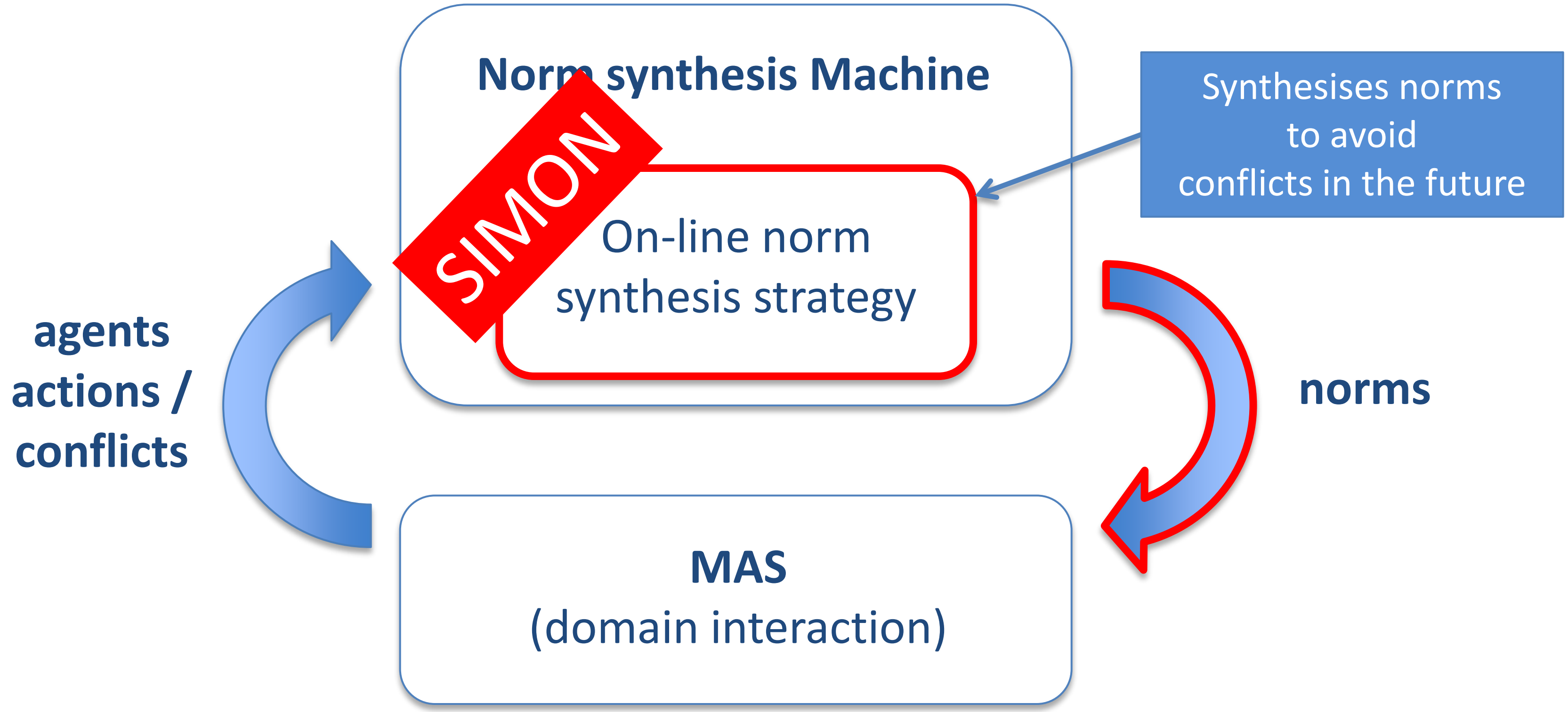
Research problem and approach



Research problem and approach



Research problem and approach



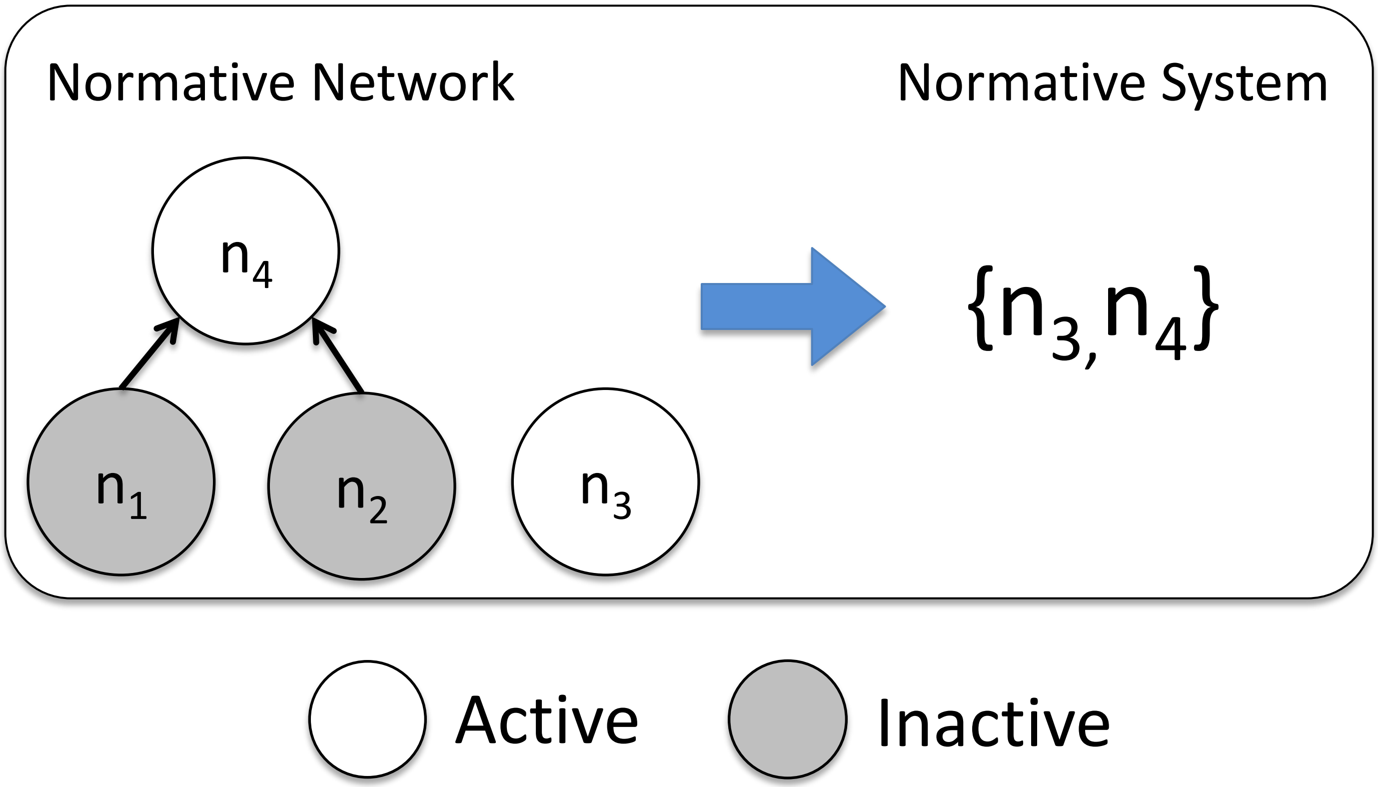
SIMON: An On-line Norm Synthesis strategy

SIMON (*Simple Minimal On-line Norm Synthesis*)

SIMON: An On-line Norm Synthesis strategy

SIMON (*Simple Minimal On-line Norm Synthesis*)

Step 1. For each detected **conflict,** SIMON **generates** a new active norm to avoid it in the future.



SIMON: An On-line Norm Synthesis strategy

SIMON (*Simple Minimal On-line Norm Synthesis*)

Step 1. For each detected **conflict,** SIMON **generates** a new active norm to avoid it in the future.

Step 2. **Evaluation** of each norm in terms of whether it is **effective** and **necessary**:

IF agents **comply with** it, **NO conflicts** arise → **EFFECTIVE**
IF agents **comply with** it, **conflicts** arise → **INEFFECTIVE**

IF agents **infringe** it, **conflicts** arise → **NECESSARY**
IF agents **infringe** it, **NO conflicts** arise → **UNNECESSARY**

SIMON: An On-line Norm Synthesis strategy

SIMON (*Simple Minimal On-line Norm Synthesis*)

Step 1. For each detected **conflict,** SIMON **generates** a new active norm to avoid it in the future.

Step 2. **Evaluation** of each norm in terms of whether it is **effective** and **necessary**.

Step 3. **Refinement** of norms. SIMON performs **optimistic** norm generalisations.

Aims at synthesising **compact** normative systems.

SIMON: An On-line Norm Synthesis strategy

SIMON (*Simple Minimal On-line Norm Synthesis*)

Step 1. For each detected **conflict,** SIMON **generates** a new active norm to avoid it in the future.

Step 2. **Evaluation** of each norm in terms of whether it is **effecti**

Adapted from AAMAS'13

Step 3. **Refinement** of norms. SIMON performs **optimistic** norm generalisations.

Aims at synthesising **compact** normative systems.

SIMON: Norm refinement

Based on three key components:

1. A **taxonomy** of terms to specify norms
2. An **optimistic** approach to norm generalisation
3. Novel norm generalisation **modes**

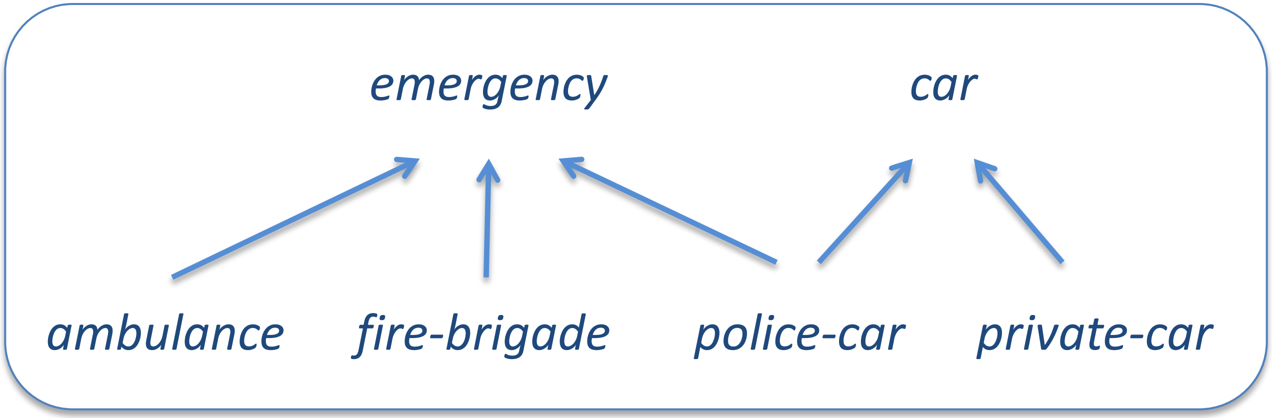
SIMON: Taxonomy of terms to specify norms

Norm examples (informal):

- Give way to **ambulances**
- Give way to **fire brigade**
- Give way to **police cars**

Give way to **emergency vehicles**

The terms employed to specify norms are part of a taxonomy.



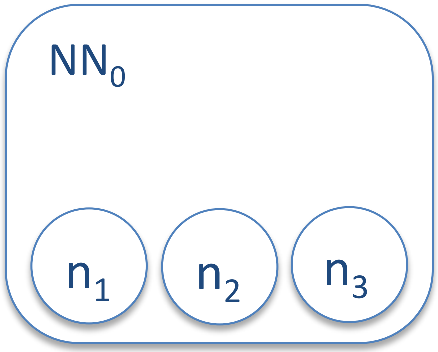
SIMON: Norm refinement

Based on three key components:

1. A **taxonomy** of terms to specify norms
2. An **optimistic** approach to norm generalisation
3. Novel norm generalisation **modes**

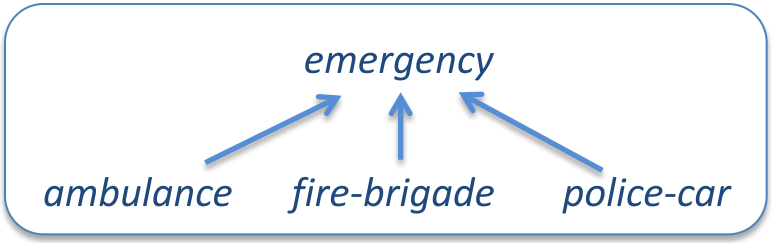
SIMON: Norm generalisation

Norm generalisation

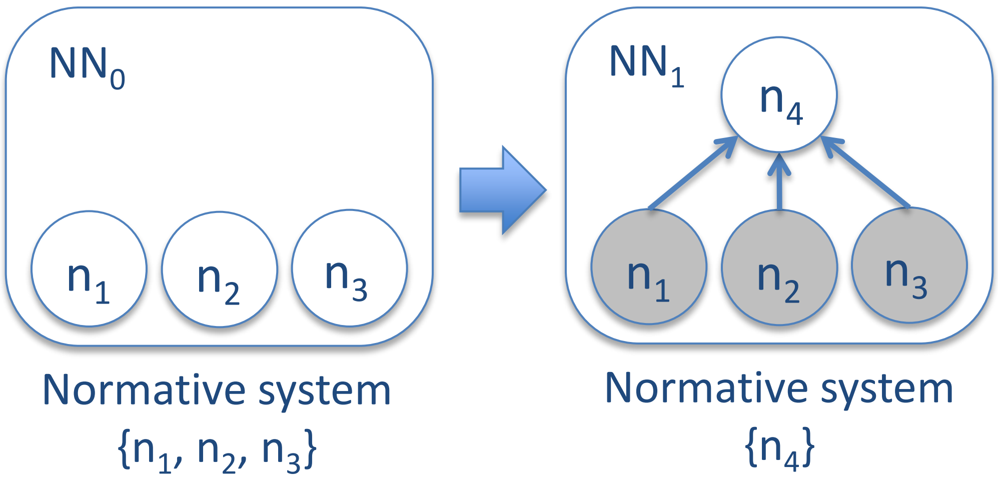


Normative system
$\{n_1, n_2, n_3\}$

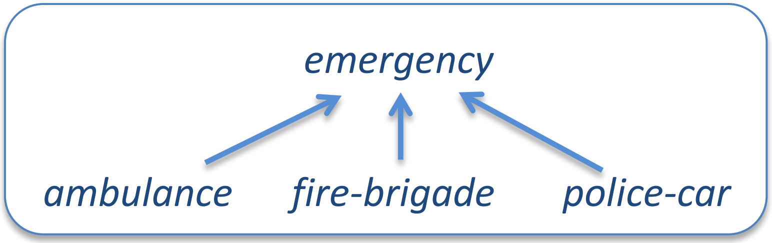
n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**
n_3: Give way to **police cars**



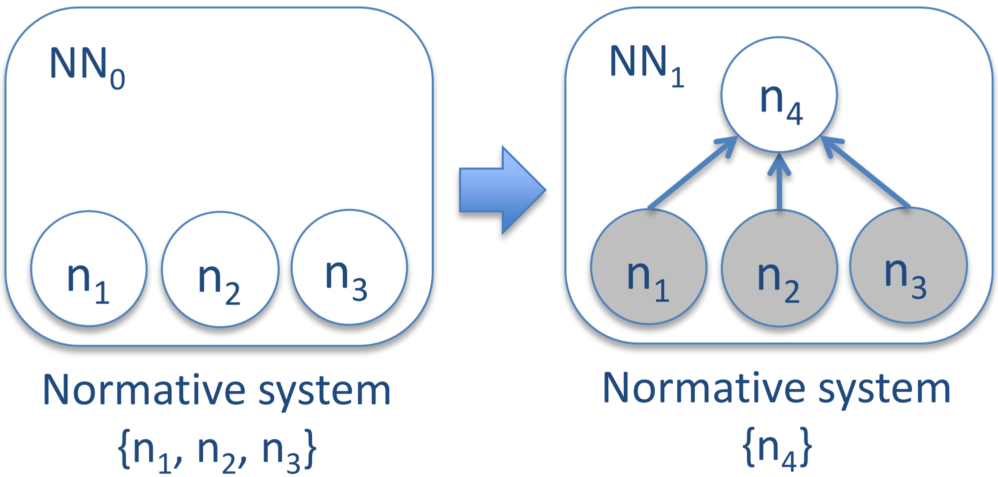
Norm generalisation



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



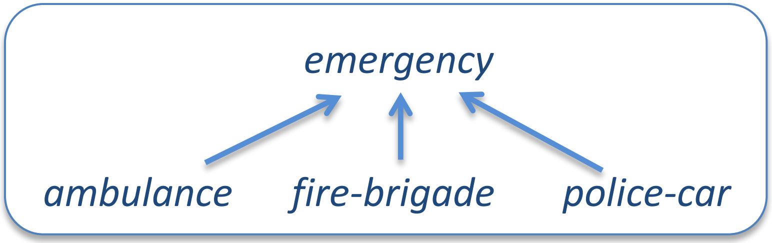
Norm generalisation



Conservative approach

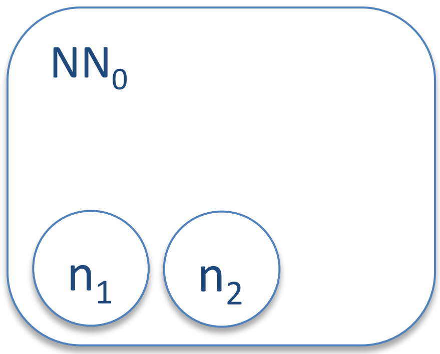
Employs **full evidence**

to generalise norms.



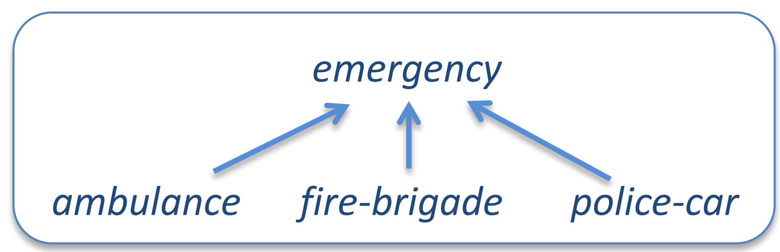
SIMON: Optimistic norm generalisation

Optimistic norm generalisation



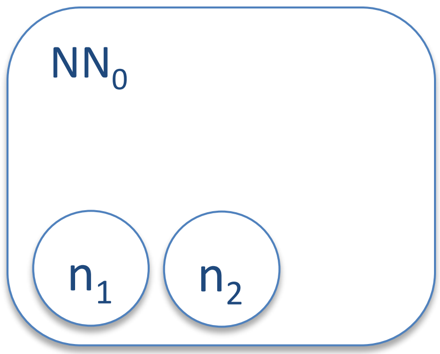
Normative system
$\{n_1, n_2\}$

n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**



SIMON: Optimistic norm generalisation

Optimistic norm generalisation

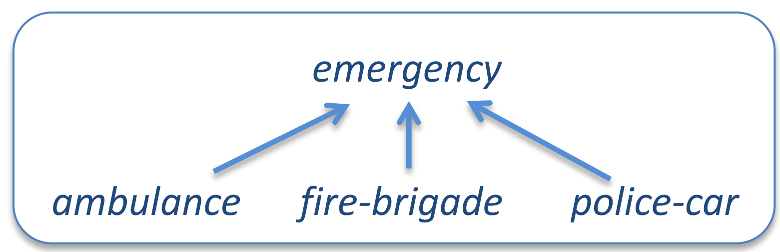


Normative system
$\{n_1, n_2\}$

Most speciffic generalisation between two terms

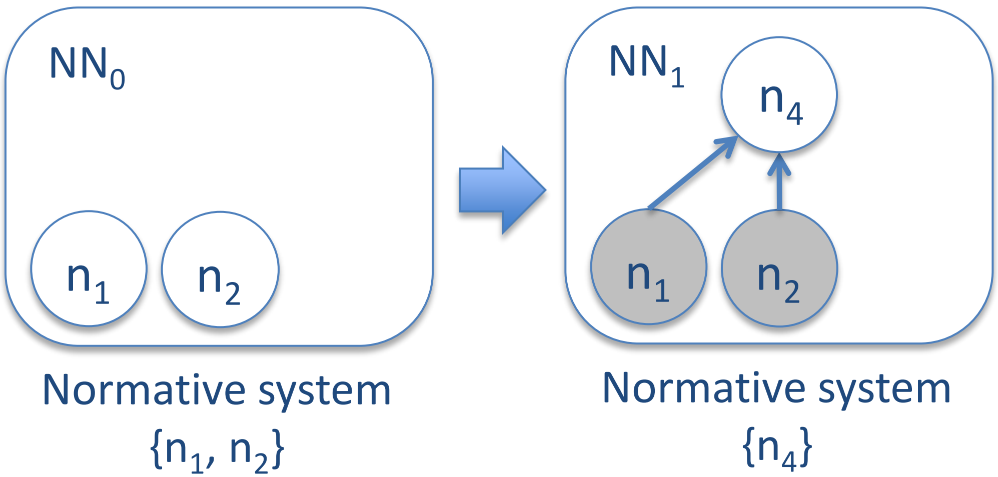
E. Armengol and E. Plaza.
Bottom-up induction of feature terms.
Machine Learning, 41(3):259–294, 2000.

n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**

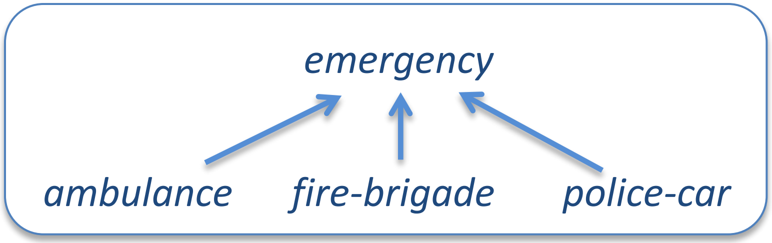


SIMON: Optimistic norm generalisation

Optimistic norm generalisation



n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**
n_4: Give way to **emergency** vehicles



SIMON: Optimistic vs. conservative generalisation

1. Conservative generalisation requires **full evidence**.

2. Optimistic generalisation just requires **partial evidence**.

3. Optimistic generalisation expected to **increase** the number of generalisations.

$\Downarrow$

More **compact** normative systems

(lower **minimality** and **simplicity**)

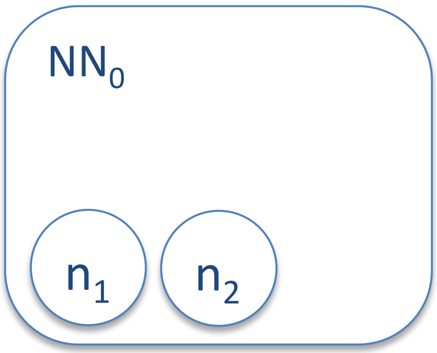
SIMON: Norm refinement

Based on three key components:

1. A **taxonomy** of terms to specify norms
2. An **optimistic** approach to norm generalisation
3. Novel norm generalisation **modes**

SIMON: Shallow norm generalisation

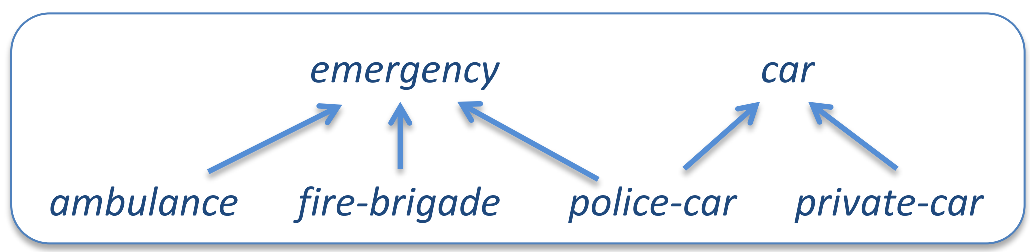
Shallow norm generalisation (S-SIMON)



Normative system
$\{n_1, n_2\}$

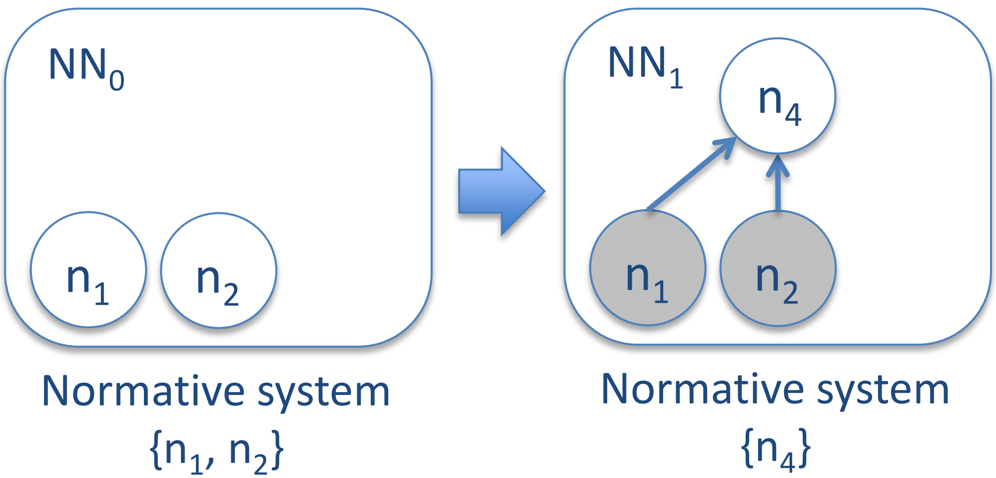
- Compares norms that are **active** in the normative network.

n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**



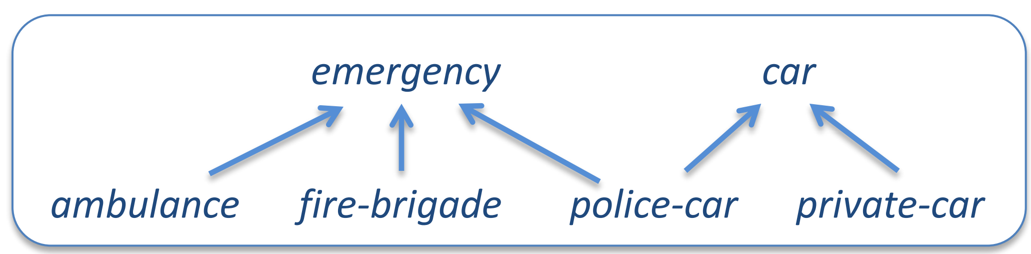
SIMON: Shallow norm generalisation

Shallow norm generalisation (S-SIMON)



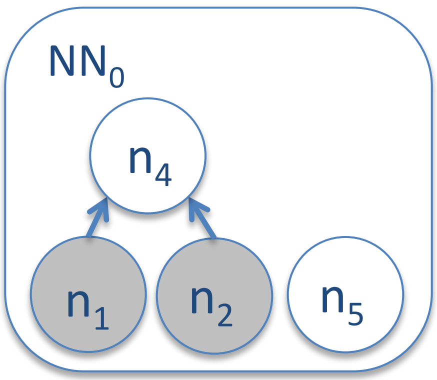
- Compares norms that are **active** in the normative network.
- **Directly** generalises two active norms.

n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**
n_4: Give way to **emergency** vehicles



SIMON: Deep norm generalisation

Deep norm generalisation (D-SIMON)

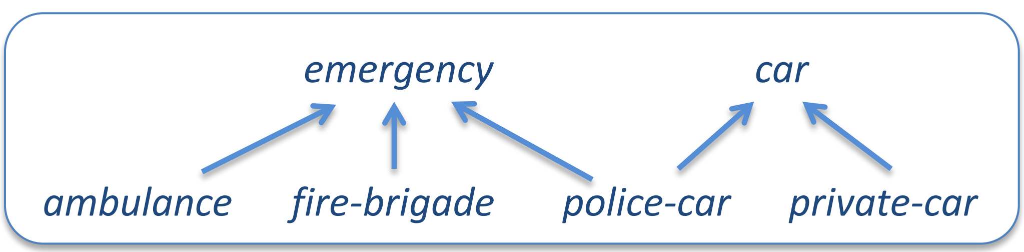


Normative system
$\{n_4, n_5\}$

- Compares norms that are **active** in the normative network.

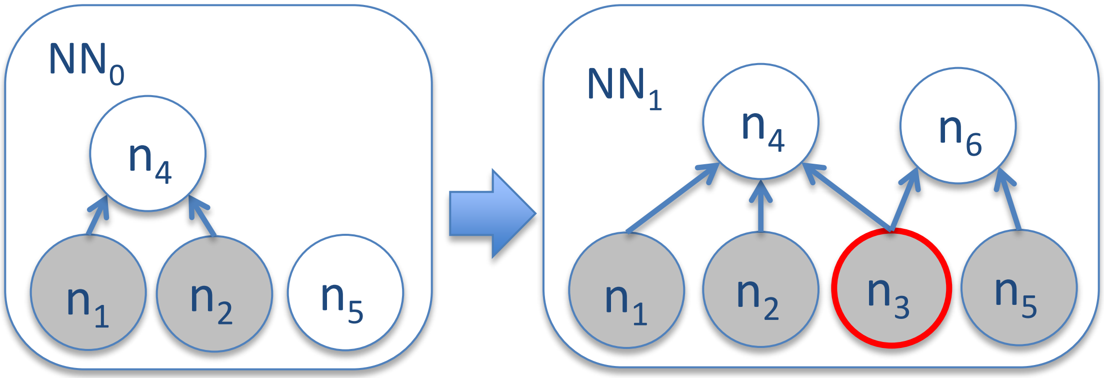
$\mathbf{n_1}$: Give way to **ambulances**
$\mathbf{n_2}$: Give way to **fire brigade**

$\mathbf{n_4}$: Give way to **emergency** vehicles
$\mathbf{n_5}$: Give way to **private cars**



SIMON: Deep norm generalisation

Deep norm generalisation (D-SIMON)

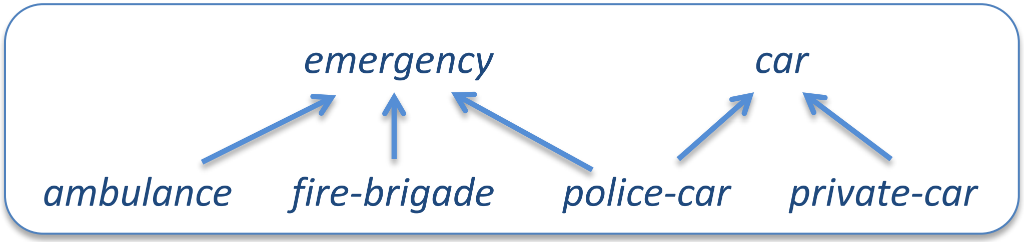


Normative system $\{n_4, n_5\}$

Normative system $\{n_4, n_6\}$

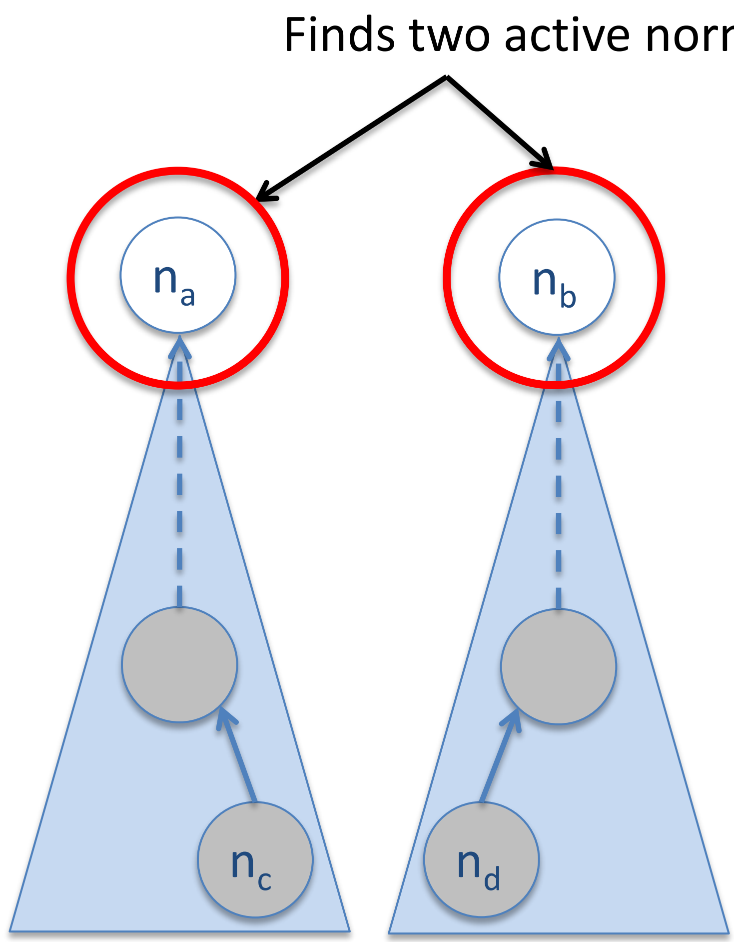
- Compares norms that are **active** in the normative network.
- **Indirectly** generalises two norms that are subsumed by two active norms.

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**



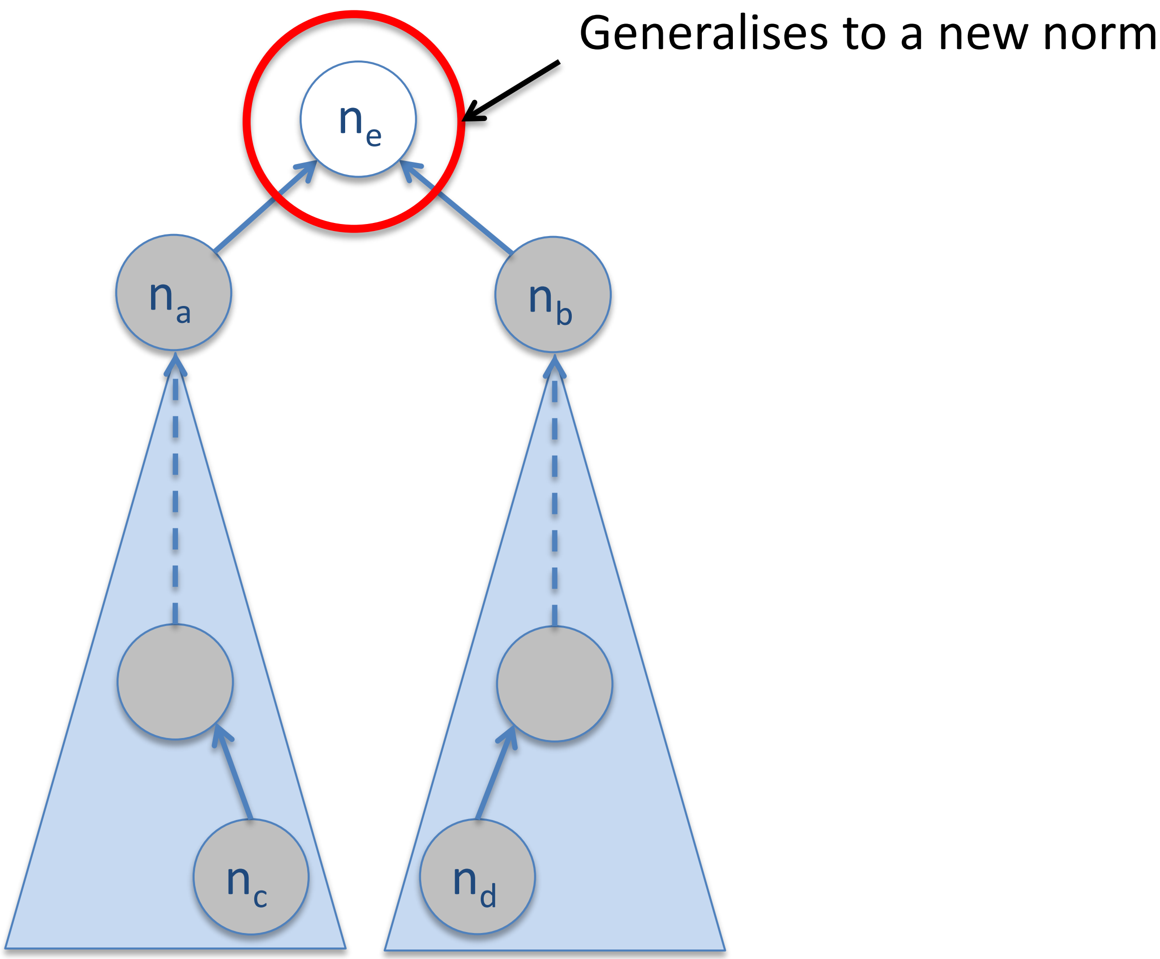
SIMON: Shallow vs Deep norm generalisation

Shallow generalisation



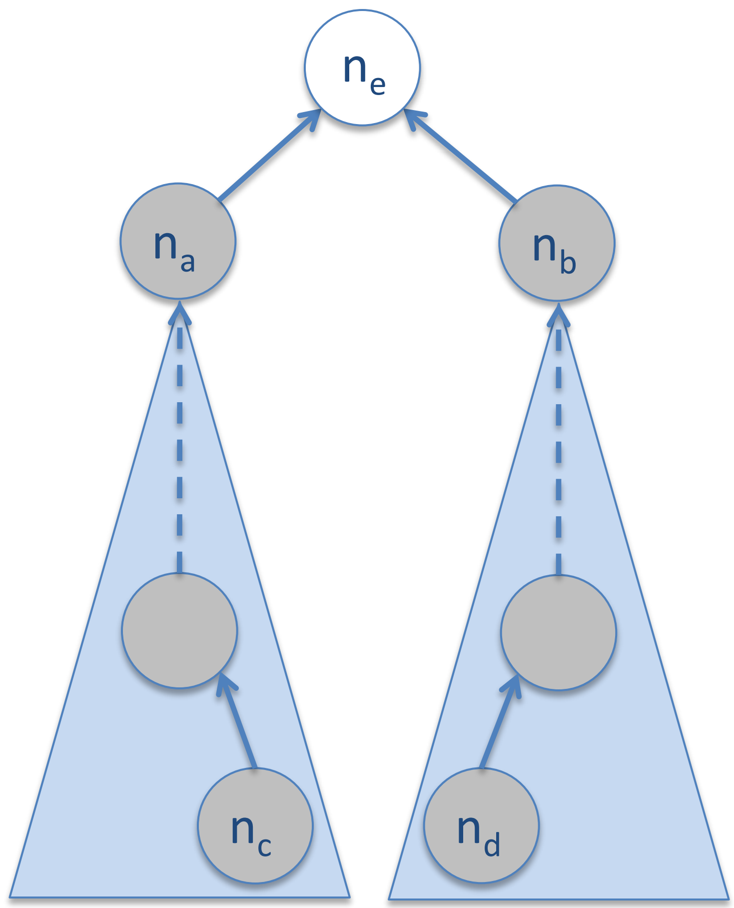
SIMON: Shallow vs Deep norm generalisation

Shallow generalisation

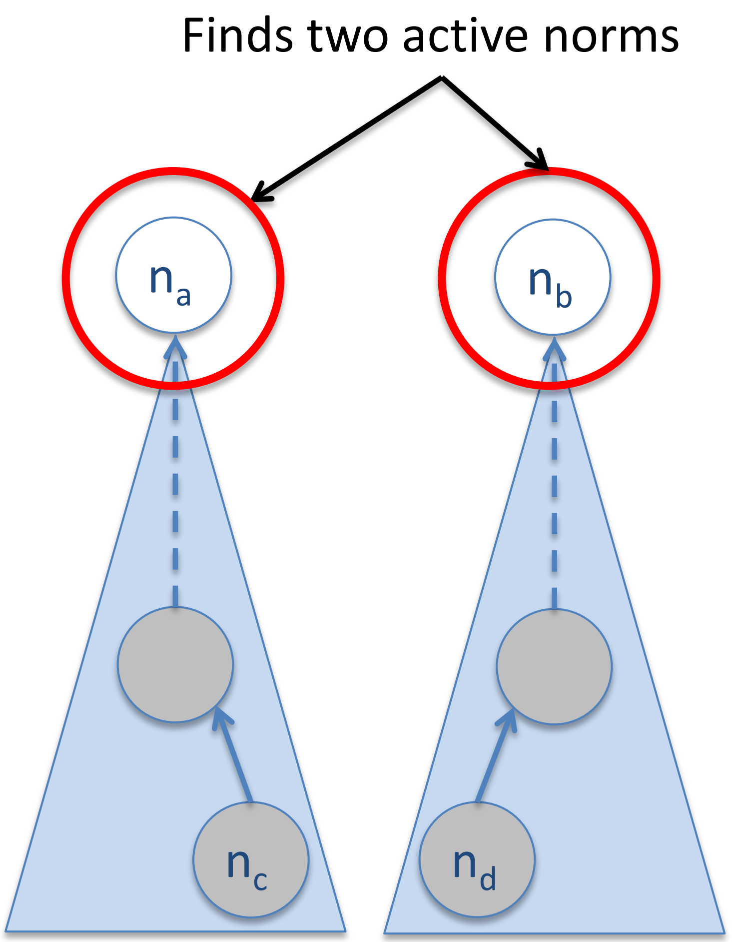


SIMON: Shallow vs Deep norm generalisation

Shallow generalisation

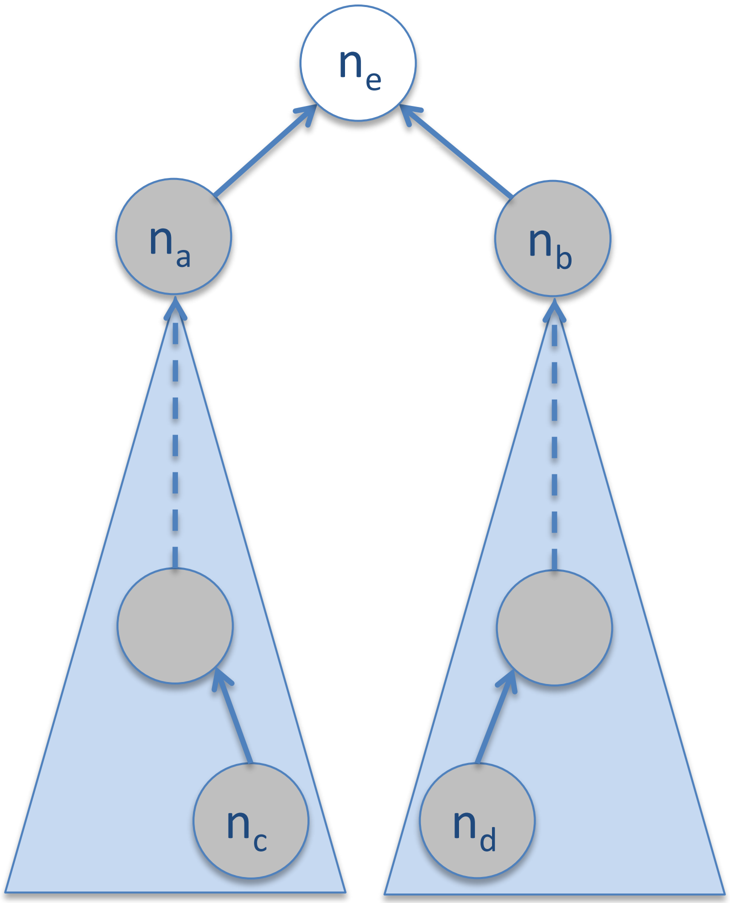


Deep generalisation



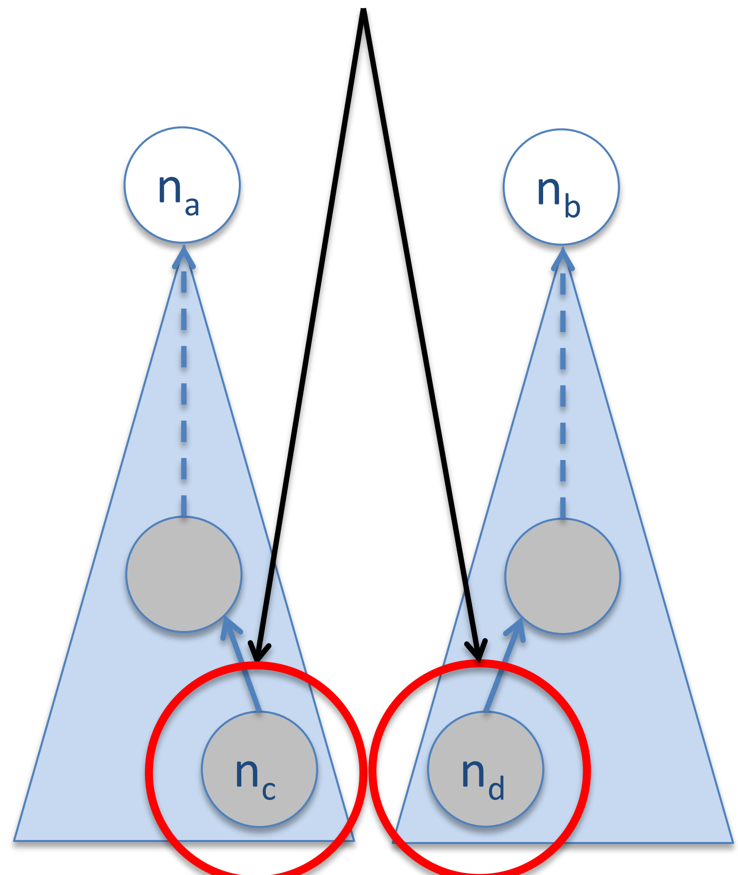
SIMON: Shallow vs Deep norm generalisation

Shallow generalisation



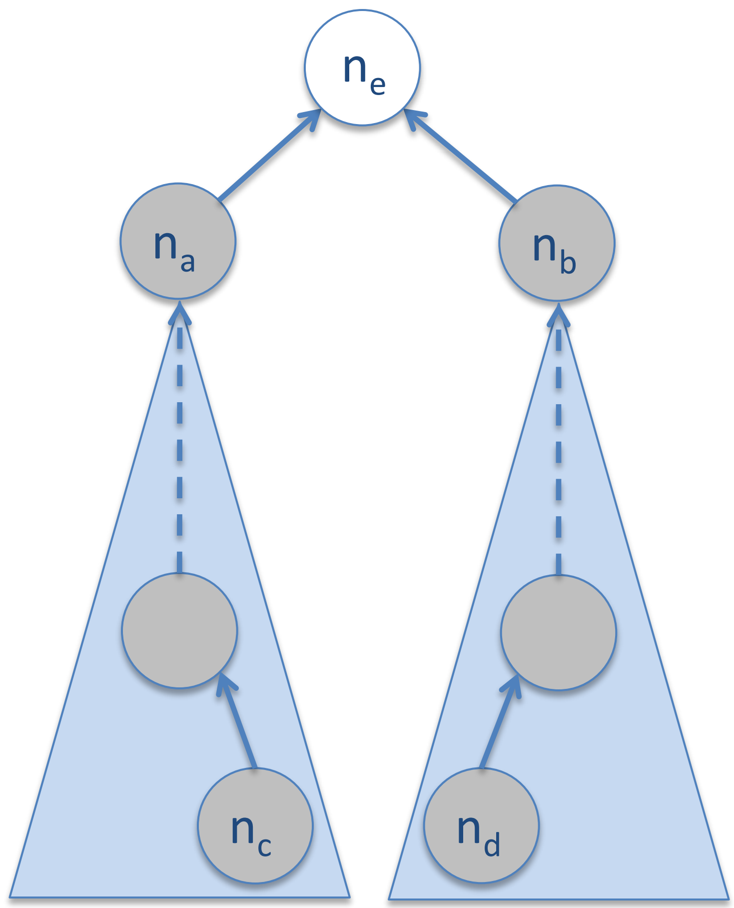
Deep generalisation

Finds two inactive norms to generalise

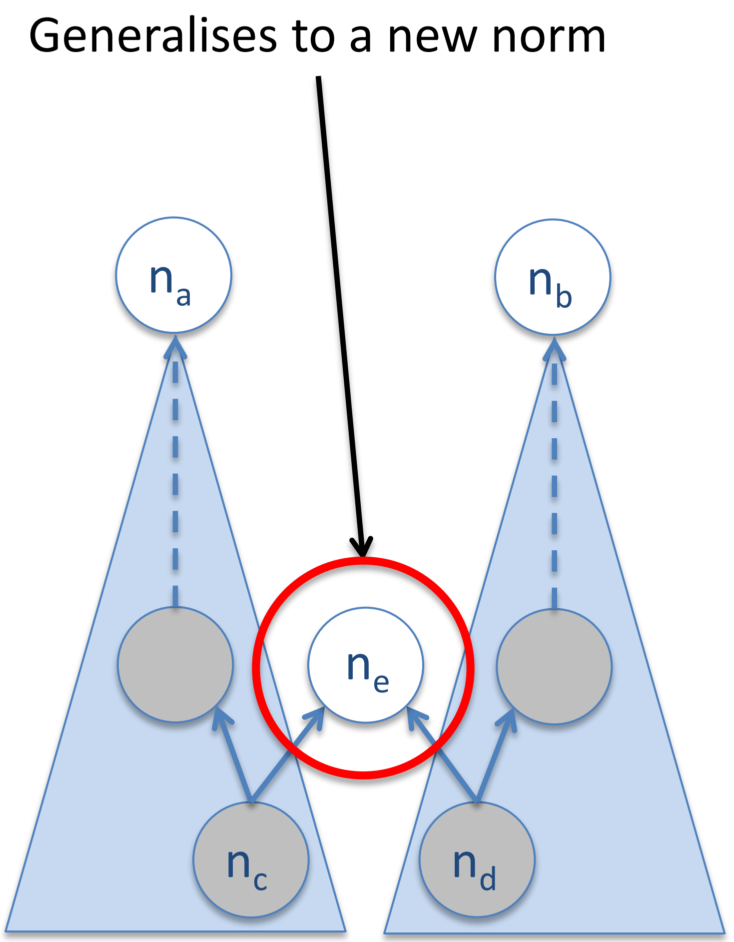


SIMON: Shallow vs Deep norm generalisation

Shallow generalisation

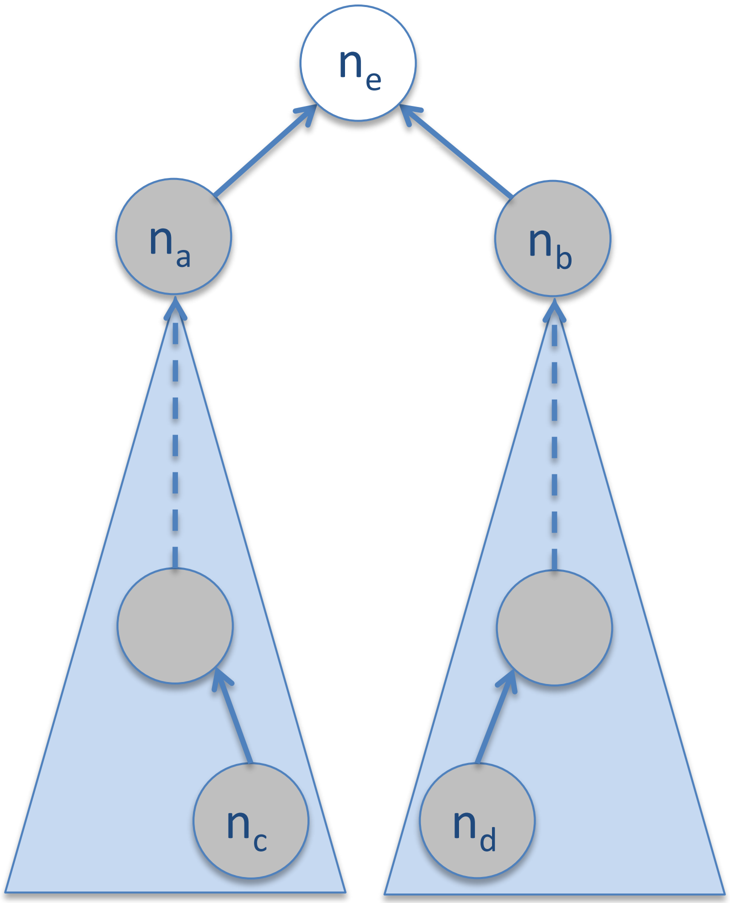


Deep generalisation



SIMON: Shallow vs Deep norm generalisation

Shallow generalisation | **Deep** generalisation



More **coarse** | More **fine-grained**

Empirical evaluation

Our goal is to **compare** **IRON** (AAMAS 2013) and **SIMON** in terms of:

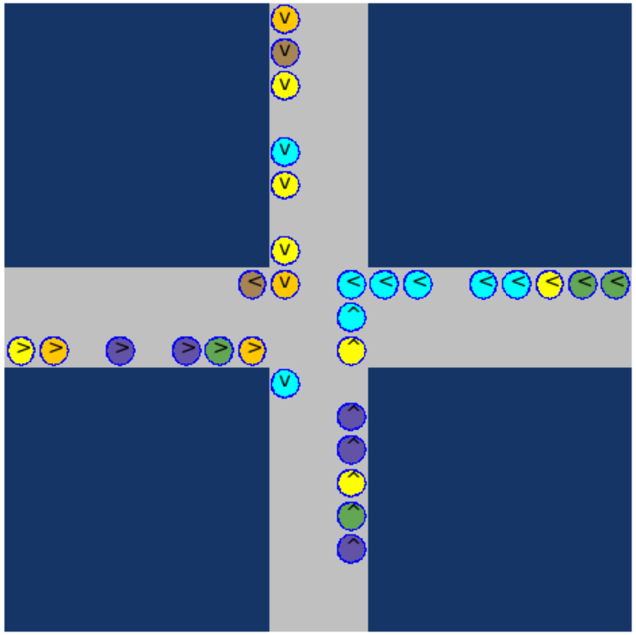
1. The **quality** of the normative systems that they synthesise.
2. The computational **costs** their synthesis processes require.
3. The **search space** of normative systems that they explore.

IRON performs **conservative** norm generalisations whereas **SIMON** performs **optimistic** norm generalisations.

Empirical evaluation: Scenario

We employ the same **simulated** **traffic junction**:

- **Agents** are cars.
- **Conflicts** are collisions among cars.
- **Our goal** is to synthesise normative systems that avoid collisions between cars.



Simulated traffic intersection scenario

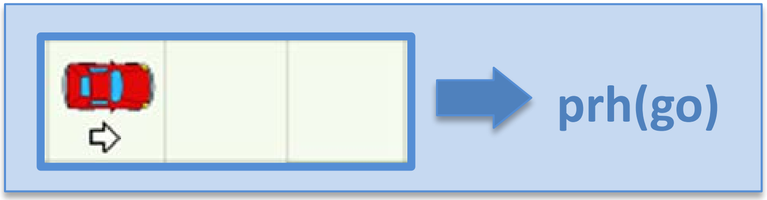
Empirical evaluation: Norms

Norms

- **IF** … **THEN**… rules.
- Norm precondition: Set of **predicates** with one **term** each.
- Norm postcondition: A **modality**.

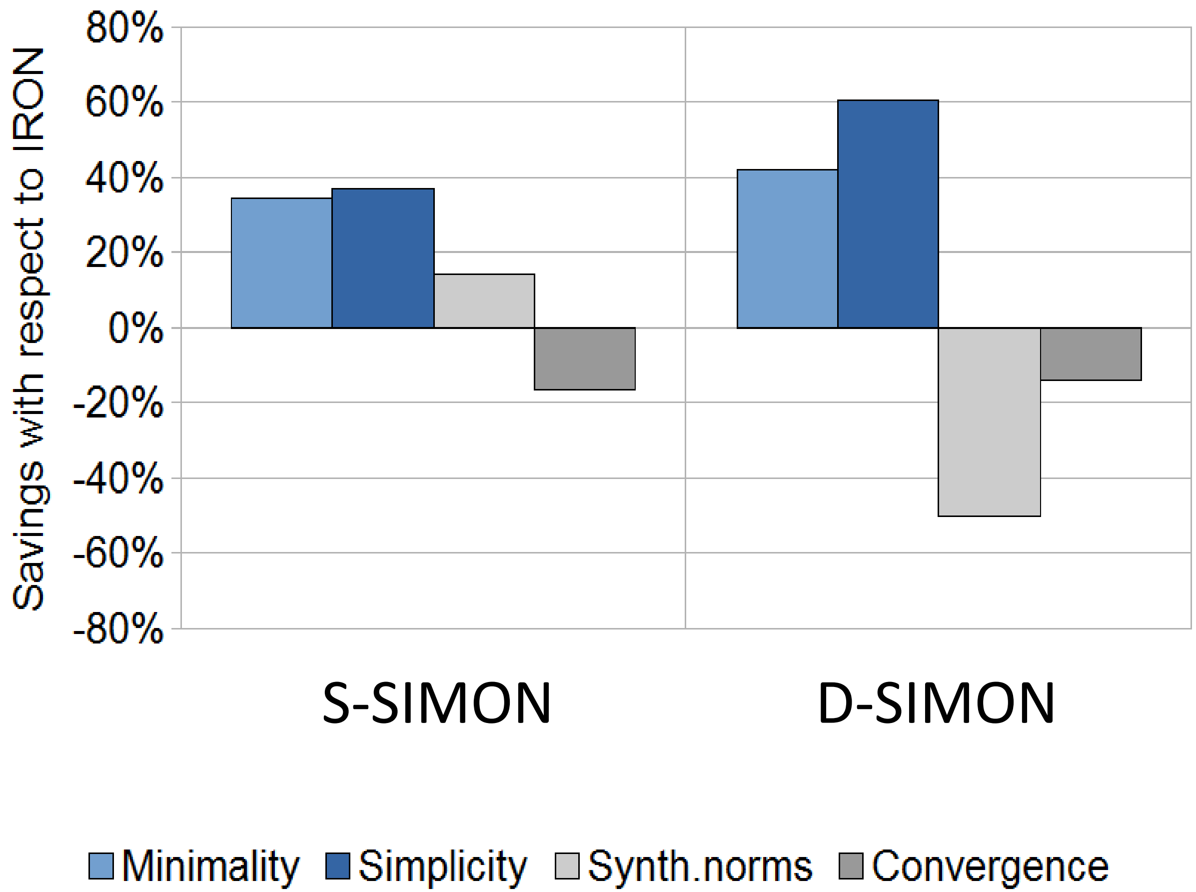
Norm example

Graphical representation



IF left(*car-heading-right*) **& front**(*nothing*) **& right**(*nothing*) **THEN prohibition**(*go*)

Empirical results



BENEFITS

D-SIMON normative systems are up to 46% more minimal and 61% simpler than IRON's.

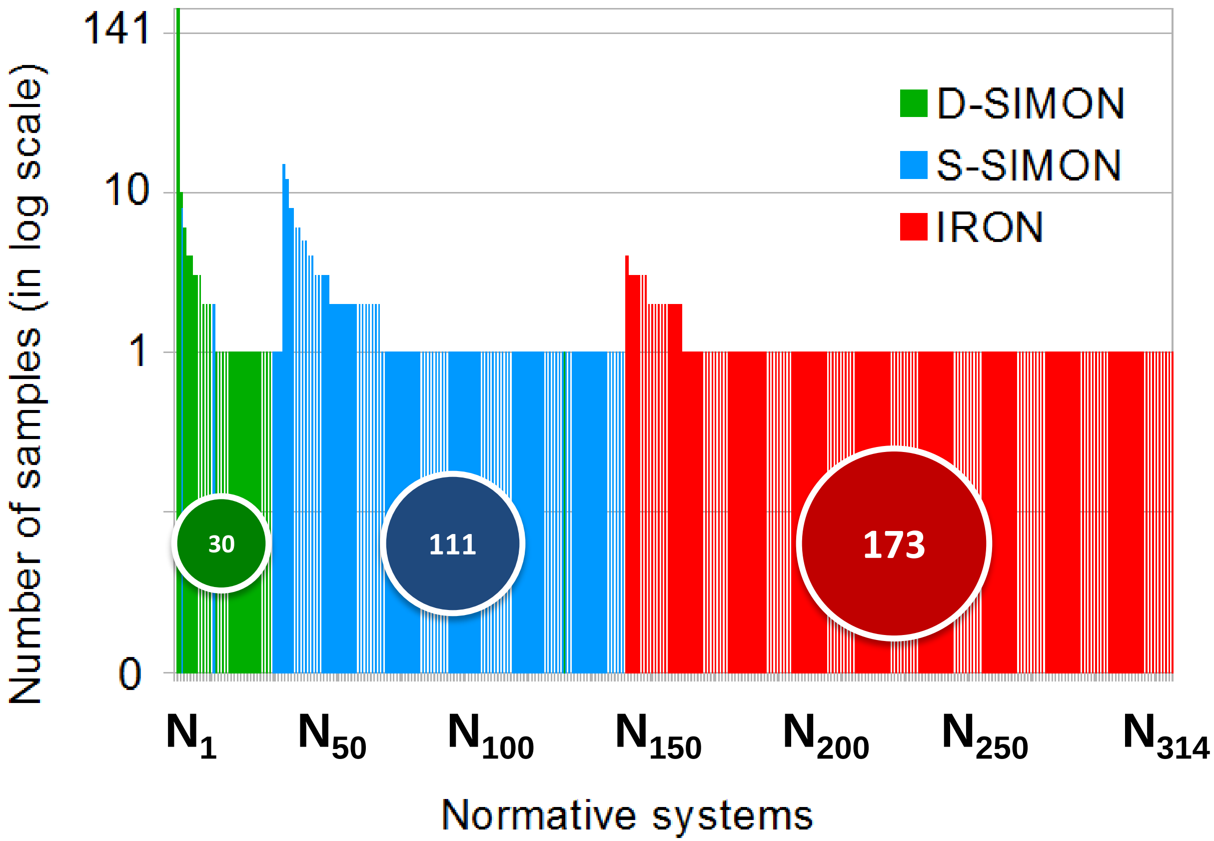
COSTS

D-SIMON requires:

1. To synthesise more norms.
2. Extra convergence time.

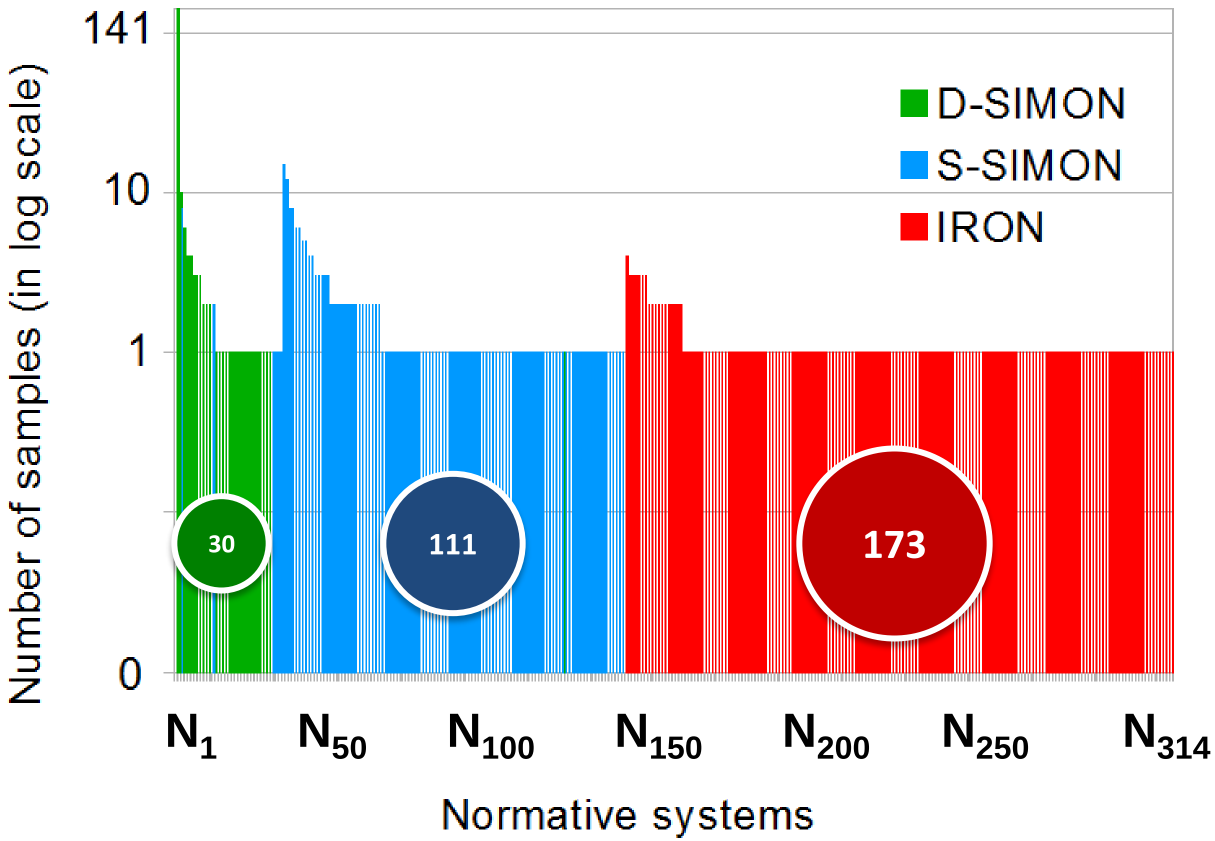
Empirical results

Why does **D-SIMON** outperform **IRON** in terms of **minimality** and **simplicity**?



Empirical results

Why does **D-SIMON** outperform **IRON** in terms of **minimality** and **simplicity**?



D-SIMON focuses on an area of the search space with more **compact** normative systems

Conclusions

1. Synthesising **conflict-free** and **compact** normative systems is important to:

   - Avoid **conflicts**.
   - Avoid **over regulation**.
   - **Ease** the reasoning of agents.

2. We have presented **SIMON**, a novel strategy for the on-line synthesis of conflict-free and compact normative systems.

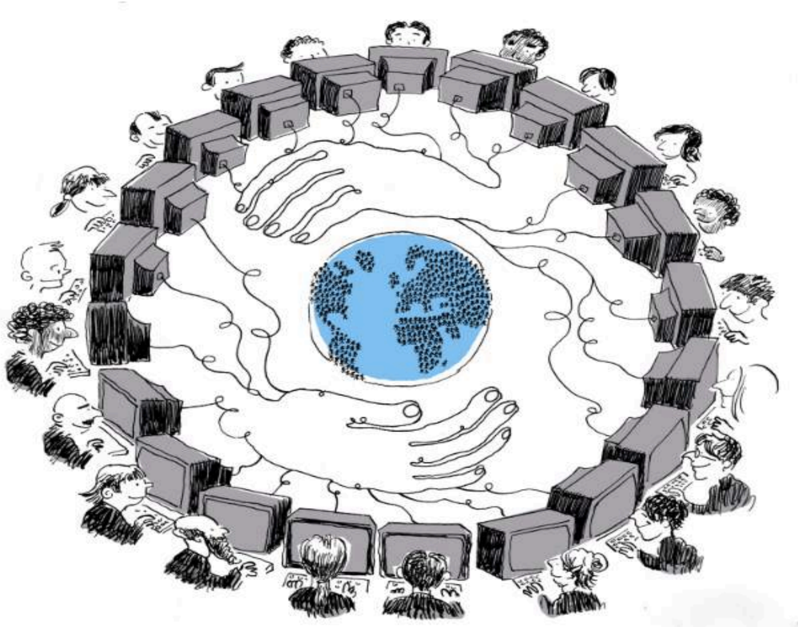
3. **SIMON** shows that being **optimistic** (non requiring full evidence) and investing computational efforts on discovering **implicit relationships** (deep generalisation) pays off.

4. Applicable to other domains.

Case study 2: Virtual Communities

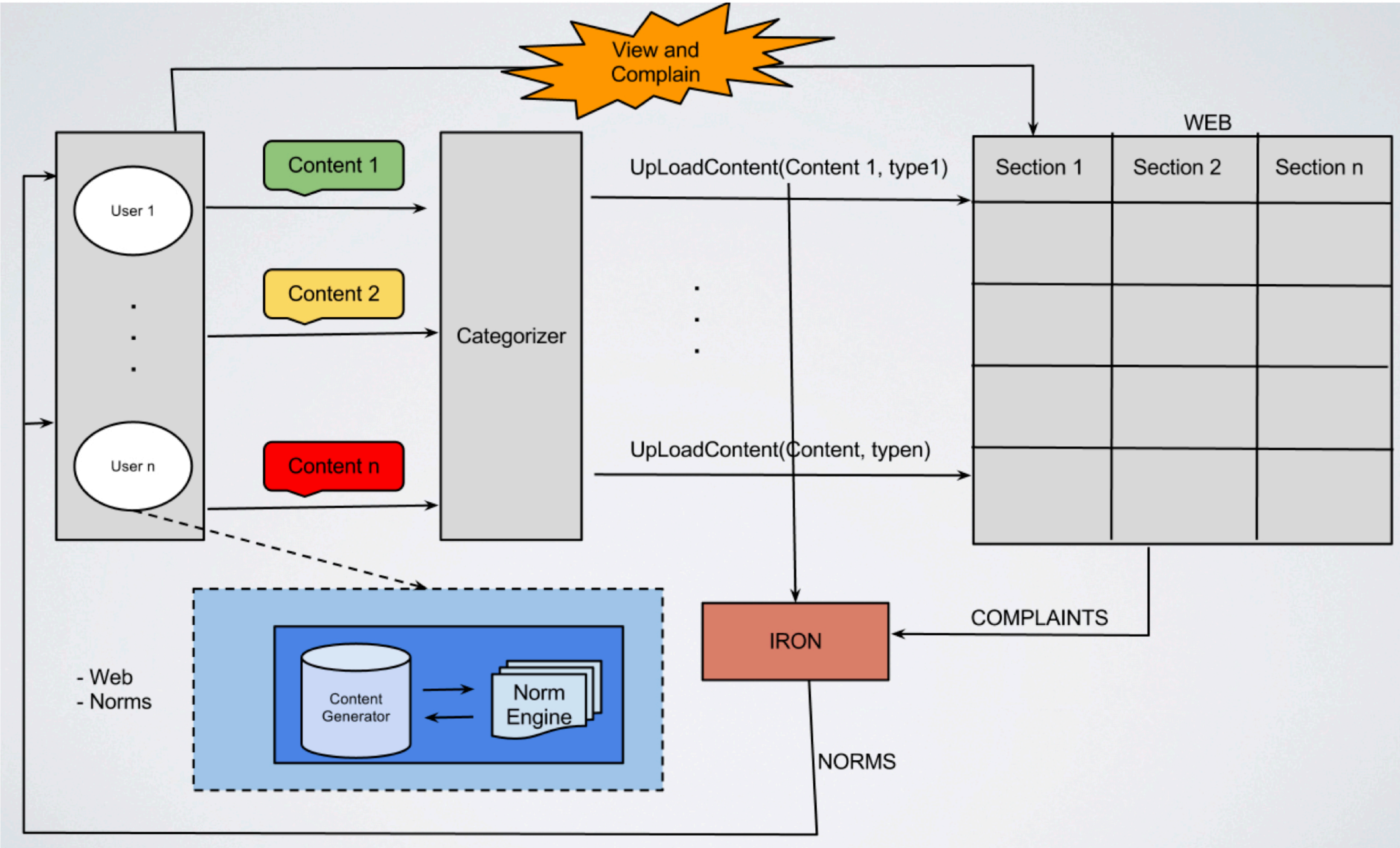
- Agents model human users interacting within virtual communities
- On-line synthesis of norms to avoid conflicts (i.e. user complaints)

Norms are like… **IF** user(1) & section(2) & contentType(porn)
THEN prh(upload(content))

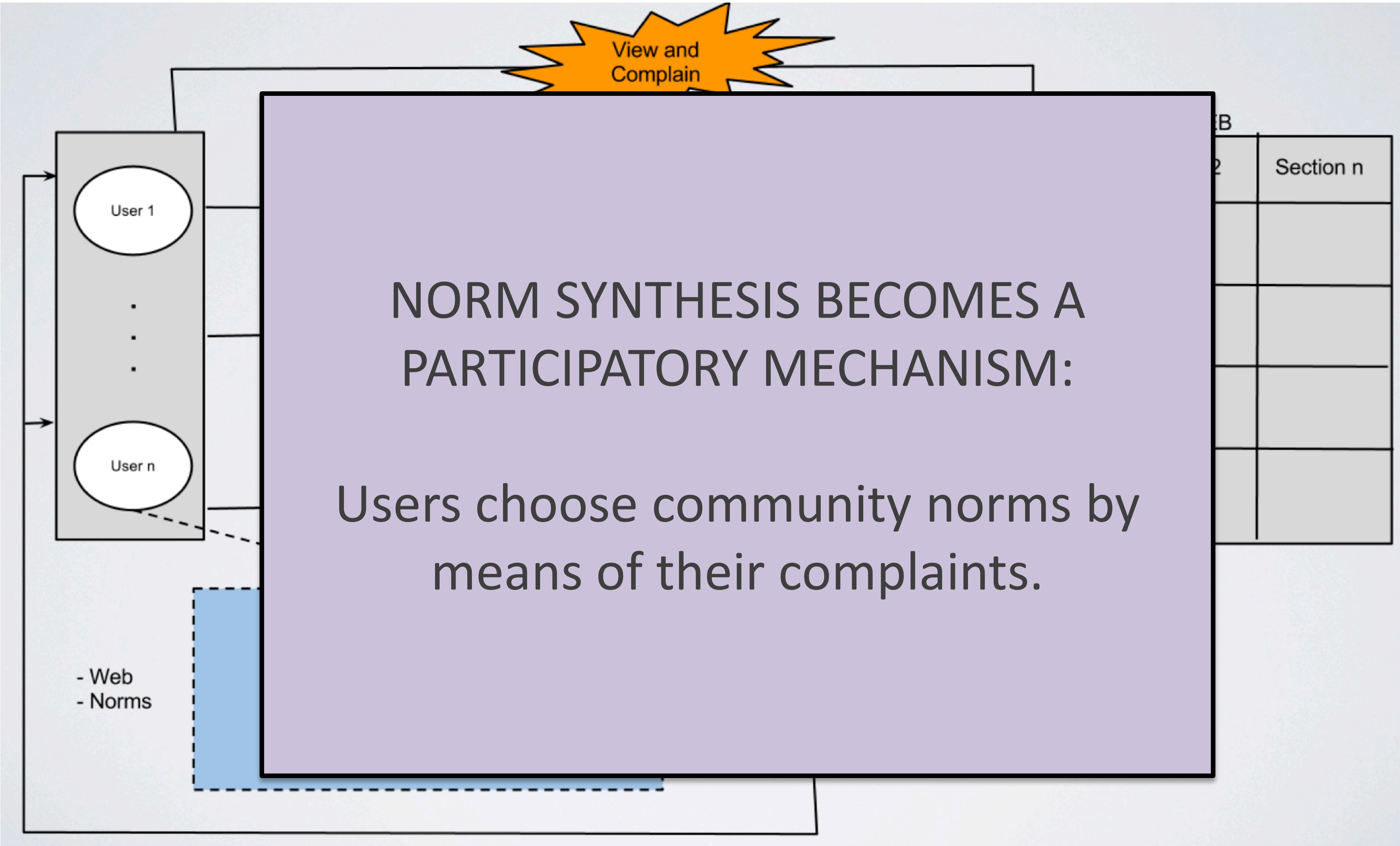


MAS = Simulated virtual community

Case Study 2: Virtual Communities Simulator



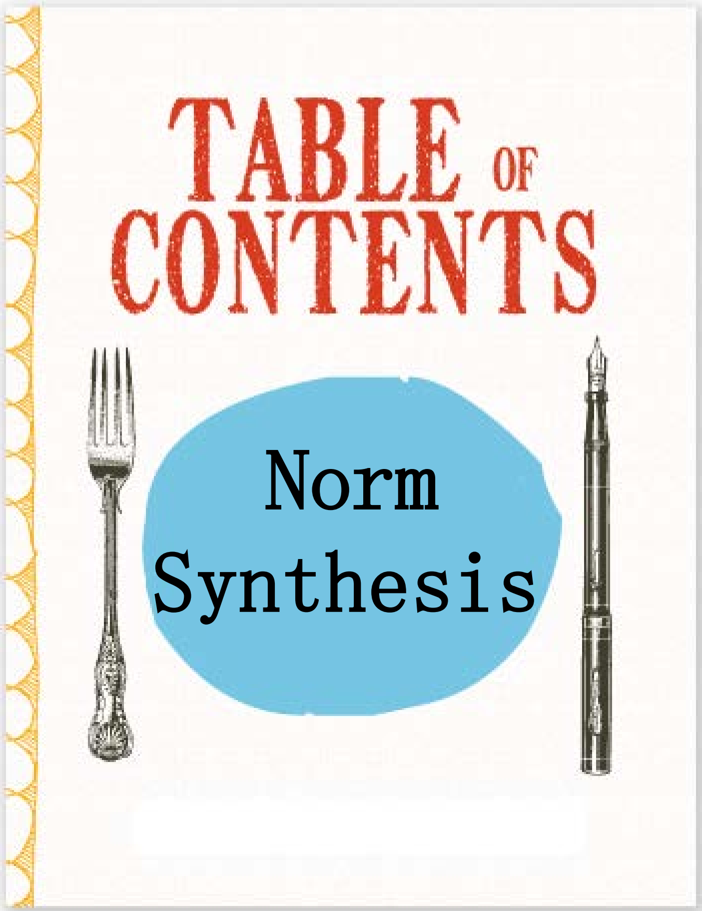
Case study 2: Virtual Communities



NORM SYNTHESIS BECOMES A PARTICIPATORY MECHANISM:

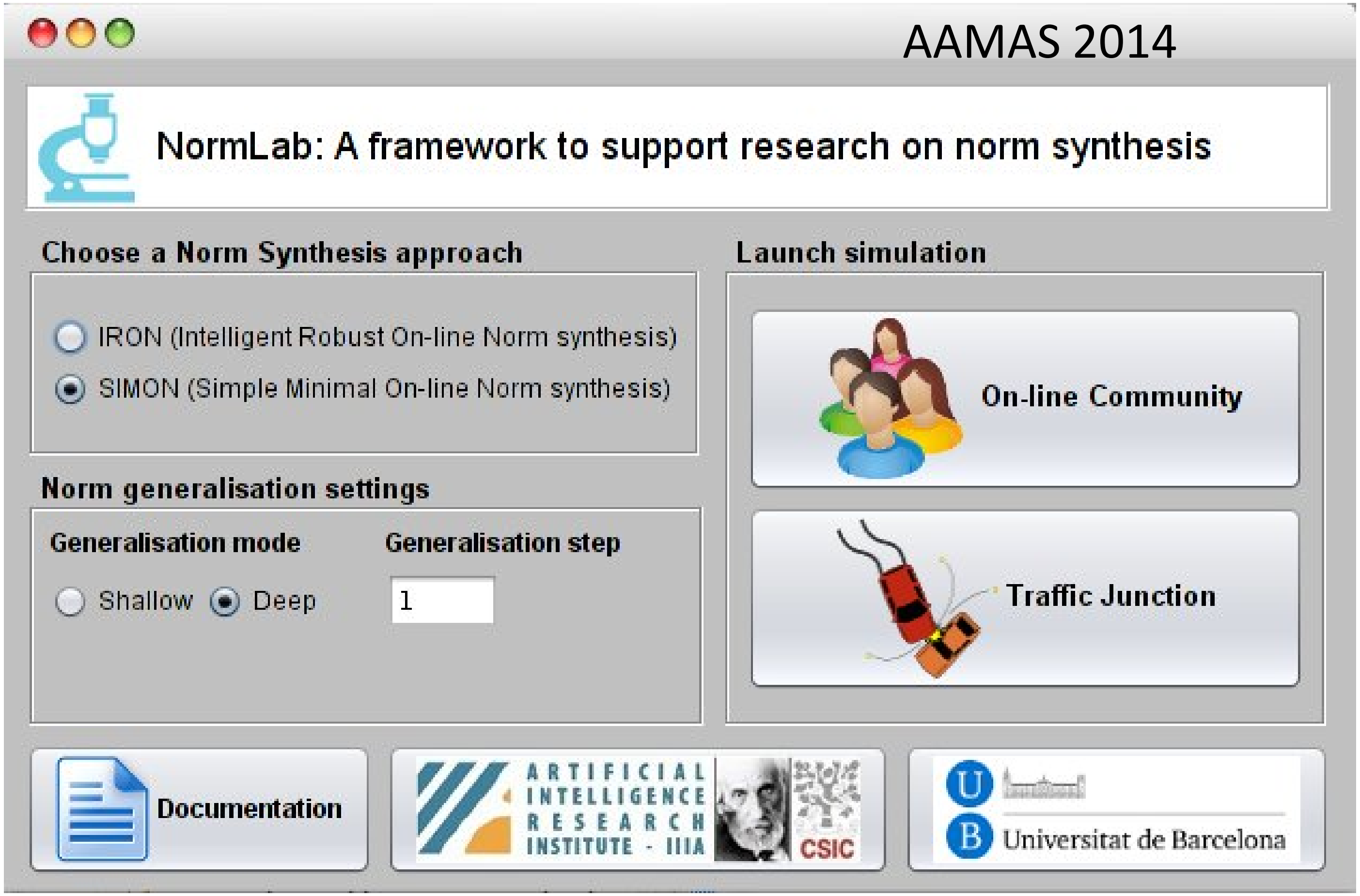
Users choose community norms by means of their complaints.

Tutorial Outline



1. Introduction to Norms and Normative MAS.
2. Overview of approaches to norm synthesis.
3. On-line automatic norm synthesis.
4. **Demo and hands-on activity**

Demo



Javier Morales, Maite López-Sánchez, Juan A. Rodríguez-Aguilar, Michael Wooldridge, Wamberto Vasconcelos

ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE - IIIA CSIC
UNIVERSITAT DE BARCELONA
UNIVERSITY OF OXFORD
UNIVERSITY OF ABERDEEN

1. What is NormLab?

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

NormLab allows to:

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

2. **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.

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

Tutorial outline

What are the contents of this tutorial?

1. An **introduction** to NormLab
    1. The NormLab architecture.
    2. The Norm Synthesis Machine.
    3. The NormLab simulators.

2. **Configuration** of the working environment
    1. NormLab download.
    2. NormLab installation.

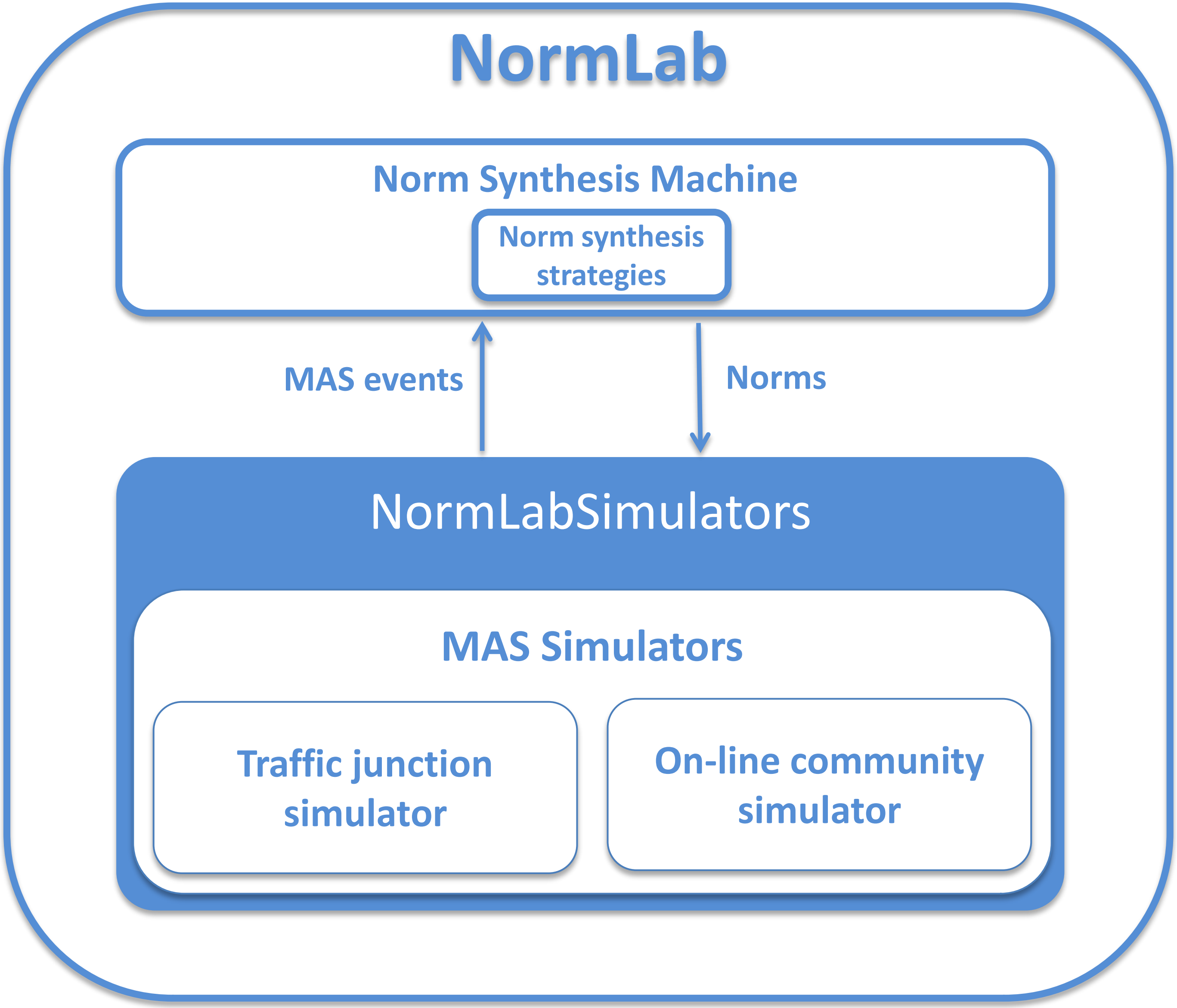
3. NormLab **execution**:
    1. Execution examples.
    2. Guided development of different norm synthesis strategies.

Tutorial outline

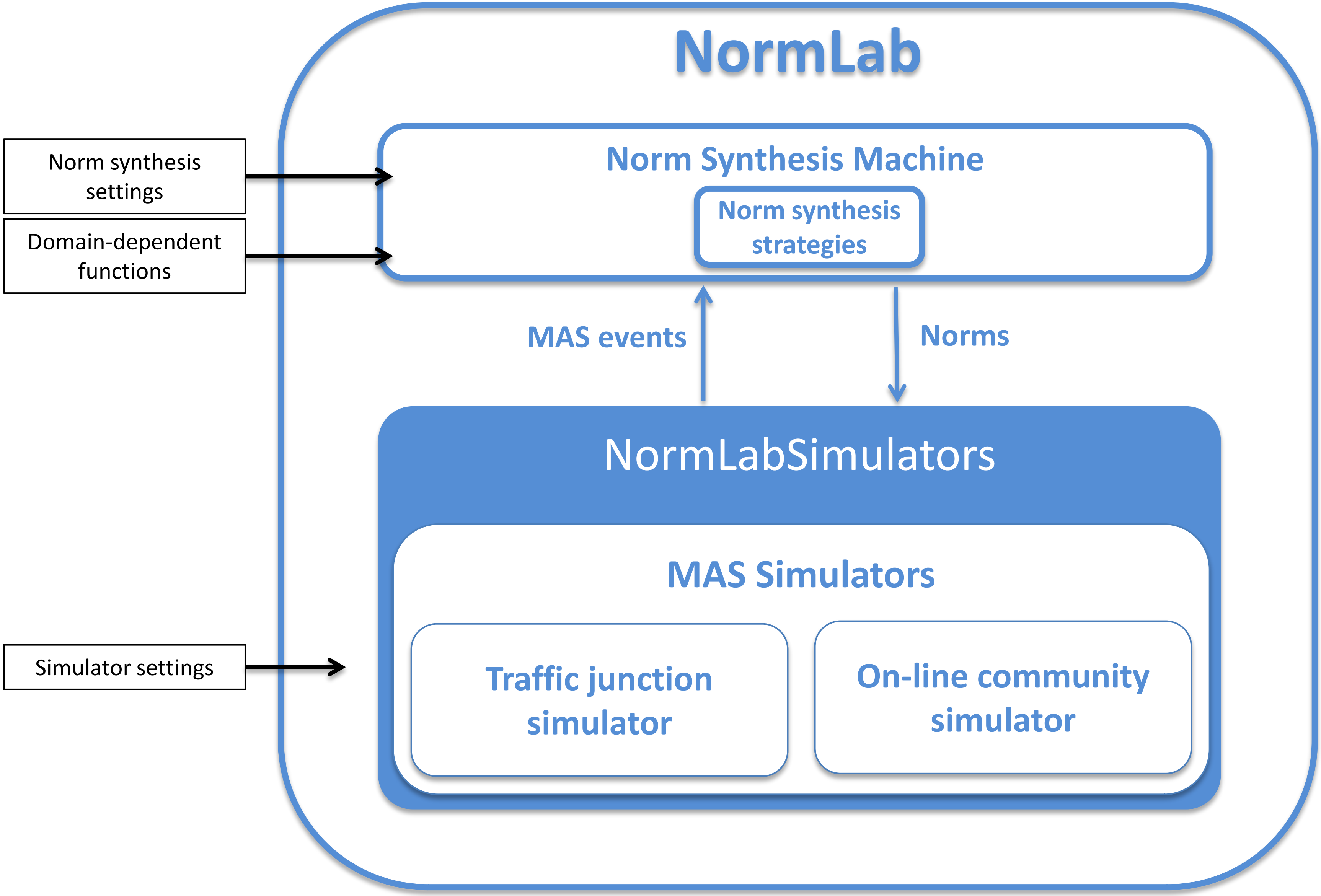
What are the contents of this tutorial?

1. An **introduction** to NormLab
    1. The NormLab architecture.
    2. The Norm Synthesis Machine.
    3. The NormLab simulators.
2. **Configuration** of the working environment
    1. NormLab download.
    2. NormLab installation.
3. NormLab **execution**:
    1. Execution examples.
    2. Guided development of different norm synthesis strategies.

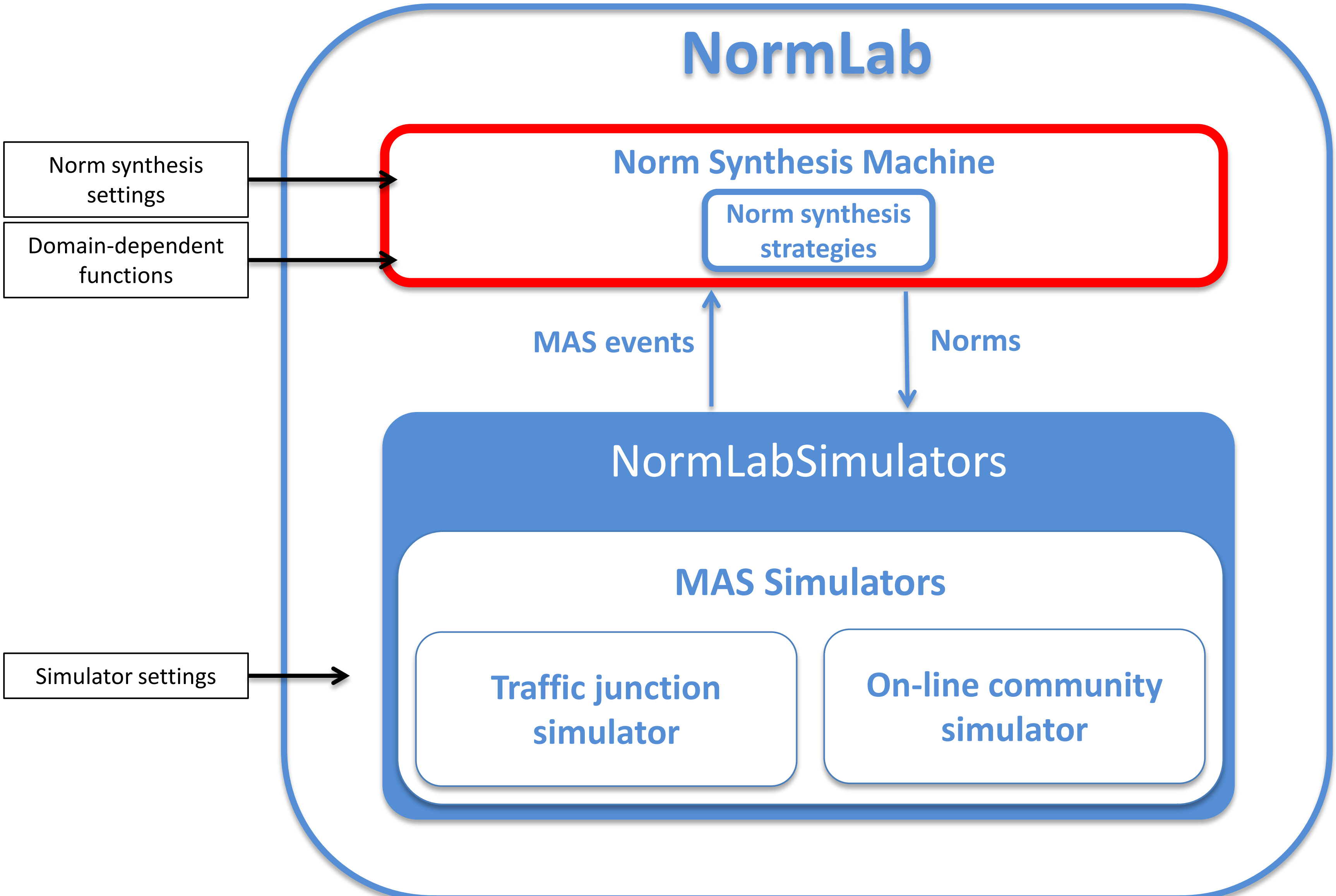
2. NormLab architecture



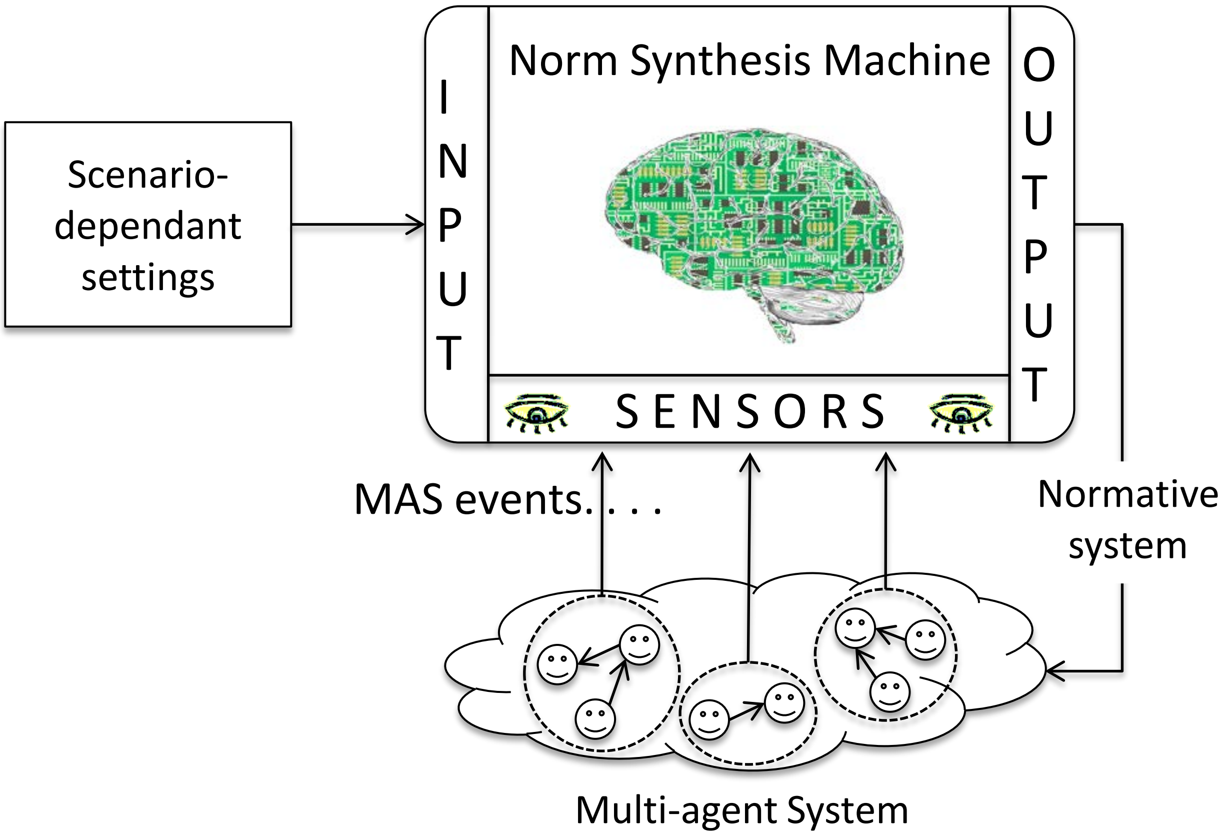
2. NormLab architecture



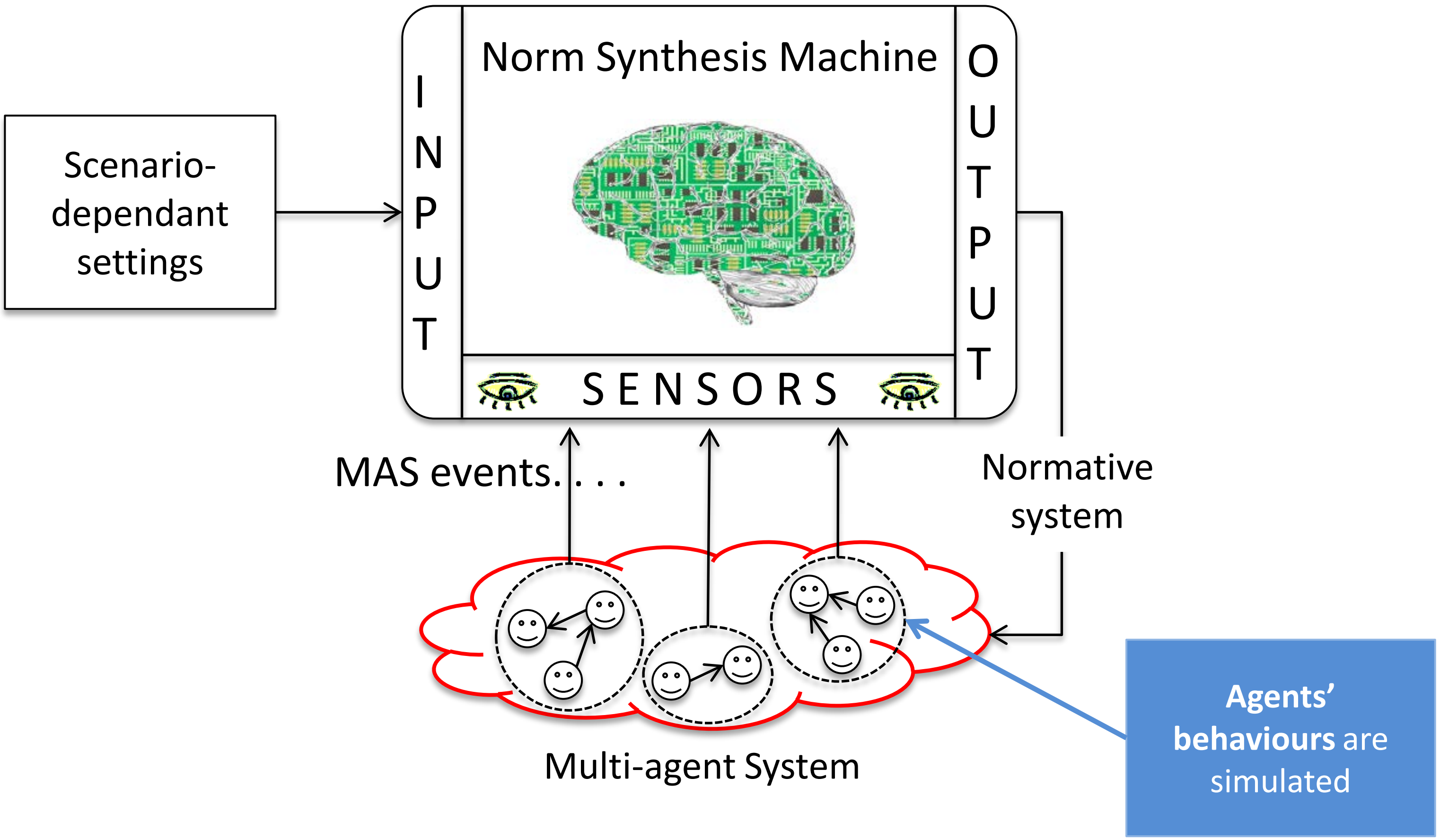
2. NormLab architecture



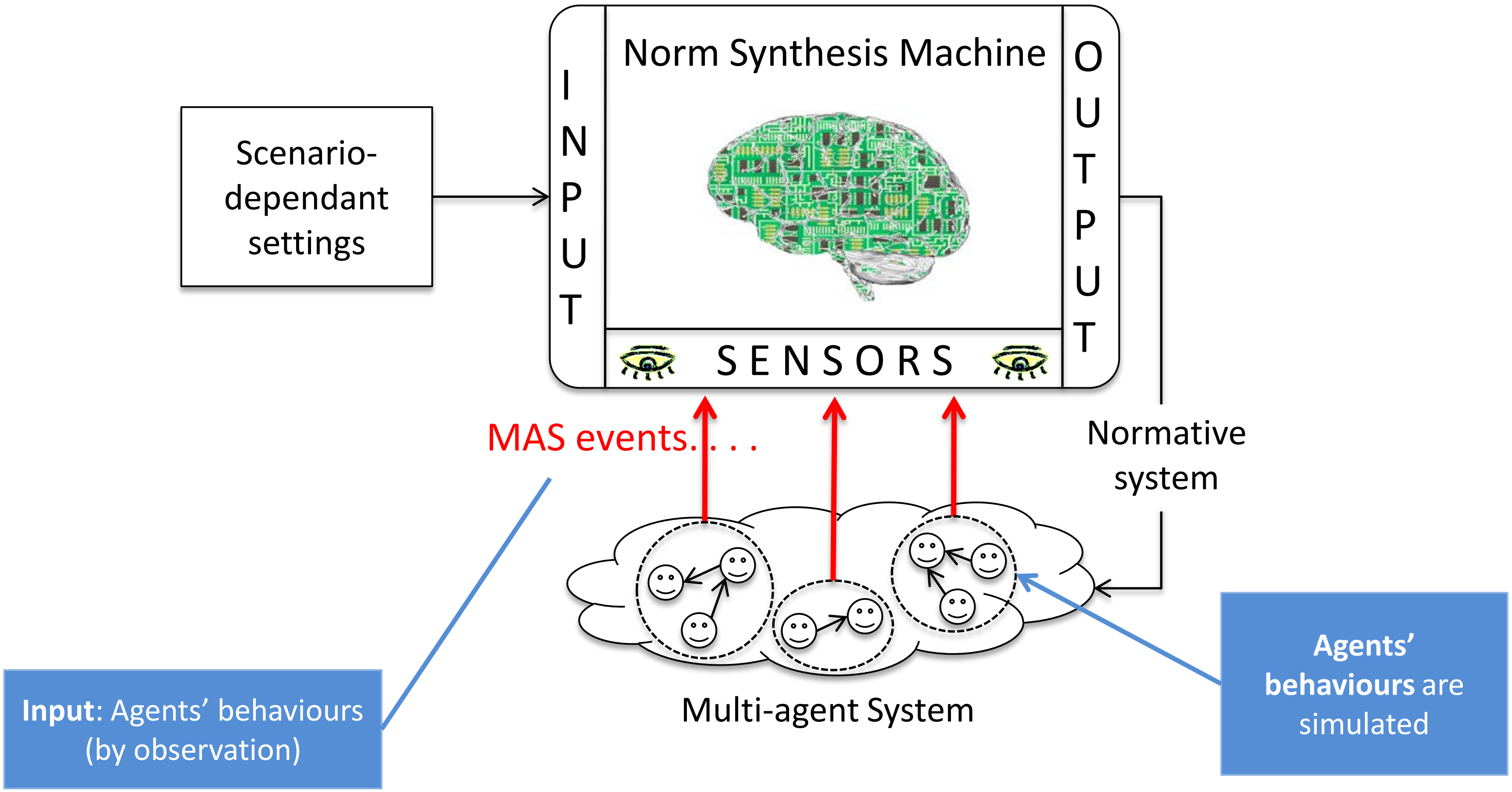
3. The Norm Synthesis Machine



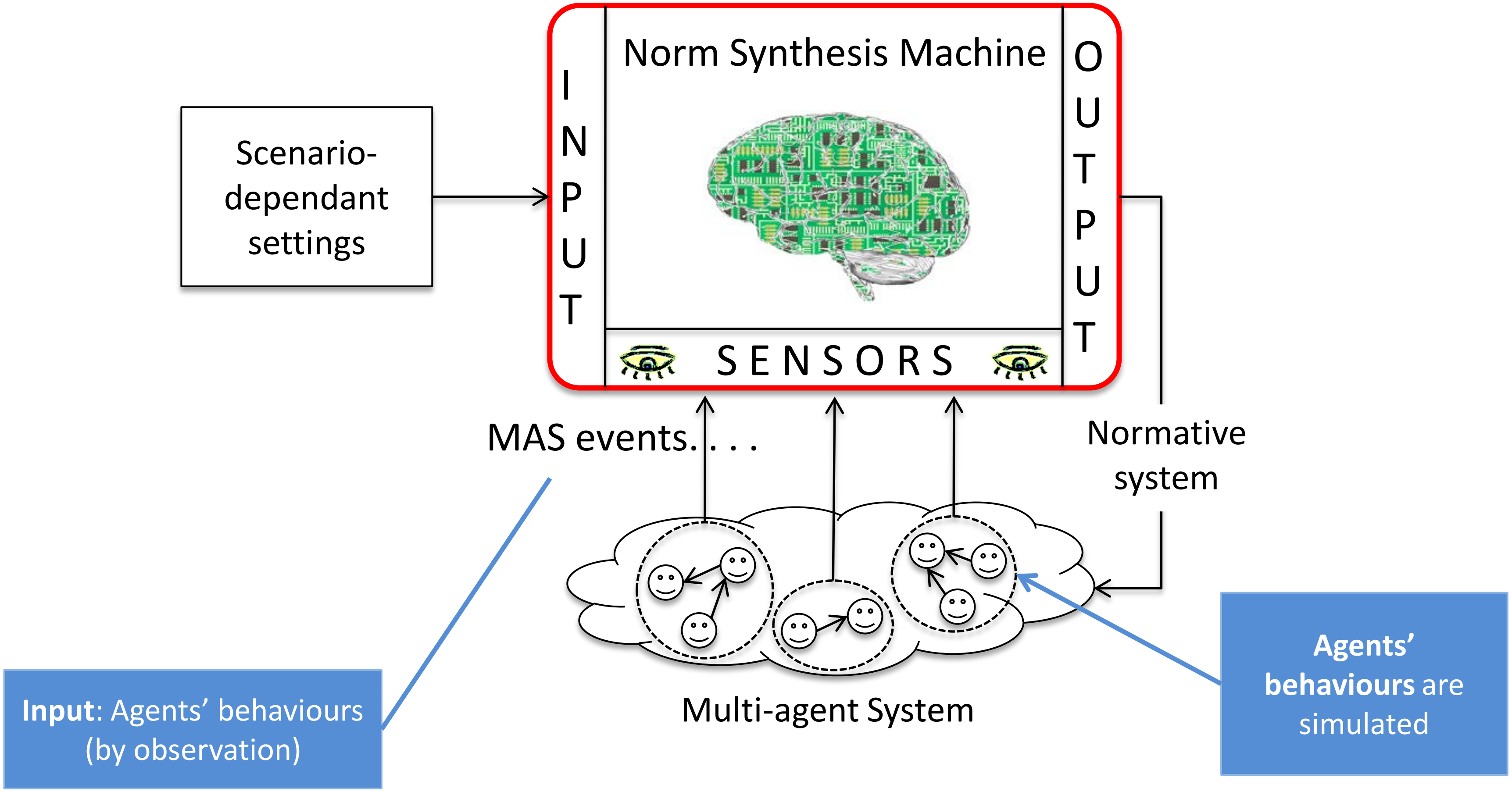
3. The Norm Synthesis Machine



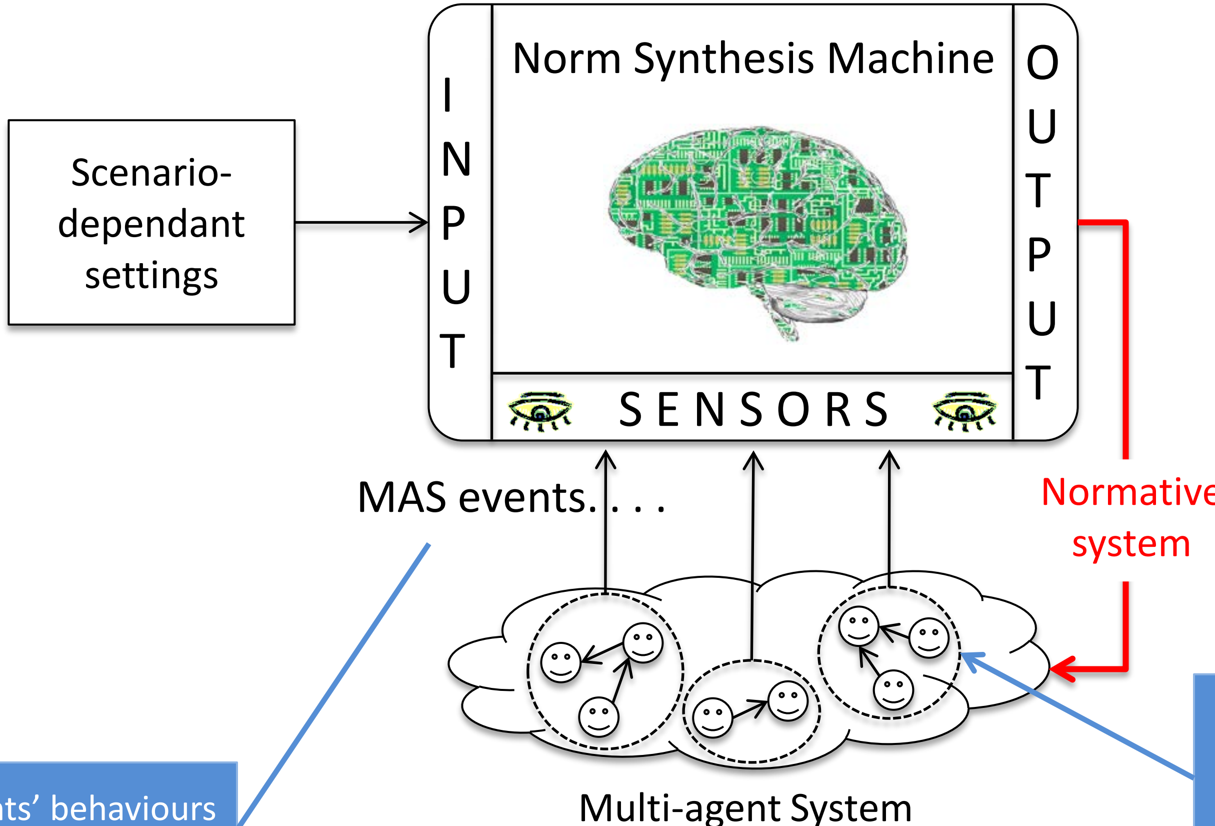
3. The Norm Synthesis Machine



3. The Norm Synthesis Machine



3. The Norm Synthesis Machine

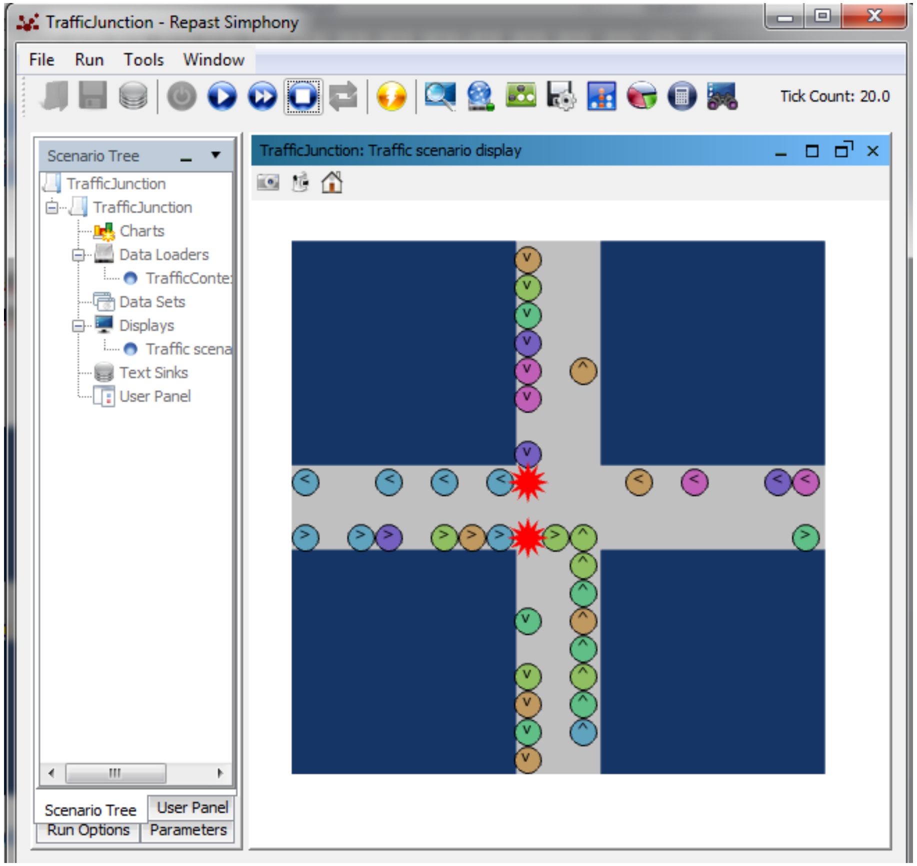


Input: Agents' behaviours (by observation)

Agents' behaviours are simulated

4. The traffic simulator

- **Based on Repast Simphony 2.1**
- **Agents** are cars, and **conflicts** are collisions among cars.
- **The goal** is to synthesise normative systems that **avoid collisions** between cars.



Tutorial outline

What are the contents of this tutorial?

1. An **introduction** to NormLab
    1. The NormLab architecture.
    2. The Norm Synthesis Machine.
    3. The NormLab simulators.
2. **Configuration** of the working environment
    1. NormLab download.
    2. NormLab installation.
3. NormLab **execution**:
    1. Execution examples.
    2. Guided development of different norm synthesis strategies.

5. NormLab download

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

Requirements

- ***Java JDK 1.6*** or greater — http://www.java.com
- ***Eclipse IDE*** (just for Linux users) — http://www.eclipse.org/downloads
- ***Repast Simphony 2.1*** — 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.
- ***NormLab***: **http://normsynthesis.github.io/NormLabSimulators/**
  Contains the code of the two MAS simulators: traffic and on-line community.

Download both projects whether in a **ZIP** or **TAR.GZ** file.

5.1. 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*

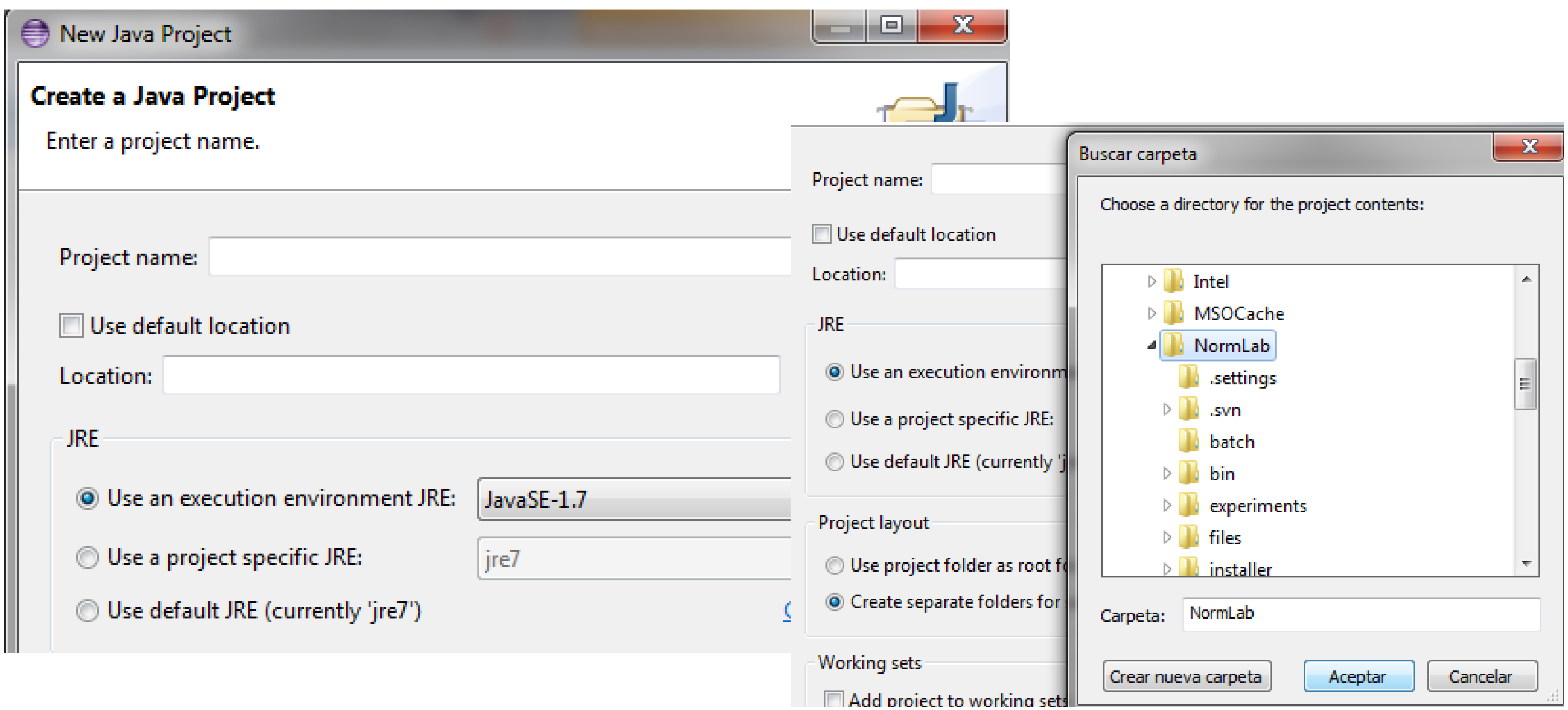
3. Rename both projects,removing the «NormSynthesis» part and the numbers. After renaming them they should look like this:

   - *NormLabSimulators*
   - *NormSynthesisMachine*

5.1. NormLab installation

Preparing the working environment

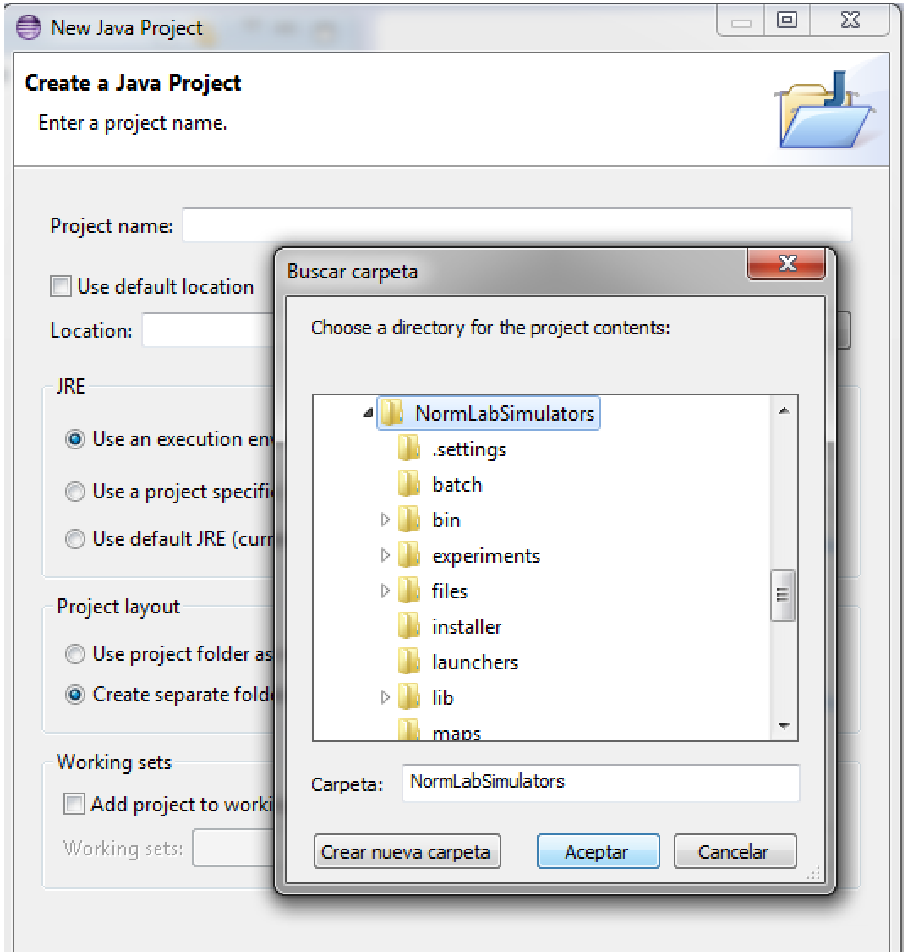
1. Open the **Repast Symphony IDE** (in Linux, open ***Eclipse IDE*** with Repast installed on it).
2. Import both projects ***NormSynthesisMachine*** and ***NormLabSimulators*** *in* ***Eclipse****.*
    1. *File>New>Java Project.*
    2. *Uncheck «Use default location» and click on «Browse».*



5.1. NormLab installation

Preparing the working environment

1. Unzip ***NormLabSimulators*** and ***NormSynthesisMachine*** projects to your HOME folder.
   - *For instance… «/Users/Javi/NormLab»*
2. Open ***Eclipse IDE***.
3. Import both projects ***NormLabSimulators*** and ***NormSynthesisMachine*** in ***Eclipse***:
   1. *File>New>Java Project.*
   2. *Uncheck «Use default location» and click on «Browse».*
1. Import projects ***NormLabSimulators*** and ***NormSynthesisMachine.***



5.2. NormLab structure

Before starting you need to know:

NormLabSimulators project is structured as follows:

src/onlinecomm: The code of the on-line community simulator.
src/traffic: The code of the traffic simulator.
launchers: The launchers that allow to run the two simulators.
repast-settings/OnlineCommunity.rs: Basic Repast settings for the on-line community simulator.
repast-settings/TrafficJunction.rs: Basic Repast settings for the traffic junction simulator.

Tutorial outline

What are the contents of this tutorial?

1. An **introduction** to NormLab
   1. The NormLab architecture.
   2. The Norm Synthesis Machine.
   3. The NormLab simulators.

2. **Configuration** of the working environment
   1. NormLab download.
   2. NormLab installation.

3. NormLab **execution**:
   1. Execution examples.
   2. Guided development of different norm synthesis strategies.

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

6. Example 1: Executing NormLab

TrafficJunction norm synthesis example 1

We are going to execute the ***TrafficJunction*** simulator with a very simple norm synthesis strategy, which is as follows:

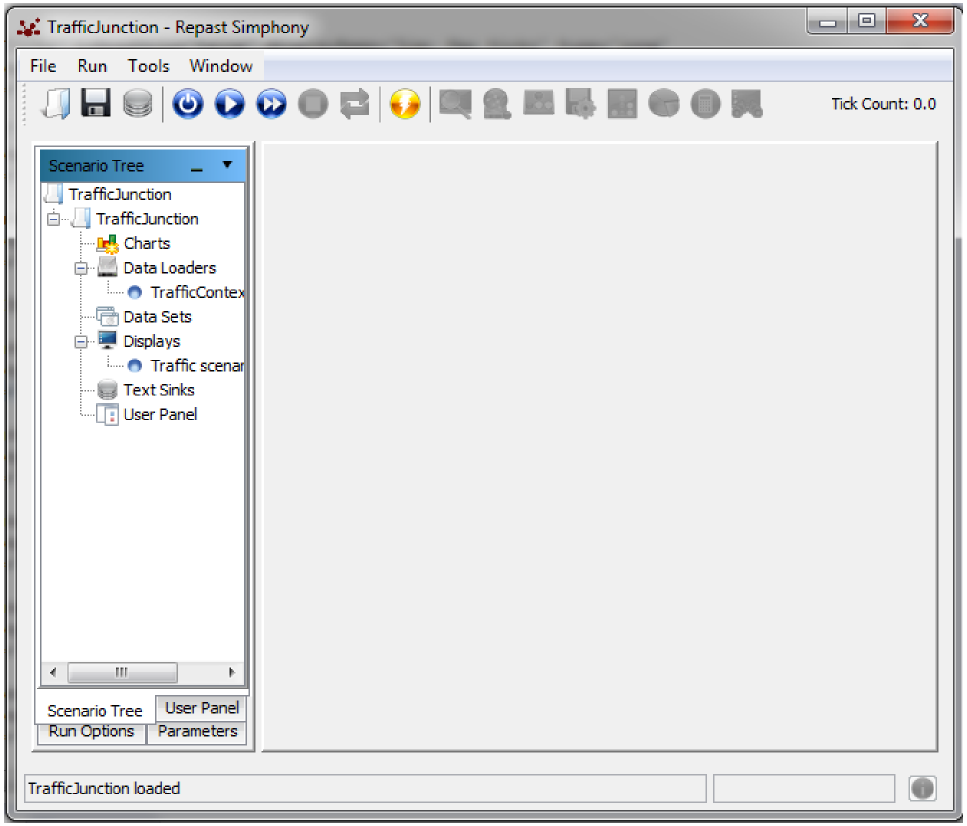
→ *Everytime the strategy is executed, return an **empty** normative system.*

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

6. Example 1: Executing NormLab

TrafficJunction norm synthesis example 1

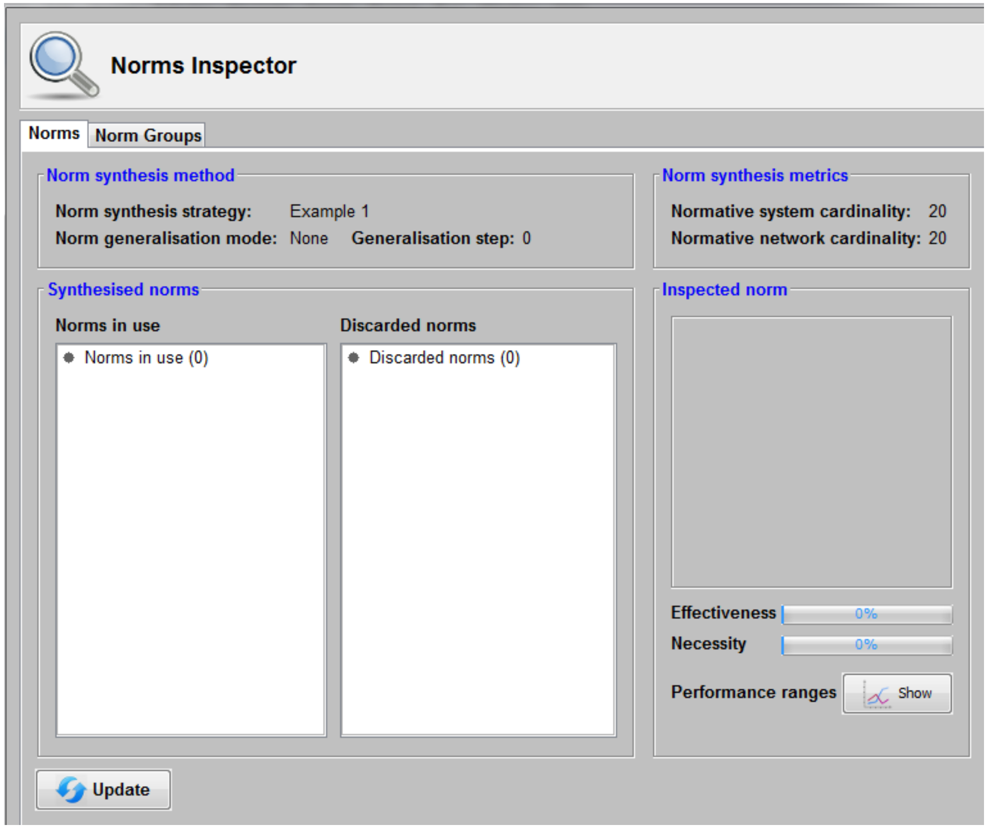
1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».



6. Example 1: Executing NormLab

TrafficJunction norm synthesis example 1

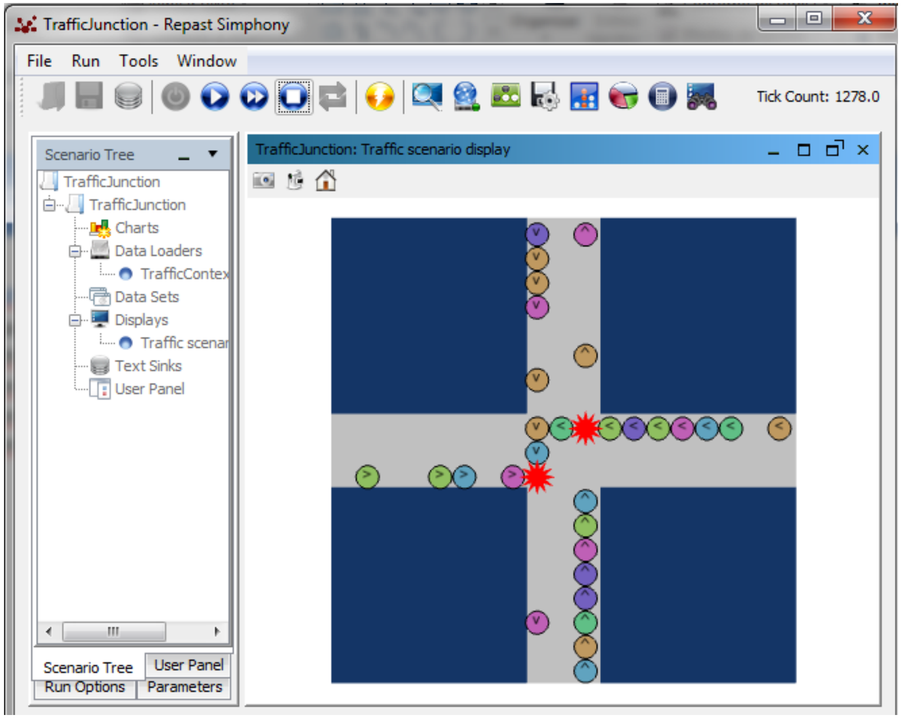
1. In Eclipse, in NormLabSimulators project, go to directory **launchers/**
2. Do right click on the file **TrafficJunctionSimulator.launch**.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.



6. Example 1: Executing NormLab

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. Example 1: Executing NormLab

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 

Tutorial outline

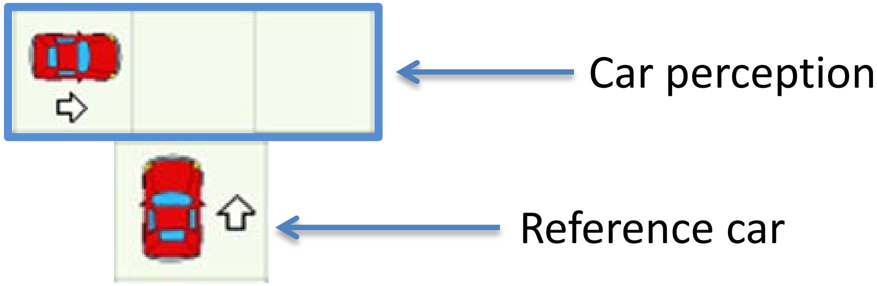
NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

7. Example 2: Using norms

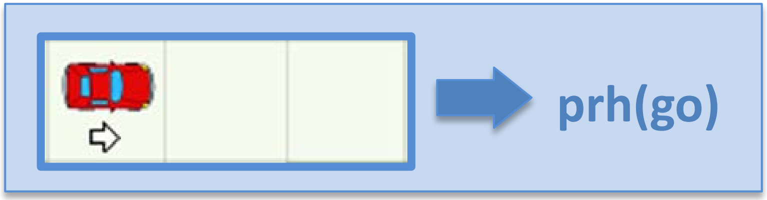
In the traffic simulator, cars perceive the scenario by means of the three cells in front of them:



Norms are…

- **IF** … **THEN**… rules.
- Norm precondition: Set of **predicates** with one **term** each.
    - Three different predicates **(left, front, right).**
    - Six different terms **(<, ^, >, v, -, w, *)** representing cars with different headings, term «-» stands for «nothing», «w» for «wall» and «*» for «anything».
- Norm postcondition: A **modality**.

Graphical representation



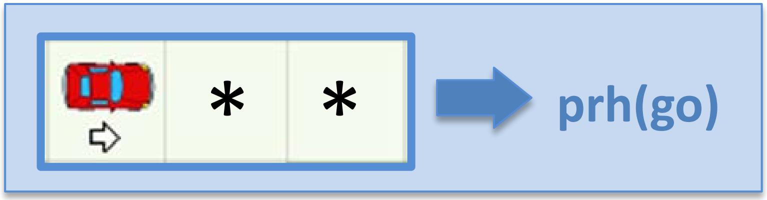
IF **left**(>) & **front**(-) & **right**(-) THEN **prohibition**(*go*)

7. Example 2: Using norms

TrafficJunction norm synthesis example 2

We are now going to execute the ***TrafficJunction*** simulator with a norm synthesis strategy that will avoid some (but not all) collisions between cars. With this aim, the strategy always returns a normative system with only one **left-side-priority** norm:

Norm 1



IF **left**(>) & **front**(*) & **right**(*) THEN **prohibition**(*go*)

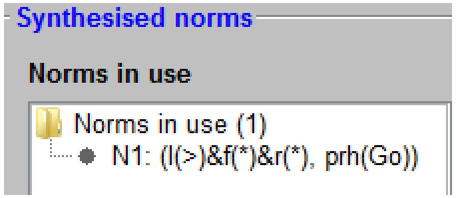
7. Example 2: Using norms

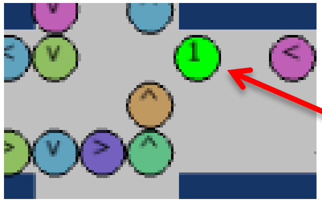
TrafficJunction norm synthesis example 2

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 traffic simulator setting parameters.
3. Search for the parameter «NormSynthesisExample».
4. Set the field «defaultValue» with the value «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.

```
<parameter  name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="2" />
```

5. Save the file.
6. Do right click on the file launchers/**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 one norm, and now cars occasionally stop to apply norm 1.





Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.
2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

8. Example 3: Removing collisions

TrafficJunction norm synthesis example 3

We are now going to execute the ***TrafficJunction*** simulator with a norm synthesis strategy that avoids all possible collisions. With this aim, it always returns the following normative system:

IF **left**(*) & **front**(^) & **right**(*) THEN **prohibition**(*go*)
IF **left**(>) & **front**(-) & **right**(*) THEN **prohibition**(*go*)
IF **left**(<) & **front**(<) & **right**(*) THEN **prohibition**(*go*)

To execute this example, you just have to **follow the steps in section 7**, but setting **defaultValue=«3»** of the NormSynthesisExample parameter (again in NormLabSimulators project, directory **repast-settings/TrafficJunction.rs ,** file **parameters.xml**)

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

9. Developing your own strategy

How are implemented all these examples? Let's implement one of the examples!

We are now going to develop our own norm synthesis strategy. In particular, we are going to implement the norm synthesis strategy of example 1, which returns an **empty normative system**.

The first thing we must do is to indicate *NormLab* that we are going to use a custom norm synthesis strategy. With this aim, follow these steps:

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 traffic simulator setting 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 give it a custom norm synthesis strategy.

```
<parameter  name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="0" />

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

9. Developing your own strategy

Now, to create your norm synthesis strategy, just follow these steps:

1. In Eclipse (NormLabSimulators project), go to **package es.csic.iiia.normlab.traffic.custom.**
2. There, create a new Java class *MyFirstStrategy.java* that implements the interface **es.csic.iiia.nsm.strategy.NormSynthesisStrategy.**
3. The interface will require you to implement two methods:
    1. **execute():** Executes the norm synthesis strategy
    2. **getNonRegulatedConflictsThisTick():** Returns a data structure containing the conflicts that the strategy has detected during the current tick

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

import java.util.List;
import java.util.Map;

import es.csic.iiia.nsm.config.Goal;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.norm.generation.Conflict;

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

  @Override
  public NormativeSystem execute() {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
    // TODO Auto-generated method stub
    return null;
  }
}
```

9. Developing your own strategy

We will create our strategy:

1. Create a new attribute **private Map<Goal, List<Conflict>> conflicts.**
2. Create a constructor for the class and, there, create the structure conflicts.
3. Make method **getNonRegulatedConflictsThisTick()** to return the attribute **conflicts.**
4. Your code should look like this:

```java
/**
 *
 */
public class MyFirstStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

  private Map<Goal, List<Conflict>> conflicts; // to save conflicts

  /**
   *
   */
  public MyFirstStrategy() {
    this.conflicts = new HashMap<Goal, List<Conflict>>();
  }

  @Override
  public NormativeSystem execute() {
    return null;
  }

  @Override
  public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
    return conflicts;
  }
}
```

9. Developing your own strategy

Now, let's implement the **execute()** method, which implements the norm synthesis strategy. This method must return an object *NormativeSystem*, that contains the norms that will be given to the agents.

There are a couple things that we must take into account:

- The Norm Synthesis Machine keeps synthesised norms in **a normative network**.
- To be able to access to the normative network, and the different elements of the Norm Synthesis Machine, we must receive the *NormSynthesisMachine* as a parameter in our strategy:

Follow now these steps:

1. In the constructor of the class, add the parameter ***es.csic.iiia.nsm.NormSynthesisMachine nsm.***
2. Now we can access the different elements of the Norm Synthesis Machine in our strategy.
3. Let's obtain the Normative Network! Add the following attribute to your class:

   private NormativeNetwork normativeNetwork;

4. Now, in your constructor, add the following code line to obtain the (initially empty) normative network:

   this.normativeNetwork = nsm.getNormativeNetwork();

5. Finally, we will now return an empty normative system at the end of the strategy execution. Add the following line of code at the end of method execute():

   return this.normativeNetwork.getNormativeSystem();

9. Developing your own strategy

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.iiia.normlab.traffic.custom;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import es.csic.iiia.nsm.config.Goal;
import es.csic.iiia.nsm.net.norm.NormativeNetwork;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.norm.generation.Conflict;

public class MyFirstStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

  private Map<Goal, List<Conflict>> conflicts; // to save conflicts
  private NormativeNetwork normativeNetwork;

  public MyFirstStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
    this.conflicts = new HashMap<Goal, List<Conflict>>();

    /* Get norm synthesis elements */
    this.normativeNetwork = nsm.getNormativeNetwork();
  }

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

  @Override
  public Map<Goal, List<Conflict>> getNonRegulatedConflictsThisTick() {
    return conflicts;
  }
}
```

10. Executing your implemented strategy

And now… how to tell *NormLab* to use your norm synthesis strategy?
We need to create an agent in the Traffic Simulator, which:

1. **Creates** and **configures** the Norm Synthesis Machine.
2. Adds **sensors** to the Norm Synthesis Machine to perceive the scenario.
3. **Creates** and **configures** the norm synthesis **strategy**.
4. **Executes** your strategy at every simulation step.

The traffic simulator incorporates a default Traffic Norm Synthesis Agent, which is implemented in class *DefaultTrafficNormSynthesisAgent* of package **es.csic.iiia.normlab.traffic.agent.**

Let's take a look at it…

10. Executing your implemented strategy

Observe the constructor **DefaultTrafficNormSynthesisAgent()**. It performs these tasks:

1. Creates the norm synthesis machine with a given configuration.
2. Adds a set of sensors to the norm synthesis machine in order to perceive the scenario.
3. Sets the norm synthesis strategy.

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains) {

  this.normativeSystem = new NormativeSystem();
  this.addedNorms = new ArrayList<Norm>();
  this.removedNorms = new ArrayList<Norm>();
  this.nsmSettings = new TrafficNormSynthesisSettings();
  this.dmFunctions = new TrafficDomainFunctions();

  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsmSettings, predDomains,
      dmFunctions, true);

  /* 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();
}
```

4. Executes the norm synthesis strategy at every simulation step.

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

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

10. Executing your implemented strategy

To create the NormSynthesisMachine, it needs to create:

1. **NormSynthesisSettings**: The settings for the norm synthesis machine.
2. **PredicatesDomains**: Information about the agents' language. That is, the predicates and terms the agents employ to describe the scenario from their local point of view.
3. **DomainFunctions**: Some domain-dependent functions that the Norm Synthesis Machine requires to synthesise norms (e.g., conflict detection, norm applicability).

```
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains) {

  this.normativeSystem = new NormativeSystem();
  this.addedNorms = new ArrayList<Norm>();
  this.removedNorms = new ArrayList<Norm>();
  this.nsmSettings = new TrafficNormSynthesisSettings();
  this.dmFunctions = new TrafficDomainFunctions();

  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsmSettings, predDomains,
      dmFunctions, true);

  /* 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. Executing your implemented strategy

NormSynthesisSettings: An interface to be implemented (located in package **es.csic.iiia.nsm.config**)

1. **getNormSynthesisStrategy():** Returns the norm synthesis strategy to use.
2. **getSystemGoals():** A list of system goals. In traffic, the only goal is “to avoid collisions”.
3. **getNormsDefaultUtility():** Norms’ default utility (0.5 by default).
4. **getNormEvaluationLearningRate():** The α rate to evaluate norms (0.1 is ok).
5. **getNormsPerformanceRangesSize():** The size of the window to compute norms’ performance ranges.
6. **getNormGeneralisationMode():** SIMON’s norm generalisation mode (Shallow/Deep).
7. **public int getNormGeneralisationStep():** SIMON’s norm generalisation step, namely the number of norm predicates that can be simultaneously generalised.
8. **getGeneralisationBoundary(Dimension dim, Goal goal):** Returns the minimum value of Effectiveness/necessity that a norm’s performance much reach to be generalised.
9. **getSpecialisationBoundary(Dimension dim, Goal goal):** Returns the value of Effectiveness/necessity under which a norm can be specialised.
10. **getNumTicksOfStabilityForConvergence():** The number of simulation ticks without conflicts or changes to the normative system to converge.

An implementation of these settings for the traffic simulator is located in package **es.csic.iiia.normlab.traffic.normsynthesis,** class *TrafficNormSynthesisSettings*

10. Executing your implemented strategy

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

The traffic simulator creates predicates and their domains in (project NormLabSimulators) class **es.csic.iiia.traffic.TrafficSimulator**, method **createPredicatesDomains().**

- Three different predicates **(l, f, r)** that represent the left, front and right positions in front of a car.
- Six different terms **(<, ^, >, v, -, w, *)** representing cars with different headings, term «-» stands for «nothing», «w» for «wall» and «*» for «anything».

10. Executing your implemented strategy

PredicatesDomains: The traffic simulator creates predicates and their domains in class **es.csic.iiia.traffic.TrafficSimulator**, method **createPredicatesDomains().**

```
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. Executing your implemented strategy

DomainFunctions: An interface to be implemented. Located in package **es.csic.iiia.nsm.config** (NormSynthesisMachine project)**.**

1. **isConsistent(SetOfPredicatesWithTerms agentContext):** Returns true if a set of predicates with terms is consistent with the domain. For instance, (left(>),front(-),right(-)) is consistent. By contrast, (left(>),front(<),right(-)) is not consistent, since two cars can not drive in opposite directions in the same lane.
2. **agentContextFunction(long agentId, View view):** Returns the local perception of a given agent at a particular system state (received as a View).
3. **agentActionFunction(long agentId,ViewTransition viewTransition):** Returns a list of the actions that an agent performed in the transition from a state $\mathbf{s_t}$ to a state $\mathbf{s_{t-1}}$
4. **getNonRegulatedConflicts(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).
5. **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, **es.csic.iiia.normlab.traffic.normsynthesis** package, class *TrafficDomainFunctions.*

10. Executing your implemented strategy

Now that we understand how *DefaultTrafficNormSynthesisAgent* works, let's tell it to use your norm synthesis strategy:

1. Open class *DefaultTrafficNormSynthesisAgent* in package **es.csic.iiia.normlab.traffic.agent.** This class implements the agent that «lives» in the traffic simulator, creates the norm synthesis machine and executes the strategy at every simulation tick.
2. Go to method **setCustomNormSynthesisStrategy()**
3. There, tell NormLab to use your norm synthesis strategy. Use this code:

```
/**
 * Sets a custom norm synthesis strategy
 */
protected void setCustomNormSynthesisStrategy() {
  MyFirstStrategy myStrategy = new MyFirstStrategy(this.nsm);
  this.nsm.useStrategy(myStrategy);
}
```

4. It is as simple as creating your norm synthesis strategy and telling the norm synthesis machine to use your strategy.
5. Execute the simulation as you did for example 1.

Congratulations, you are using your own strategy!

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

11. Adding norms to your strategy

Let's now **add some norms** to our strategy. We will use the same set of norms than used in the **example 2** (with one only left-hand-side priority norm).

1. In package es.csic.iiia.normlab.traffic.custom (NormLabSimulators project), **copy** your first norm synthesis strategy (*MyFirstStrategy.java*) as **a new strategy** *MySecondStrategy.java.*
2. To add norms to the normative network we need to know the system goals (in traffic, the only system goal is to avoid collisions). With this aim, add the following attribute to your strategy.
   - **private List<Goal> goals;**
3. Now obtain the system goals in your constructor:
   - **this.goals = nsm.getNormSynthesisSettings().getSystemGoals();**
4. Your code should look like this:

```
private List<Goal> goals;
private Map<Goal, List<Conflict>> conflicts; // to save conflicts
private NormativeNetwork normativeNetwork;

/**
 *
 * @param nsm
 */
public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
  this.conflicts = new HashMap<Goal, List<Conflict>>();

  /* Get norm synthesis elements */
  this.goals = nsm.getNormSynthesisSettings().getSystemGoals();
  this.normativeNetwork = nsm.getNormativeNetwork();

}

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

11. Adding norms to your strategy

1. Let's create the normative system. Norms have four elements: (1) a norm precondition; (2) a modality (in our case, a prohibition); (3) an action to obligate/prohibit. In our implementation, the norm also includes the goal it is aimed to achieve.
2. Now, create a new method **createNormativeSystem()** that will add the norms to the normative network:

```java
/**
 * Creates a normative system that allows all possible collisions in the
 * road traffic scenario
 */
private void createNormativeSystem() {

  /* Get system goal (to avoid collisions) */
  Goal goalAvoidCollisions = goals.get(0);

  /* 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, goalAvoidCollisions);

  /* Add the norms to the normative network and activate them */
  this.normativeNetwork.add(n1);
  this.normativeNetwork.activate(n1);
}
```

3. This code first gets the only system goal (to avoid collisions between cars)
4. Then, it creates a norm precondition (set of predicates with terms) and adds the predicates «l» (left), «f» (front) and «r» (right), with its corresponding term.
5. Finally, it creates the norm adding the pre-condition, the modality «Prohibition» over the action «Go», and the goal of the norm (to avoid collisions).

11. Adding norms to your strategy

1. Now, call method **createNormativeSystem()** at the end of your constructor. Your code should look like this:

```java
/**
 *
 */
public class MySecondStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {

	private List<Goal> goals;
	private Map<Goal, List<Conflict>> conflicts; // to save conflicts
	private NormativeNetwork normativeNetwork;

	/**
	 *
	 * @param nsm
	 */
	public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
		this.conflicts = new HashMap<Goal, List<Conflict>>();

		/* Get norm synthesis elements */
		this.goals = nsm.getNormSynthesisSettings().getSystemGoals();
		this.normativeNetwork = nsm.getNormativeNetwork();

		this.createNormativeSystem();
	}

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

2. At each execution, the strategy will return the norms that are active in the normative network (i.e., the normative system).

11. Adding norms to your strategy

To finish, set the traffic norm synthesis agent to use your new strategy.

1. Open class *DefaultTrafficNormSynthesisAgent* in package **es.csic.iiia.normlab.traffic.agent.** This class implements the agent that «lives» in the traffic simulator, creates the norm synthesis machine and executes the strategy at every simulation tick.
2. Go to method **setCustomNormSynthesisStrategy()**
3. There, tell *NormLab* to use your norm synthesis strategy. Use this code:

```
/**
 * Sets a custom norm synthesis strategy
 */
protected void setCustomNormSynthesisStrategy() {
  MySecondStrategy myStrategy = new MySecondStrategy(this.nsm);
  this.nsm.useStrategy(myStrategy);
}
```

4. You can now execute the Traffic Simulator and see how your second strategy works. Observe that:
   1. The normative system contains now one norm.
   2. The unique norm is never evaluated (click on button *Show* of norms’ performance ranges).

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

12. A strategy with automatic norm generation

Let's see now how can we automatically generate norms on-line.

For this example we are going to use the code of example 4, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex4.**

There, we can find the following classes:

TrafficNSExample4_NSAgent
The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample4_NSOperators
Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample4_NSStrategy
A norm synthesis strategy that generates norms to avoid arisen collisions in the future.

12. A strategy with automatic norm generation

Let's see now how can we automatically generate norms on-line.

For this example we are going to use the code of example 4, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex4.**

There, we can find the following classes:

TrafficNSExample4_NSAgent
The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample4_NSOperators
Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample4_NSStrategy
A norm synthesis strategy that generates norms to avoid arisen collisions in the future.

This agent works along the lines of the DefaultTrafficNormSynthesisAgent

12. A strategy with automatic norm generation

TrafficNSExample4_NSOperators: How do operators work?

Create:

1. Receives a *Conflict* and a system *Goal.*
2. Employs a Case-Based Reasoning (CBR) norm generation approach to generate a norm aimed at avoiding the given conflict in the future.
3. If the norm does not exist in the normative network, then it adds it.
4. If the norm exists in the normative network, then it activates it (since it may be inactive).

Add:

1. Adds a norm to the normative network.
2. Activates the norm in the normative network.

Activate:

1. Sets the state of a norm as «Active» in the normative network

Deactivate:

1. Sets the state of a norm as «Inactive» in the normative network.

   → This operator is not invoked in this example since it does not refine norms (and hence does not deactivate norms).

12. A strategy with automatic norm generation

TrafficNSExample4_NSStrategy: How does the norm synthesis strategy work?

Everytime the strategy is executed, it:

1. **Perceives** the scenario by means of the monitor. It saves perceptions in the form of *ViewTransitions*. A *ViewTransition* describes a part of the scenario at time t-1 and at time t (that is, its transition from the previous to the current tick).

```java
/**
 * Calls scenario monitors to perceive agents interactions
 *
 * @return a {@code List} of the monitor perceptions, where each perception
 *         is a view transition from t-1 to t
 */
private void obtainPerceptions(List<ViewTransition> viewTransitions) {
  this.monitor.getPerceptions(viewTransitions);
}
```

2. **Detects conflicts** in perceptions by invoking method **getNonRegulatedConflicts**() of *DomainFunctions*.

```java
/**
 * Given a list of view transitions (from t-1 to t), this method
 * returns a list of conflicts with respect to each goal of the system
 *
 * @param viewTransitions the list of perceptions of each sensor
 */
protected Map<Goal, List<Conflict>> conflictDetection(
    List<ViewTransition> viewTransitions) {
  this.conflicts.clear();

  /* Conflict detection is computed in terms of a goal */
  for(Goal goal : this.nsmSettings.getSystemGoals())    {
    List<Conflict> goalConflicts = new ArrayList<Conflict>();

    for(ViewTransition vTrans : viewTransitions) {
      goalConflicts.addAll(dmFunctions.getNonRegulatedConflicts(goal, vTrans));
    }
    conflicts.put(goal, goalConflicts);
  }
  return conflicts;
}
```

12. A strategy with automatic norm generation

3. **Generates norms** (one for each detected conflict) by means of operator **create**.

```
/**
 * 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);
    }
  }
}
```

To execute this strategy, follow these steps:

1. In Eclipse, 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=«4»**.

Execute the simulator and see how, as long as cars collide, it generates norms to avoid those collisions in the future.

Tutorial outline

NormLab **execution**:

1. Execution examples
   1. **Example** strategy 1: Returns an **empty** set of norms.
   2. **Example** strategy 2: Returns a fixed set of **1 norm**.
   3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
   1. **Development** of example strategy 1: **Empty** set of norms.
   2. **Development** of example strategy 2: Fixed set of **1 norm**.
   3. **Studying** example 4: A strategy with norm **generation**.
   4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
   5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

13. Automatic norm generation + evaluation

We have seen how to automatically **generate** norms on-line.
Let's see now how can we automatically **evaluate** norms on-line.

For this example we are going to use the code of **example 5**, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex5.**

There, we can find the following classes:

TrafficNSExample5_NSAgent

The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample5_NSOperators

Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample5_NSStrategy

A norm synthesis strategy that generates norms to avoid arisen collisions in the future, and continuously **evaluates** them in base of their outcomes in the scenario.

TrafficNSExample5_NSUtilityFunction

A function to **evaluate** norms' utility based on their outcomes whenever agents fulfill/infringe norms.

13. Automatic norm generation + evaluation

We have seen how to automatically **generate** norms on-line.
Let's see now how can we automatically **evaluate** norms on-line.

For this example we are going to use the code of **example 5**, which is located in the package **es.csic.iiia.normlab.traffic.examples.ex5.**

There, we can find the following classes:

TrafficNSExample5_NSAgent
The agent that creates the Norm Synthesis Machine and executes the strategy.

TrafficNSExample5_NSOperators
Operators to **create**, **add**, **activate** and **deactivate** norms in the normative network.

TrafficNSExample5_NSStrategy
A norm synthesis strategy that generates norms to avoid arisen collisions in the future, and continuously **evaluates** them in base of their outcomes in the scenario.

TrafficNSExample5_NSUtilityFunction
A function to **evaluate** norms' utility based on their outcomes whenever agents fulfill/infringe norms.

You know how these things work…

13. Automatic norm generation + evaluation

How does norm evaluation work?

- Norm fulfilled + no conflicts → ***Effective*** norm (It avoids conflicts).
- Norm fulfilled + conflicts → ***Ineffective*** norm (It does not avoid conflicts).

- Norm infringed + no conflicts → ***Unnecessary*** norm (No conflicts arise when it is not fulfilled).
- Norm infringed + conflicts → ***Necessary*** norm (Conflicts arise when it is not fulfilled).

To evaluate norms at each tick, the norm synthesis strategy requires to retrieve:

1. The norms that have been **fulfilled** and **infringed** during the transition from the previous tick to the current tick.
2. Information about whether norm fulfilments and infringements led to conflicts or not in the current tick.

13. Automatic norm generation + evaluation

TrafficNSExample5_NSStrategy: How does this new norm synthesis strategy work?

Everytime this particular strategy is executed, it performs **norm generation** + **norm evaluation**. You already know norm generation. But… How is norm evaluation implemented?

Norm evaluation consists on the following steps:

1. **Compute norm applicability**, namely to 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;
}
```

- As you can see in the code, for each *ViewTransition* it employs a ***NormReasoner*** to compute the norms that apply to each agent in the viewTransition.
- The *NormReasoner* employs the *DomainFunctions* to retrieve the norms that apply to each agent.

13. Automatic norm generation + evaluation

2. **Compute norm compliance**, namely to assess if agents **complied or not** with their applicable norms during the transition from the previous tick (time t-1) to the current tick (time t), and if they lead to **conflicts** or not.

```
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

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

```
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);
      }
    }
  }
}
```

Each norm is evaluated in terms of:

- **The system goals**. Are norms useful to achieve system goals?
  Example: In the case of traffic, are norms useful to avoid car collisions?

- **Two dimensions**, effectiveness and necessity. Are norms effective to avoid collisions? Are they necessary to avoid collisions?

13. Automatic norm generation + evaluation

Finally, the **normEvaluation**() method puts together norm applicability, norm compliance and update utilities:

```
/**
 * Executes the norm evaluation phase
 */
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);
}
```

Let's execute this strategy. Follow these steps:

1. In Eclipse, 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=«5»**. This will indicate NormLab that we do not want to load a pre-designed example.

Execute the simulator and see how now it generates norms and evaluates them. Observe how the effectiveness and necessity of norms change along time.

Tutorial outline

NormLab **execution**:

1. Execution examples
    1. **Example** strategy 1: Returns an **empty** set of norms.
    2. **Example** strategy 2: Returns a fixed set of **1 norm**.
    3. **Example** strategy 3: Returns a fixed set of **3 norms**.

2. Guided development of different norm synthesis strategies
    1. **Development** of example strategy 1: **Empty** set of norms.
    2. **Development** of example strategy 2: Fixed set of **1 norm**.
    3. **Studying** example 4: A strategy with norm **generation**.
    4. **Studying** example 5: A strategy with norm **generation** + **evaluation**.
    5. **Studying** SIMON: A strategy with norm **generation** + **evaluation** + **refinement.**

14. SIMON. A complete norm synthesis strategy

We are now going to see how to implement a complete norm synthesis strategy that performs:

1. Norm generation
2. Norm evaluation
3. Norm refinement

You already know these phases

Let's see how SIMON refines the normative system

With this aim, we will execute the SIMON norm synthesis strategy. First of all, let's tell NormLab that we want to execute SIMON:

1. In Eclipse, 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=«2».** This will indicate *NormLab* that we want to use the **SIMON** norm synthesis strategy.
5. Search for the parameter «**NormGeneralisationMode**» and set the field **defaultValue=«1».** This will indicate *NormLab* that we want SIMON to use ***Deep*** norm generalisation.
6. Search for the parameter «**NormGeneralisationStep**» and set the field **defaultValue=«1».** This will indicate *NormLab* that we want SIMON to generalise just one norm predicate simultaneously in each norm generalisation.

14. SIMON. A complete norm synthesis strategy

Your **parameters.xml** file should look like this:

```
<parameter  name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="0" />

<parameter  name="NormSynthesisStrategy" isReadOnly="false" displayName="NSM: Norm synthesis strategy (CUSTOM/IRON/SIMON/XSIMON)" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="2" />

<parameter  name="NormGeneralisationMode" isReadOnly="false" displayName="NSM: Norm generalisation mode (SHALLOW/DEEP)" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="1" />

<parameter  name="NormGeneralisationStep" isReadOnly="false" displayName="NSM: Norm generalisation step" type="int"
            converter="repast.simphony.parameter.StringConverterFactory$IntConverter"
            defaultValue="1" />
```

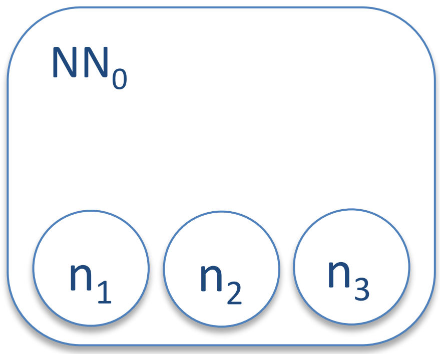
14. SIMON. A complete norm synthesis strategy

Norm refinement: Generalises norms when possible, and **specialises** norms when necessary.

- Norm generalisations allow to synthesise compact normative systems by generalising several norms to one unique norms that implicitly represents them.

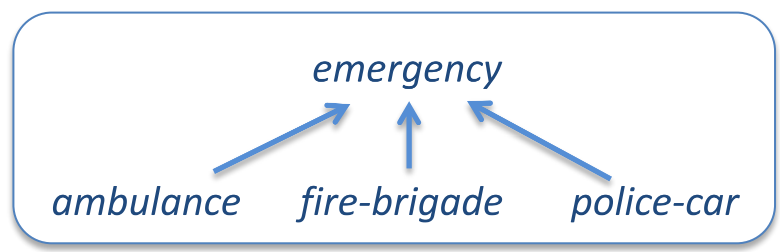
14. SIMON. A complete norm synthesis strategy

Norm generalisations allow to increase the compactness of the normative system



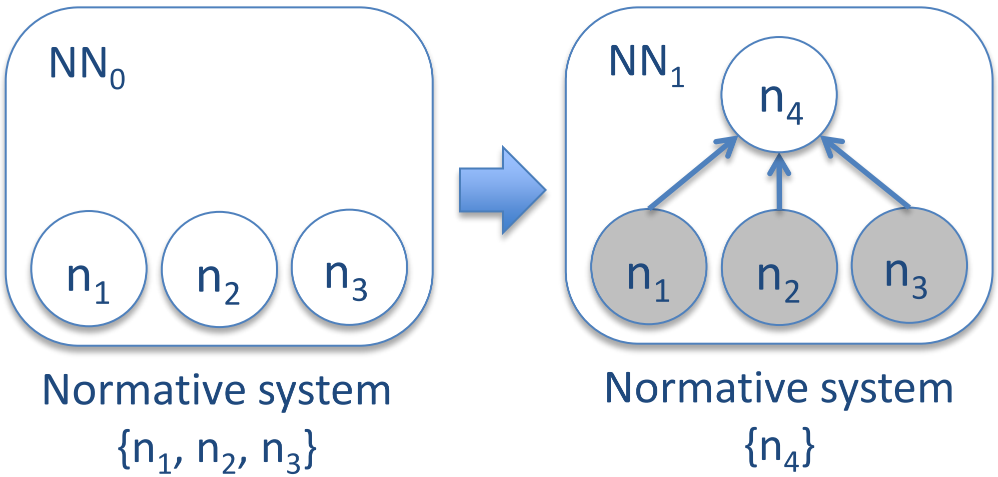
Normative system
$\{n_1, n_2, n_3\}$

n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**
n_3: Give way to **police cars**

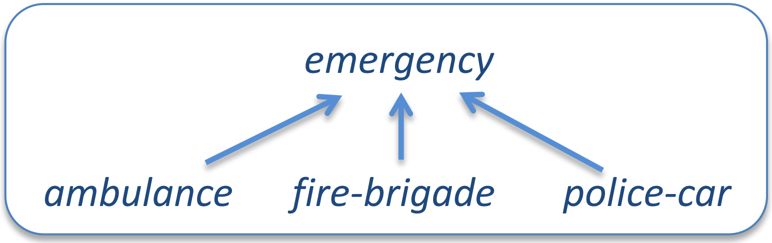


14. SIMON. A complete norm synthesis strategy

Norm generalisations allow to increase the compactness of the normative system

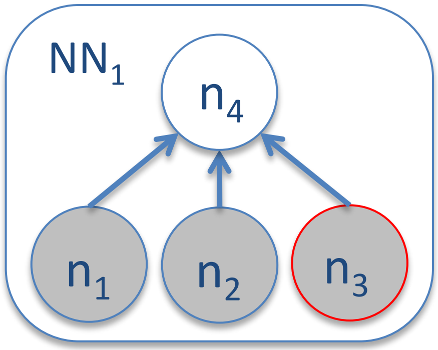


n_1: Give way to **ambulances**
n_2: Give way to **fire brigade**
n_3: Give way to **police cars**
n_4: Give way to **emergency** vehicles



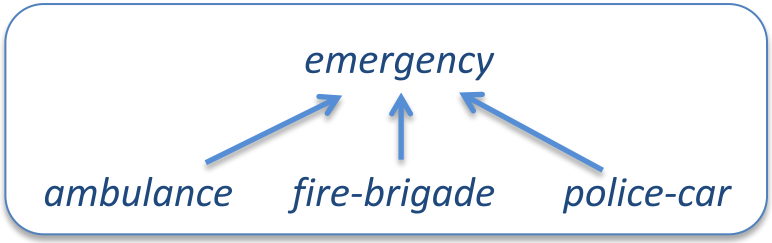
14. SIMON. A complete norm synthesis strategy

Norm specialisations allow to remove from the normative system those norms that under-perform



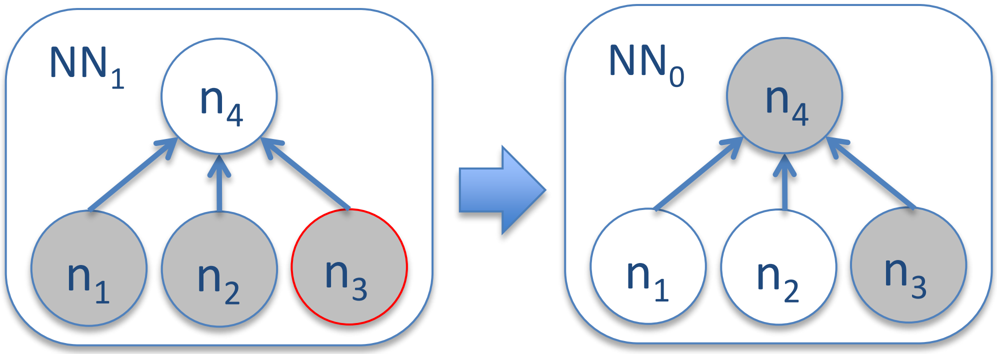
Normative system
$\{n_4\}$

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

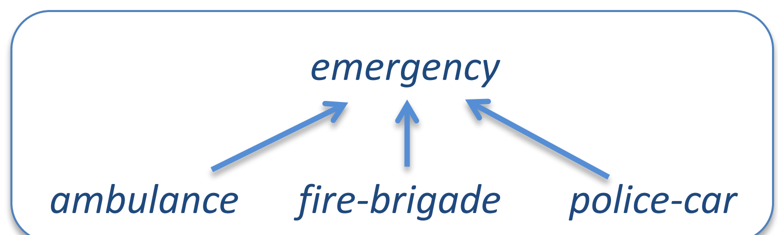
Norm specialisations allow to remove from the normative system those norms that under-perform



Normative system $\{n_4\}$

Normative system $\{n_1, n_2, n_3\}$

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



14. SIMON. A complete norm synthesis strategy

Norm refinement: Generalises norms when possible, and **specialises** norms when necessary.

1. Norms are **generalised** whenever their effectiveness **and** necessity are **over** a generalisation threshold.
2. Norms are **specialised** whenever their effectiveness **or** necessity are **under** a specialisation theshold.

```java
private void normRefinement() {

  /* Add norms that have been rewarded with a negative value */
  this.addNegRewardedNorms(negRewardedNorms);

  /* Monitor norm utilities to detect utilities passing thresholds */
  this.checkThresholds();

  /* Specialise norms that under perform */
  for(Norm norm : this.specialisableNorms)  {
    specialiseDown(norm);
  }

  /* Generalise norms that may be generalised */
  for(Norm norm : this.generalisableNorms) {
    generaliseUp(norm, genMode, genStep);
  }

  /* Link new norms that have been generalised to
   * other potential child norms in the normative network */
  for(Norm normA : this.createdNorms) {
    for(Norm normB : this.normativeNetwork.getActiveNorms()) {
      if(!normA.equals(normB))
        this.searchRelationships(normA, normB, null, visitedNorms);
    }
  }
}
```