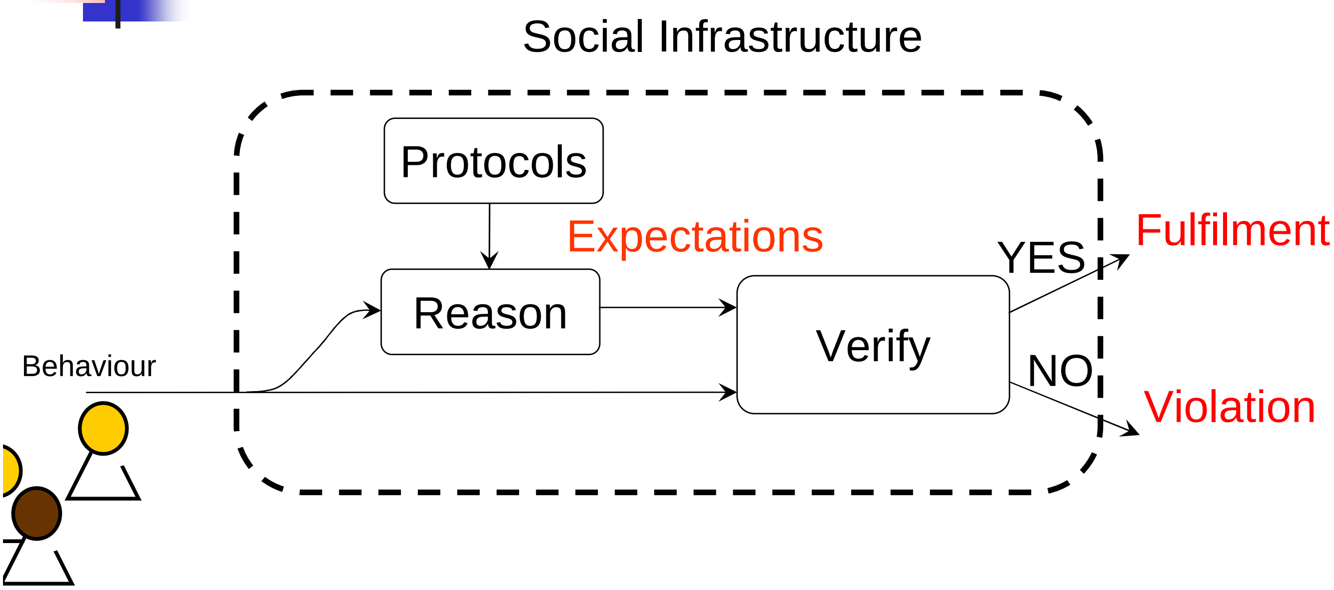
Specification and verification of agent interaction using *SOCS-SI*

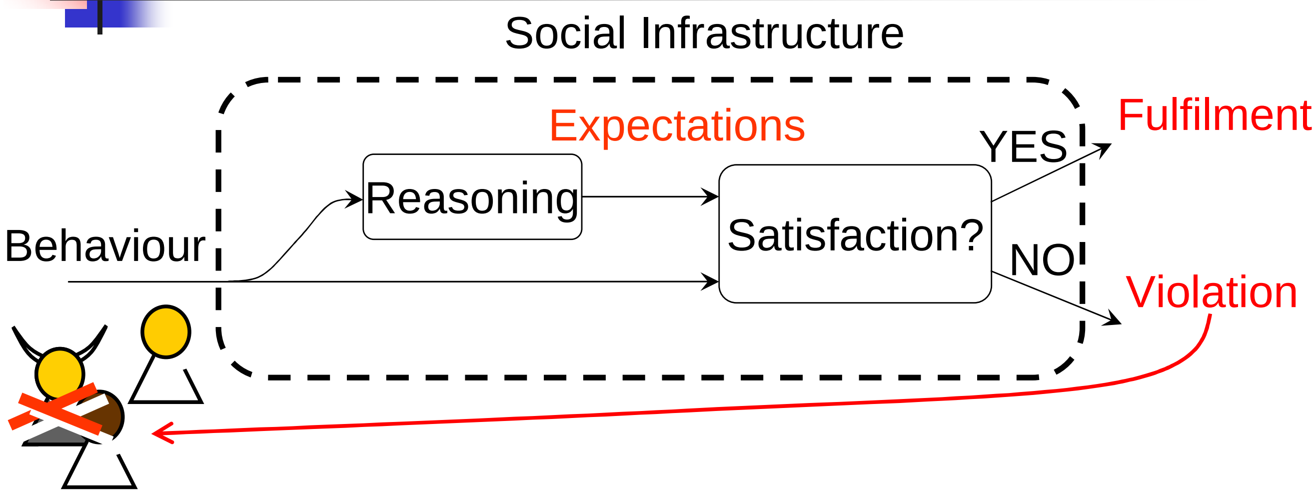
Federico Chesani – Università di Bologna

Marco Gavanelli – Università di Ferrara

Social Infrastructure

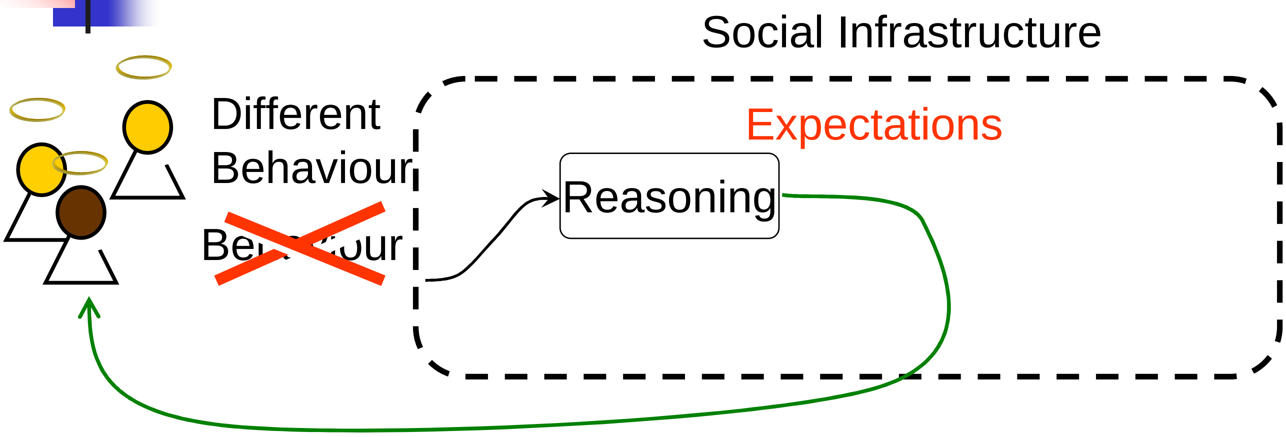


Use of expectations



(1) check of compliance to protocols

Use of expectations



(2) feedback loop that influences behaviour

Part I: Synopsis

- Types of society
- The *S*CIFF language
  - syntax and semantics
- Verification
  - Types of verification

Multi-Agent Systems (MAS)

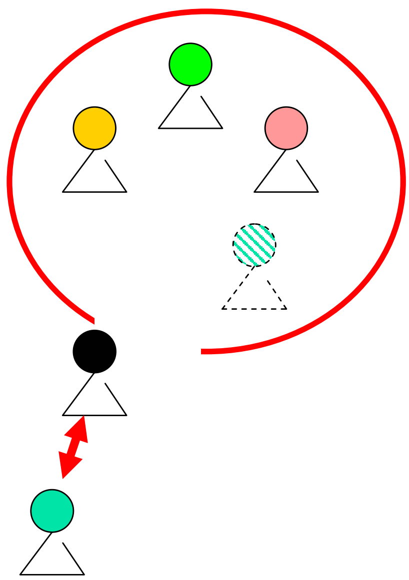
- Agent: a software system
  - Autonomous: the agent is able to work autonomously
  - Reactive: it is able to react to external stimuli, changes in the environment
  - Proactive: it can have objectives, goal-directed behaviour
  - Social: can interact with other agents to reach its goals

Design

- Design a MAS means designing
  - Type of the society Open
  - Interaction
  - Roles
  - …

Society types [Davidsson 01]

- Closed: Agents cannot enter (fixed number of agents)
- Semiclosed: Agents cannot enter, but can spawn a representative in the society
- Semiopen: agents can enter by registering to a gatekeeper
- Open: agents can enter without restrictions



Open society

"it supports openness and flexibility very well, but it is very difficult to make such a society stable and trustful." [Davidsson 01]

- But, we design the MAS in order to obtain some goals …
  - Give semantics to communication
  - Agents comply to rules
  - The MAS reaches its goals
  - The MAS has some required properties

Protocols

- Definition: language
- Impose that agents behave according to protocols?
  - Not in an open society!
  - Verification / raising violations
- Protocol properties

Example: Auction

- We want to design a MAS for managing auctions
- We have to
  - Design the communication acts, and their semantics
  - Design the protocol
  - Use the protocol for
    - Guiding the agents behaviour
    - Prove society properties

Syntax: events

- happened events (ground)

$$\mathbf{H}(Desc, Time)$$

- Desc (term)
- Time (integer)
- Eg $\mathbf{H}(tell(bob, alice, bid(pen, \mathit{1}\ \$), auc1), 3)$

  Bob tells Alice that he bids 1$ for the pen in the auction *auc*1 at time 3
- Events compose a *history* $\mathbf{HAP}$

Syntax: Expectations

- Events that should / should not happen

  $\mathbf{E}(Desc, Time)$ $\quad$ $\mathbf{EN}(Desc, Time)$

- Eg $\mathbf{E}(tell(alice, bob, answ(A, pen, 1\$), auc1), T_{Ans}), T_{Ans} > 3$

  Alice should answer to Bob's bid, after time 3
- Eg $\mathbf{EN}(tell(B, alice, bid(pen, P), auc1), T_{Bid}), T_{bid} > 3, P < 1\$$

  No agent should place a bid to Alice for the pen in auction 1 for less than 1$, after time 3
- Expectations compose the set Δ

Declarative semantics

- *S*CIFF: abductive semantics

  $KB \cup \Delta \models G$ $\quad$ $KB \cup \Delta \models IC$

- Coherence of set Δ

  $$\forall p, \mathbf{E}(p), \mathbf{EN}(p) \notin \Delta$$

  $$\forall p, \neg\mathbf{E}(p), \mathbf{E}(p) \notin \Delta \qquad \forall p, \mathbf{EN}(p), \neg\mathbf{EN}(p) \notin \Delta$$

- Compliance to protocol

  $$\forall p, \mathbf{E}(p) \rightarrow \mathbf{H}(p) \qquad \forall p, \mathbf{EN}(p) \rightarrow not\mathbf{H}(p)$$

Syntax

- Social Organization Knowledge Base (SOKB)
  - clauses *Atom* ← *Cond*
  - *Cond:* conjunction of literals, constraints, expectations
- Social Integrity Constraints (ICs)
  - *Body* → *Head*
  - *Body:* conjunction of literals defined in SOKB, H, E, EN and CLP constraints
  - *Head:* a disjunction of conjunction of E, EN literals and CLP constraints

Auction: communicative acts

- Openauction: opens an auction for an item
- Bid: propose to buy an item for a given price
- Answer (win/lose): communicate if a bid wins or loses
- Deliver: provide the item
- Pay: pay for the item

Auction: semantics of communicative acts

- If I open an auction, I am willing to give the item for some amount of money

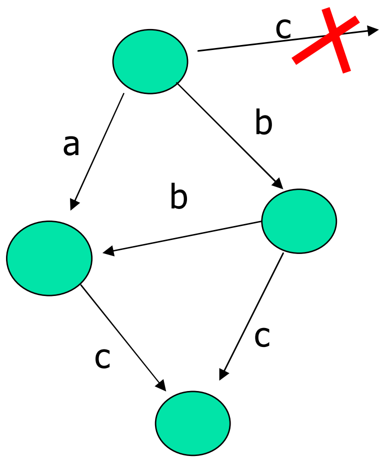
$$
\begin{array}{l}
\mathbf{H}(tell(A, Bidders, opauc(Item, \tau, T_{notify}, Type), D), T_{open}),\\
\mathbf{H}(tell(B, A, bid(Item, Q), D), T_{Bid}),\\
\mathbf{H}(tell(A, B, answ(win, Item, Q), D), T_{Win}),\\
\rightarrow \mathbf{E}(tell(A, B, deliver(Item), D), T_{Del}),\\
T_{Del} < T_{Win} + T_{Deliver\_Deadline}
\end{array}
$$

- If I place a bid, I am willing to pay such amount of money for the item

$$
\begin{array}{l}
\mathbf{H}(tell(B, A, bid(Item, Q), D), T_{Bid}),\\
\mathbf{H}(tell(A, B, answ(win, Item, Q), D), T_{Win}),\\
\mathbf{H}(tell(A, B, deliver(Item), D), T_{Del})\\
\rightarrow \mathbf{E}(tell(B, A, pay(Item, Q), D), T_{Pay}),\\
T_{Pay} < T_{Del} + T_{Pay\_Deadline}
\end{array}
$$

Protocols

- Protocols are often seen as finite state automata
- Define allowed moves, the rest is forbidden
- Could be a limit in open societies
- $\mathcal{S}$CIFF: define explicitly allowed/ necessary/ forbidden moves



Auctions

- Before placing bids, there must have been an openauction

$$\mathbf{H}(tell(S,R,bid(Item,P),D),Tbid)$$
$$\rightarrow \mathbf{E}(tell(R,\_,openauction(Item,Tend,\_),D),Topen)$$
$$\wedge Topen < Tbid \wedge Tbid \leq Tend.$$

- The auctioneer should reply to bids

$$\mathbf{H}(tell(B,A,bid(Item,P),Anumber),Tbid)$$
$$\wedge \mathbf{H}(tell(A,\_,openauction(Item,Tend,Tdeadline),D),Topen)$$
$$\rightarrow \mathbf{E}(tell(A,B,answer(Answer,B,Item),D),Tanswer)$$
$$\wedge Tanswer \geq Tend \wedge Tanswer \leq Tdeadline \wedge Answer :: [win,lose].$$

Auctions

- no contradicting answers

$$\mathbf{H}(tell(A,B,answer(A1,B,Itemlist),D),T1)$$
$$\rightarrow \mathbf{EN}(tell(A,B,answer(A2,B,Itemlist),D),T2)$$
$$A1 \neq A2$$

Auctions

- Payment

$$\begin{aligned}&\mathbf{H}(tell(A, Bw, answer(win, Bw, Item), D), Tw)\\ \wedge\ &\mathbf{H}(tell(Bw, A, bid(Item, P), D), Tbid)\\ \rightarrow\ &\mathbf{E}(tell(Bw, A, pay(P), D), Tp).\end{aligned}$$

English auction: protocol

- You cannot place bids lower than the previous

$$\begin{aligned}&\mathbf{H}(tell(Bidder_1, Auc, bid(Item, Q_1)), T_1)\\ \rightarrow\ &\mathbf{EN}(tell(Bidder_2, Auc, bid(Item, Q_2)), T_2), T_2 > T_1, Q_2 \leq Q_1\end{aligned}$$

- Either one places a higher within τ units after my bid, or I win

$$\begin{aligned}&\mathbf{H}(tell(Auc, Bidders, opauc(Item, \tau, T_{notify}, english), D), T_{open}),\\ &\mathbf{H}(tell(Bidder_1, Auc, bid(Item, Q_1), D), T_1)\\ \rightarrow\quad &\mathbf{E}(tell(Bidder_2, Auc, bid(Item, Q_2), D), T_2),\\ &Q_2 > Q_1, T_2 < T_1 + \tau\\ \vee\quad &\mathbf{E}(tell(Auc, Bidder_1, answ(win, Item, Q_1), D), T_{win}),\\ &T_{win} < T_1 + T_{notify}\end{aligned}$$

First price sealed bid auction

- Predefined deadline. Either there is a higher bid, or I must be declared winner

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{H}(tell(Auc, Bidders, opauc(Item, T_{dead}, T_{notify}, fpsb), D), T_{open}),} \\
\multicolumn{2}{l}{\mathbf{H}(tell(Bidder_1, Auc, bid(Item, Q_1), D), T_1), T_1 < T_{dead}} \\
\rightarrow & \mathbf{E}(tell(Bidder_2, Auc, bid(Item, Q_2), D), T_2), \\
 & Q_2 > Q_1, T_2 < T_{dead} \\
\vee & \mathbf{E}(tell(Auc, Bidder_1, answ(win, Item, Q_1), D), T_{win}), \\
 & T_{win} < T_{dead} + T_{notify}
\end{array}
$$

Vickrey auction

- You should pay at least the amount of the other bidders

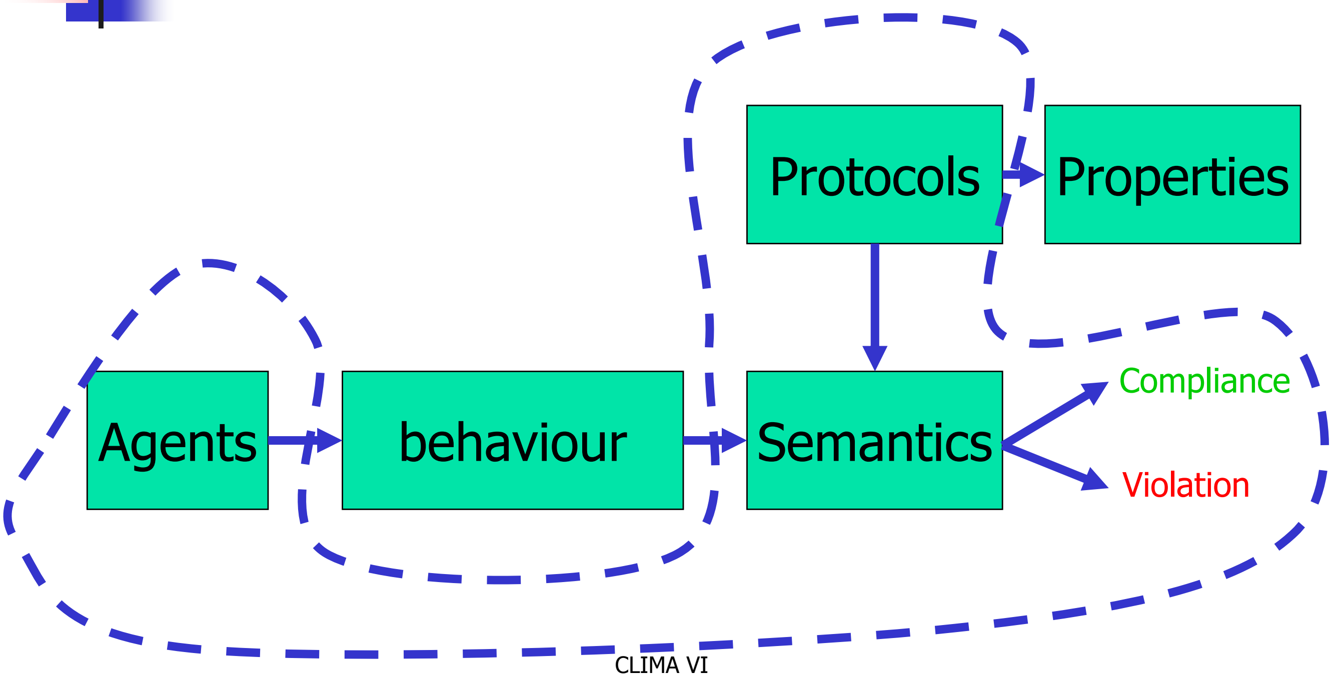
$$
\begin{array}{l}
\quad \mathbf{H}(tell(A, Bw, answer(win, Bw, Item), D), Tw) \\
\wedge\, \mathbf{H}(tell(A, Bl, answer(lose, Bl, Item), D), Tl) \\
\wedge\, \mathbf{H}(tell(Bl, A, bid(Item, Pl), D), Tbid) \\
\rightarrow \mathbf{E}(tell(Bw, A, pay(P), D), Tp) \wedge P \geq Pl.
\end{array}
$$

Verification

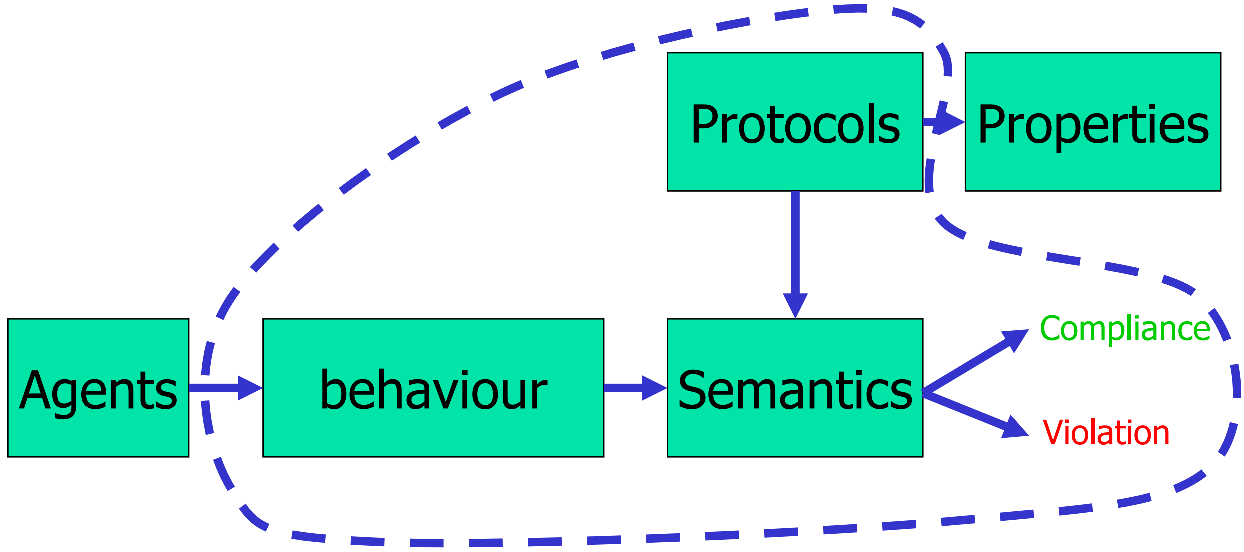
Types of verification [Guerin, Pitt 02]

- Type 1: An agent will always comply to protocol (required: agent specification, not available in open societies)
- Type 2: verification through observation (on the fly)
- Type 3: verification of protocol properties (if agents behave according to protocols, does the MAS respect specifications?)

Type 1 Verification



Type 2 verification



Operational Semantics

- *S*CIFF: Extension of the IFF abductive proof-procedure [Fung-Kowalski]

- Generation of expectations
- Abduction of literals with universally quantified variables
- Dynamically happening events
- CLP constraints on variables (both existentially and universally quantified)

Operational Semantics

- Data structure

T = <R,CS,PSIC,EXP,HAP,FULF,VIOL>

Where

- R: Conjunction of literals
- CS: Constraint Store
- PSIC: Implications
- EXP: (Pending) Expectations
- FULF: Fulfilled expectations
- VIOL: Violated Expectations

Transitions

- IFF-Like (extended)
- Fulfilment, violation
- Dynamically growing history
- Consistency
- CLP

IFF-like transitions

- unfolding: p(X), A   p(Y) ← B   → X=Y, B, A
- propagation: p(X), B → C   p(Y)   → X=Y, B → C
- splitting: distributes conjunctions and disjunctions
- case analysis: c(X,Y) → A
  - either   c(X,Y), A
  - or   ¬c(X,Y)
- factoring tries to reuse previously made hypotheses;
- rewrite rules: use the inferences in the Clark Equality Theory;
- logical simplifications A,false ↔ false, A ← true ↔ A, etc.

Fulfilment / Violation

- Violation EN:

$$EXP_k = \{EN(E_1), \ldots\}$$
$$HAP_k = \{H(E_2), \ldots\}$$

$CS_{k+1} = CS_k \cup \{E_1 = E_2\}$
$VIOL_{k+1} = VIOL_k \cup \{EN(E_1)\}$
$EXP_{k+1} = \{\ldots\}$

$CS_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

- Rely on the constraint solver

Fulfilment / Violation

- Fulfilment E:

$EXP_k = \{E(E_1), \ldots\}$
$HAP_k = \{H(E_2), \ldots\}$

$CS_{k+1} = CS_k \cup \{E_1 = E_2\}$
$FULF_{k+1} = FULF_k \cup \{E(E_1)\}$
$EXP_{k+1} = \{\ldots\}$

$CS_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Dynamically Growing History

- External set of incoming events
- A transition Happening takes an event from the external set and puts it into the HAP set

$T_n = \langle R_n, CS_n, PSIC_n, EXP_n, \mathbf{HAP_n}, FULF_n, VIOL_n \rangle$

H_1	H_2	H_3	…	

Happening

$T_{n+1} = \langle R_n, CS_n, PSIC_n, EXP_n, \mathbf{HAP_n} \cup \mathbf{\{H1\}}, FULF_n, VIOL_n \rangle$

Other transitions reason about non-happening, closure of the history

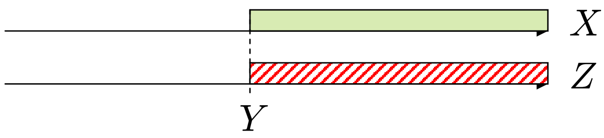
CLP

- Constraint solving
  - constrain
  - case-analysis
- Equalities & disequalities are constraints
- The solver also considers the quantification of variables

$\exists Y \forall X > Y\, \mathbf{EN}(p(X))$

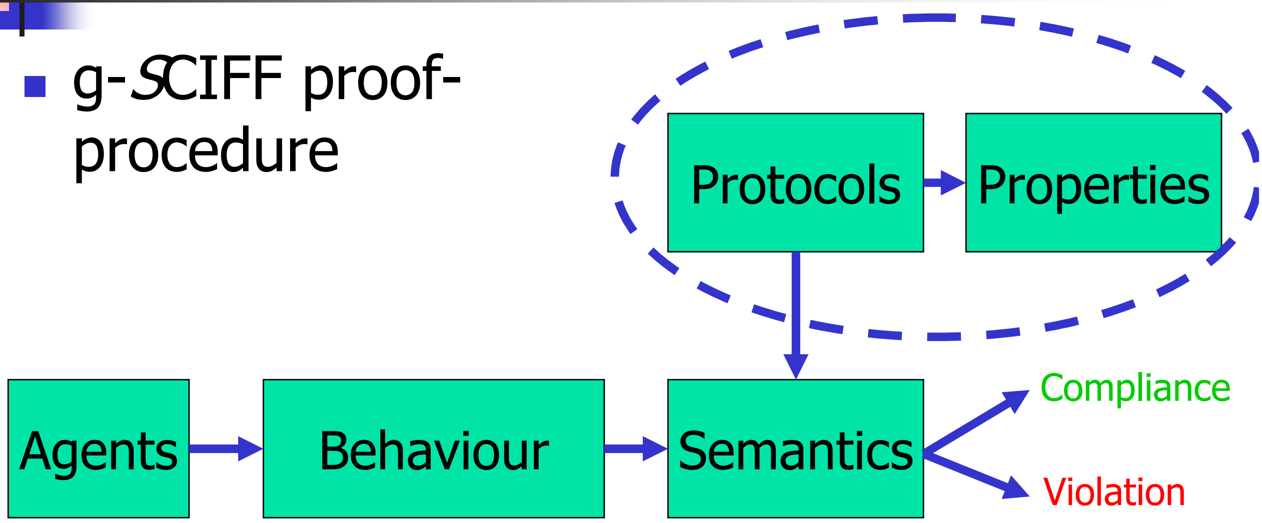
$\exists Z\, \mathbf{E}(p(Z))$

$Z \leq Y$



Type 3 verification

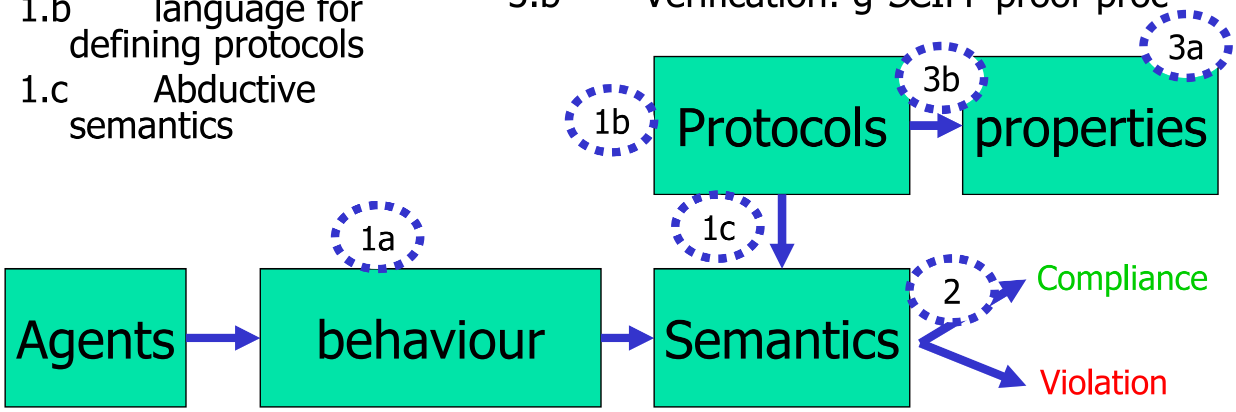
- g-*S*CIFF proof-procedure



*S*CIFF abductive framework

1 *S*CIFF abductive framework:
1.a representation of agents' behaviour
1.b language for defining protocols
1.c Abductive semantics

2 Proof-procedure *S*CIFF
3 Properties of protocols:
3.a representation
3.b Verification: g-*S*CIFF proof-proc



Specification and verification of agent interaction using *SOCS-SI*

Federico Chesani – Università di Bologna
Marco Gavanelli – Università di Ferrara

Part 2

Part 2
Outline

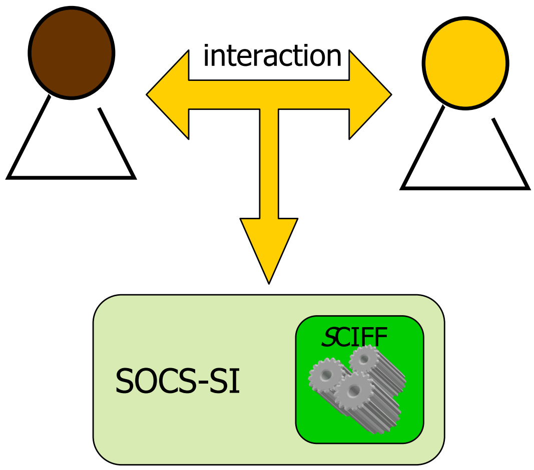
- Overview of SOCS-SI
- How to define a protocol (e.g. First-price, Sealed-bid, private values Auctions) and the related social knowledge base
- Simulating the defined protocol
- Property check of the defined protocol (work in progress)

The software SOCS-SI
Which use?

SOCS-SI is a tool for verifying agent interactions w.r.t. a defined protocol

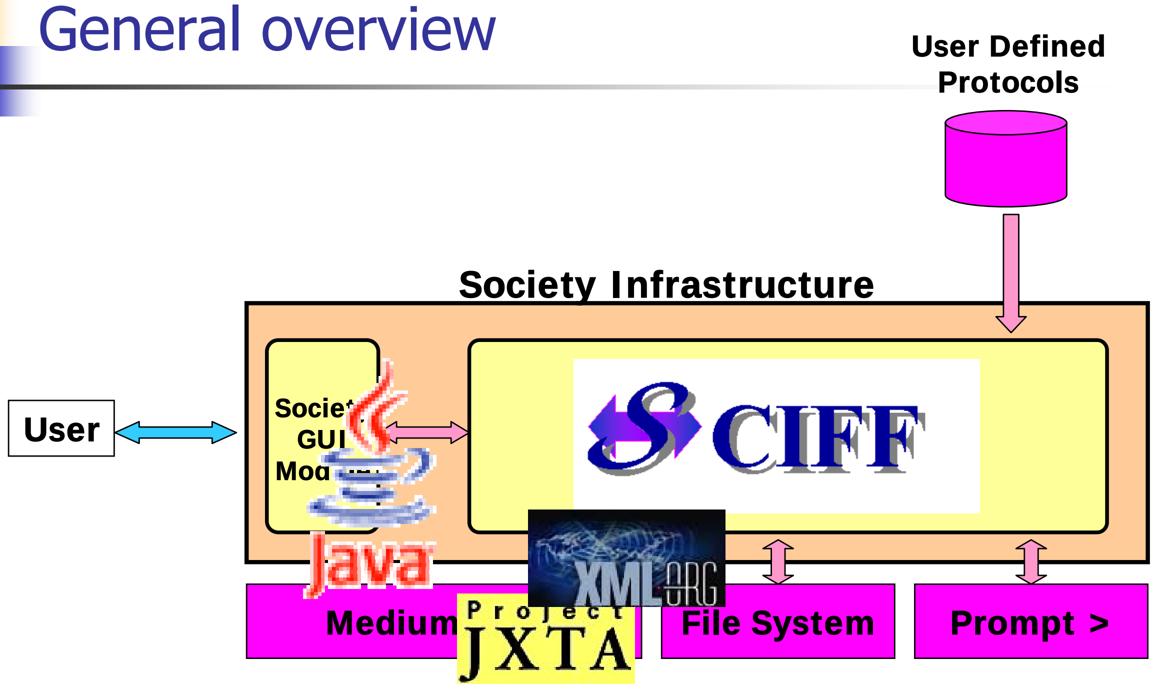
- It provides a way for formally specifying protocols
- It shows the state of the society as events occur (history, expectations,…)
- Given a protocol definition and an agent interaction, it verifies if the interaction "follows" the protocol
- Such a verification has been called "Type 2"

The software SOCS-SI
General overview

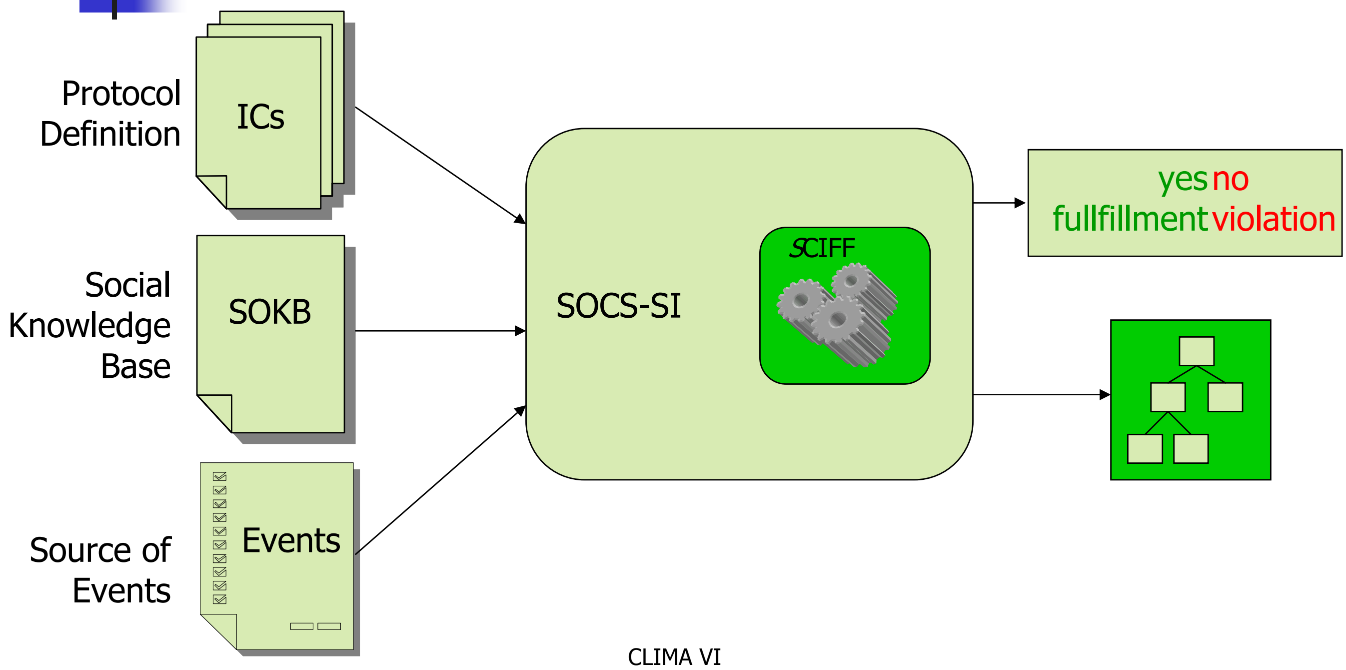


- We assume that SOCS-SI can access all relevant messages exchanged during an interaction

The software SOCS-SI
General overview



The software SOCS-SI
General Overview



The software SOCS-SI
Inputs

Inputs:

- A file containing the protocol description (ICs formalism)
- A file containing the social knowledge base
- A source of events
  - The SOCS agent communication platform
  - Jade platform
  - TuCSoN
  - E-Mail system
  - A log file
  - …

The software SOCS-SI
Outputs (1)

Outputs:

- An answer (yes/no) about the compliance of the happened (or happening) events w.r.t. the given protocol *S* CIFF
- It is possible at any time to inspect the state of the system in case no more events will ever occur ("closure" of the history)

The software SOCS-SI
Outputs (2)

- The tree explored by the *S*CIFF proof procedure
- "Browsing" of the tree:
  - Each node represents a quiescence state of the proof
  - Each intermediate node can be accessed
  - In each node the state of the proof can be inspected

The software SOCS-SI
Running SOCS-SI

In order to execute the tool, the inputs must be provided. Several ways:

- **`run`**

  Loads and executes using default configuration file, "society.config"

- **`run --config config_file`**

  Loads and executes using the settings stored in the file config_file

- **`run --graphic`**

  Open a gui for selecting by hand the inputs

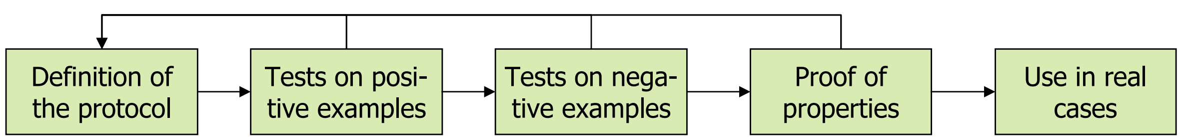
- **`run --manual …`**

  Specifies the input parameters directly on the command line

Defining a protocol
Using ICs to define a protocol

- SOCS-SI can be used as a platform to try different methodologies for designing protocols
- Ongoing work: definition of such a general methodology

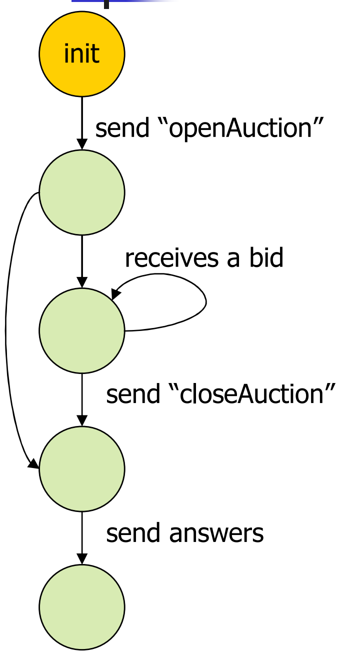
- A possibility is below:



Defining a protocol
The auction example

- We will consider a "first-price sealed-bid" auction with private values
- Each bidder submits one bid, without knowing the others' bid.
- The highest bid wins the item and pays the amount of his bid

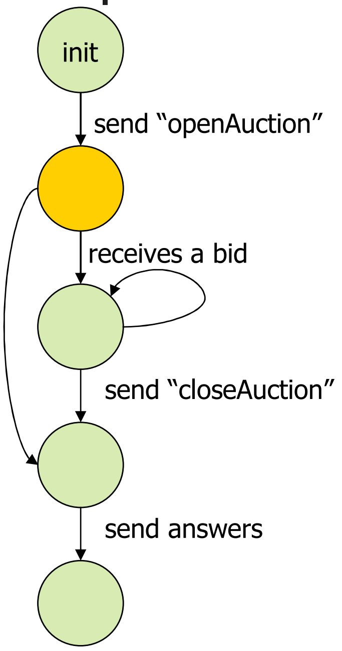
Defining a protocol
The auction example



It is possible to identify the following "steps" in the protocol:

1. The auctioneer opens the auction (by sending the message "openauction" to the invited bidders)
2. Each bidder places its bid
3. The auctioneer communicates the closing of the auction
4. The auctioneer communicates to each bid if it has won or it has lost.

Defining a protocol
The auction example

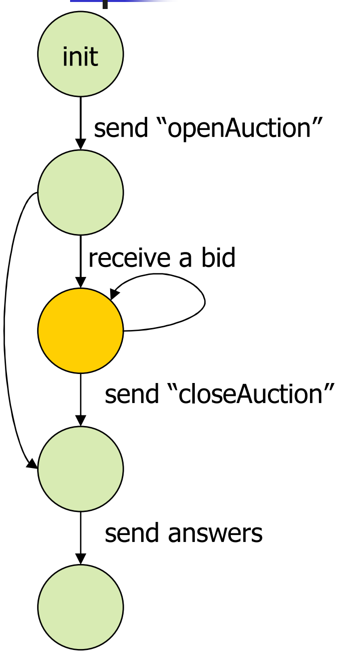


Step 1:

- No “preconditions” about the opening auction step.
- Only one future event required: the auctioneer will close the auction at a certain time.

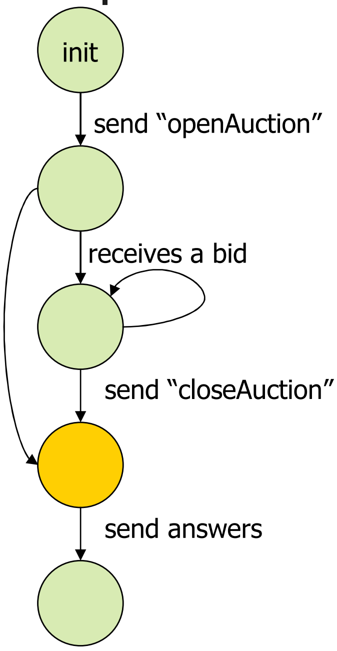
H(tell(A, B, openauction(Item,TEnd,TDeadline), D), TOpen)
--->
E(tell(A, B, closeauction, D), Tend)
/\ Tend > Topen.

Defining a protocol
The auction example



H(tell(A, B, openauction(Item,TEnd,TDeadline), D), TOpen)
/\ H(tell(B, A, bid(Item,Price), D), TBid)
/\ TOpen < TBid
/\ TOpen < TEnd
/\ TEnd < TDeadline
--->
E(tell(A, B, answer(win,Item,Price), D), TWin)
/\ TWin <= TDeadline
/\ TEnd < TWin
\/
E(tell(A, B, answer(lose,Item,Price), D), TLose)
/\ TLose <= TDeadline
/\ TEnd < TLose.

Defining a protocol
The auction example

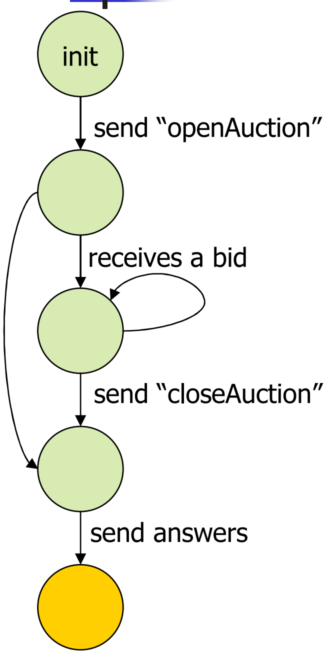


Step 3:

- In order to communicate the "closeAuction" message, the auctioneer should have opened the auction with a previous message

```
H( tell( A, B, closeauction, D), Tend)
--->
E( tell( A, B, openauction(Item,TEnd,TDeadline), D), TOpen)
/\ Tend > Topen.
```


Defining a protocol
The auction example




```
H( tell( A, B, answer(_,Item,Price), D), TAnswer)
--->
E( tell( B, A, bid(Item,Price), D), TBid)
/\ E( tell( A, _, openauction(Item,TEnd,TDeadline), D), TOpen)
/\ TOpen < TBid
/\ TBid < TEnd
/\ TEnd < TDeadline
/\ TAnswer <= TDeadline.
```

- No future consequence if you conclude the protocol here…

Checking the defined protocol

Checking compliant histories

- Once the protocol has been defined, a first test can be conducted by simulating correct agent interactions and observing the answer given by SOCS-SI
- SOCS-SI can read interactions saved on files (a sort of log of the interactions)
- The interactions are represented in the form of exchanged messages
- A first "feedback" on the "quality" of the protocol defined consists in being assured that all the desired interactions are considered compliant w.r.t. the protocol definition

Checking the defined protocol

Checking compliant histories

```
tell([s0], auction1, federico, bidder1, openauction, [laptop,10,20],5).
tell([s0], auction1, federico, bidder2, openauction, [laptop,10,20],5).
tell([s0], auction1, federico, bidder3, openauction, [laptop,10,20],5).

tell([s0], auction1, bidder1, federico, bid,[laptop,100],6).
tell([s0], auction1, bidder3, federico, bid,[laptop,80],8).
tell([s0], auction1, bidder2, federico, bid,[laptop,120],9).

tell([s0], auction1, federico, bidder1, closeauction, [],10).
tell([s0], auction1, federico, bidder2, closeauction, [],10).
tell([s0], auction1, federico, bidder3, closeauction, [],10).

tell([s0], auction1, federico, bidder2, answer, [win,laptop,120], 12).
tell([s0], auction1, federico, bidder1, answer, [lose,laptop,100],13).
tell([s0], auction1, federico, bidder3, answer, [lose,laptop,80], 15).
```

Checking the defined protocol
Checking non-compliant histories

- A second "feedback" on the "quality" of the formalization of the protocol consists to verify that "wrong" interactions are detected as "violating" the protocol.
- Easy to do only for naive violations
- Useful anyway for verifying conjectures about the formalization of the protocol

Checking the defined protocol
Checking non-compliant histories

```
tell([s0], auction1, federico, bidder1, openauction, [laptop,10,20],5).
tell([s0], auction1, federico, bidder2, openauction, [laptop,10,20],5).
tell([s0], auction1, federico, bidder3, openauction, [laptop,10,20],5).

tell([s0], auction1, bidder1, federico, bid,[laptop,100],6).
tell([s0], auction1, bidder3, federico, bid,[laptop,80],8).
tell([s0], auction1, bidder2, federico, bid,[laptop,120],9).

tell([s0], auction1, federico, bidder1, closeauction, [],10).
tell([s0], auction1, federico, bidder2, closeauction, [],10).
tell([s0], auction1, federico, bidder3, closeauction, [],10).

tell([s0], auction1, federico, bidder2, answer, [lose,laptop,120], 12).
tell([s0], auction1, federico, bidder1, answer, [lose,laptop,100],13).
tell([s0], auction1, federico, bidder3, answer, [lose,laptop,80], 15).
```

Checking the defined protocol

Fixing the protocol?

- If at least one bid has been placed by an allowed bidder, then there must be at least one winning message

```
H( tell( A, B, openauction(Item,TEnd,TDeadline), D), TOpen)
/\ H( tell( B, A, bid(Item,Price), D), TBid)
/\ TOpen < TBid
/\ TOpen < TEnd
/\ TEnd < TDeadline
--->
E( tell( A, _, answer(win,_,_), D), TWin)
/\ TWin <= TDeadline
/\ TEnd < TWin.
```


Proving properties

The generative $\mathcal{S}$CIFF

- We are developing a version of the $\mathcal{S}$CIFF proof procedure that is able to disprove a property w.r.t a given protocol specification.
- It is still a work in progress; some results have been presented in a previous talk
- Given a property P and a protocol Q, if we are able to generate a history that is compliant with $(Q \cup \neg P)$, hence P does not hold.

Conclusions

- Abductive Framework *S*CIFF:
  - Interaction Representation
  - Protocol Definition Language (ICs)
- SOCS-SI
  - Verify if an interaction is compliant with a given protocol definition
  - Can be used for simulating the defined protocol
- Protocol properties

Links

- SOCS Project:
  http://lia.deis.unibo.it/SOCS/
- SCIFF proof procedure
  http://lia.deis.unibo.it/Research/sciff/
- Acknowledgments:

This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 project in the context of the Global Computing initiative of the FET (Future Emerging Technology) initiative

Bibliography

- General Framework
  - Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), April 2004.
  - Marco Alberti, Anna Ciampolini, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. A social ACL semantics by deontic constraints. In Vladimir Marik, Jorg Muller, and Michal Pechoucek, editors, *Multi-Agent Systems and Applications III. 3rd International Central and Eastern European Conference on Multi-Agent Systems CEEMAS 2003*, volume 2691 of *LNAI*
  - Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The SOCS computational logic approach to the specification and verification of agent societies. In Corrado Priami and Paola Quaglia, editors, *Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers*, volume 3267 of *LNCS*
  - Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. A logic based approach to interaction design in open multi-agent systems. In Martin Fredriksson, Rune Gustavsson, Alessandro Ricci, and Andrea Omicini, editors, *13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004)*, pages 387-392, Washington, DC, USA, September 2004. IEEE Computer Society.
- Operational Semantics
  - Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Abduction with hypotheses confirmation. In Rossi and Panegai ed., CILC 2004
  - Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The SCIFF abductive proof-procedure. In Fabio Zanzotto, editor, *IX Congresso nazionale Associazione Italiana per l'Intelligenza Artificiale*, Lecture Notes in Artificial Intelligence, Berlin, 2005. Università degli studi di Milano Bicocca, Springer Verlag. to appear.
- Implementation
  - Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: a logic-based tool. In Robert Trappl, editor, *Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Vol. II, Symposium ``From Agent Theory to Agent Implementation'' (AT2AI-4)*, pages 570-575, Vienna, Austria, April 13-16 2004. Austrian Society for Cybernetic Studies.
- Applications
  - Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Modeling interactions using social integrity constraints: a resource sharing case study. In Joao Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies, First International Workshop, DALT 2003, Melbourne, Australia, July 15, 2003, Revised Selected and Invited Papers*, volume 2990 of *Lecture Notes in Computer Science*, pages 243-262, Melbourne, Australia, 2004. Springer Verlag.
  - Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Paola Mello, and Paolo Torroni. Expressing interaction in combinatorial auction through social integrity constraints. *Intelligenza Artificiale*, II(1):22-29, 2005.