Verifying Protocol Conformance on the fly by using SC-IFF proof procedure

Paola Mello, Federico Chesani

Universita` di Bologna

14 Luglio 2004

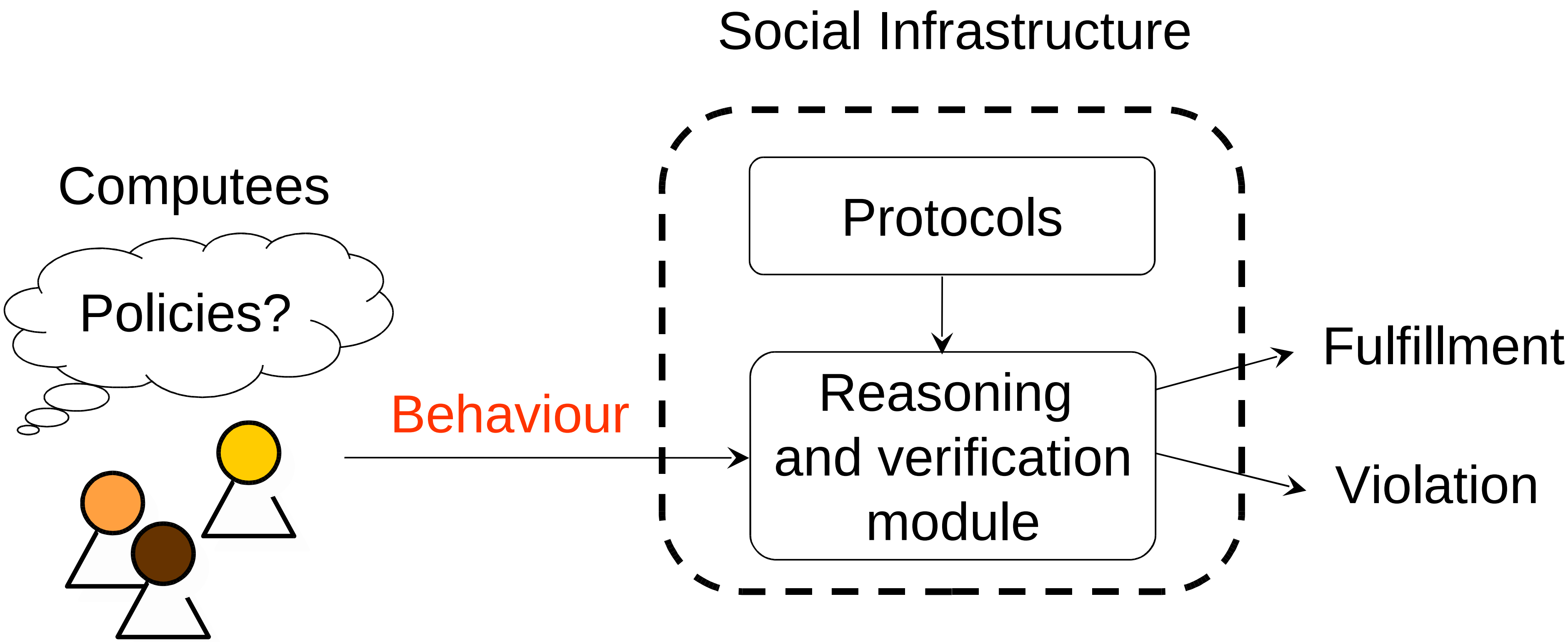
Open Societies of agents/computees

- Open societies
  - agents/computees are heterogeneous
  - no assumptions on the behaviour of agents
  - observation of external behaviour of agents (interactions)
- Interactions
  - agent communication language
  - interaction protocols
  - issues: formal specification, verification of compliance

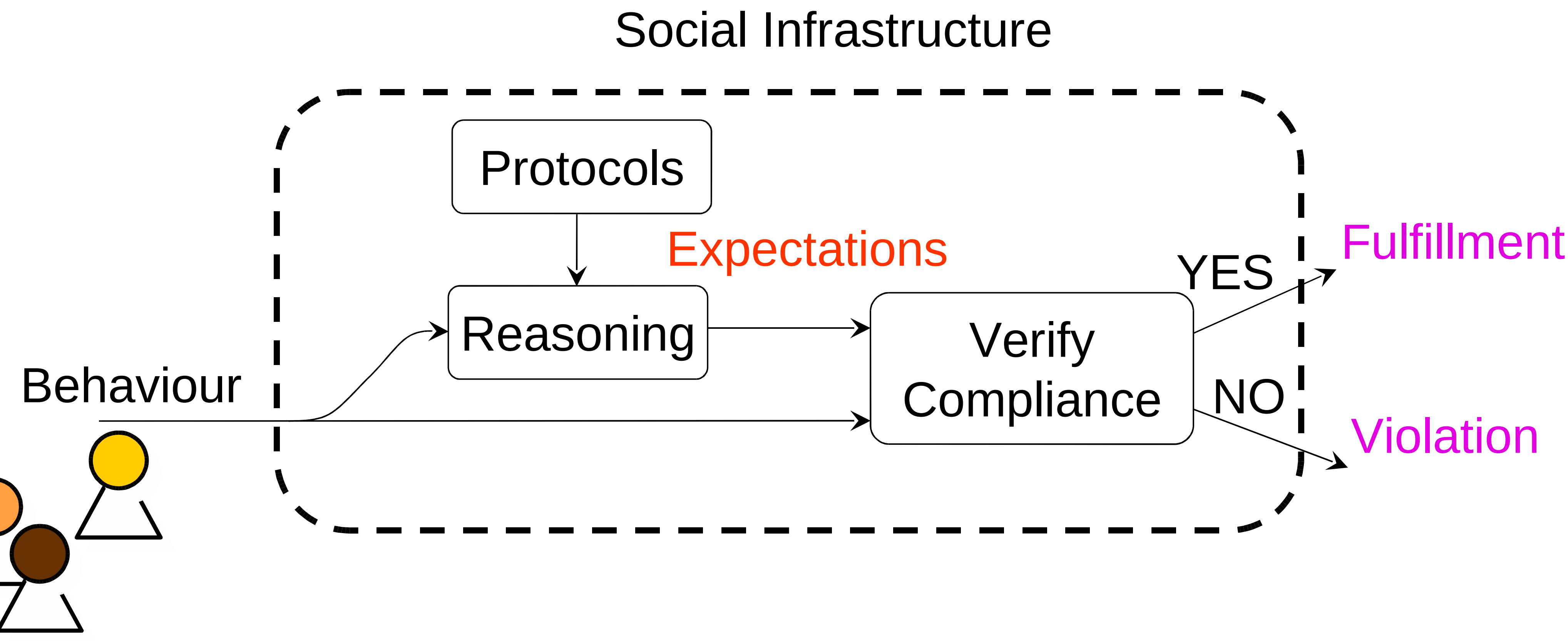
Aims of the Society model

- Use of a declarative and CL-based representation for the specification of ACL/protocols
- Uniform computational model to understand different aspects of interaction in an open and global environment
- Support goal-directed behaviour of societies
- Corresponding operational model
- Possibility to verify interactions, and prove properties

Compliance Verification



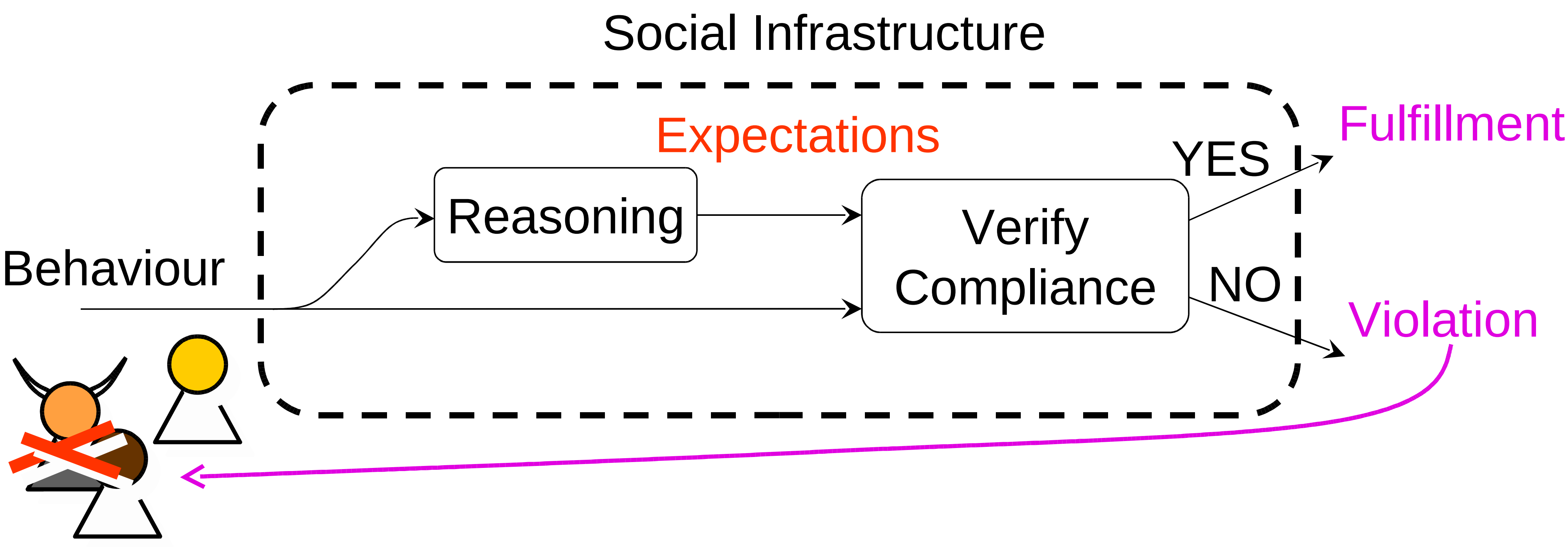
Social infrastructure



Knowledge Representation
<SOKB, SEKB, IC_s, Goals>

- **SOKB: society knowledge base (roles and rules)**
  - ***LP* clauses with expectations**
- **SEKB: consists of**
  - **History (HAP): set of *Happened* events that are "socially relevant" (e.g. communicative acts). *Ground facts of the kind:* H(*Event* [,*Time*]).**
  - **EXP: (positive or negative) expectations on the correct (social) behaviour of members: [¬] E(*Event* [, *Time*]) / [¬] EN(*Event* [, *Time*])**
- ***IC_s: protocol specification by means of integrity constraints (social semantics)***
- ***Goals:* Society can be goal-directed**
  - **LP Goals**

Social infrastructure



(1)on-the fly verification of compliance
to protocols

Social Integrity Constraints (ICs)

- **Example of Social Integrity Constraint**

  Society where agents can **exchange resources**:

  If I make you an offer, you are expected to answer to me by either accepting or refusing before a deadline d

  H(tell(Me,You,offer(Item,Price),T) $\rightarrow$

  E(tell(You,Me,accept(Item,Price),T'), T'<=T+d $\vee$

  E(tell(You,Me,refuse(Item,Price), T'), T'<=T+d

  If you accept my offer, you are expected to not refuse it later

  H(tell(You,Me,accept(Item,Price), T) $\rightarrow$

  EN(tell(You,Me,refuse(Item,Price), Tr), Tr>=T

Example (fulfilment)

yves

thomas

H(tell(yves,thomas,offer(scooter,10$),1)

E(tell(thomas,yves,accept(scooter,10$),T’), T’ < 7

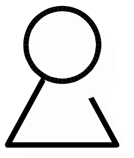
∨ **E**(tell(thomas,yves,refuse(scooter,10$),T’), T’ < 7

H(tell(thomas,yves,accept(scooter,10$),5)

fulfillment!

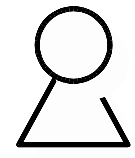
Example (violation)

yves



H(tell(yves,thomas,offer(scooter,10$),1)

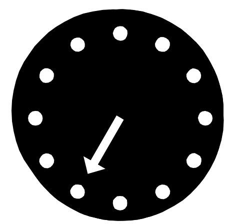
thomas



E(tell(thomas,yves,accept(scooter,10$),T’), T’ < 7

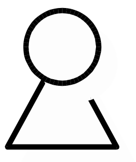
∨ **E**(tell(thomas,yves,refuse(scooter,10$),T’), T’ < 7

violation!

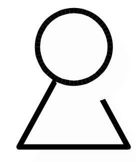


Example (violation)

yves



thomas



H(tell(yves,thomas,offer(scooter,10$),1)

H(tell(thomas,yves,**accept**(scooter,10$),5)

H(tell(thomas,yves,accept(Item,Price), T)

EN(tell(thomas,yves,refuse(Item,Price), Tr), Tr>=T

H(tell(thomas,yves,**refuse**(scooter,10$),8)

violation!

Society Instance as Abductive Logic Program

- Instance of a Society (S_{HAP}): ALP <P,Ab,IC> with
  - $P = SOKB \cup HAP$
  - $Ab = \{E, EN, \neg E, \neg EN\}$
  - $IC = IC_S$
- Consistency:
  - IC_S-Consistency
  - E-Consistency
  - ¬-Consistency
  - Fulfillment

Consistency

Given a society and a set HAP of events…

3. a set of expectations EXP is ***IC_s*-consistent** iff

$$\text{SOKB} \cup \text{HAP} \cup \text{EXP} \quad IC_s$$

4. a set of expectations EXP is **E-consistent** iff

$$\{\, E(p), EN(p) \,\} \not\subseteq \text{EXP}$$

5. a set of expectations EXP is ***¬-consistent*** iff

$$\{\, E(p), \neg E(p) \,\} \not\subseteq \text{EXP}$$

$$\{\, EN(p), \neg EN(p) \,\} \not\subseteq \text{EXP}$$

6. a (IC_s,E,¬) consistent EXP is ***fulfilled*** iff

$$\text{HAP} \cup \text{EXP}\ \{E(p) \rightarrow H(p)\} \cup \{EN(p) \rightarrow \neg H(p)\} \not\models \bot$$

7. if no consistent set of expectations fulfilled exists, HAP produces a **violation** in the society

Society Constraint Proof Procedure (wrt IFF)

Called SCIFF (Society Constraint IFF)

Extends IFF.

New features:

- Accepts new events as they happen (incremental, dynamic)
- Generates the expectations E, not E, EN, not EN on the basis of behaviour of the members of a society and of IC_s.
- Verifies the correspondence between the happened events and the expected events (fulfillment)
- Identifies (as soon as possible) the situation where there is violation and/or inconsistency
- The abduced atoms may have variables quantified $\forall$ and variables quantified $\exists$

Properties

- General properties (of the framework):
  - **well-definedness of programs/ICs**
  - **termination of conformance checking (link to structural properties of ICs)**
- Properties of interaction
  - **mechanism viewpoint:**
    - **“general” properties (fairness, termination, …)**
    - **“specific” (some proposition/formula holds): can be defined by way of ICs ???**
  - **agent viewpoint:** conformance **to protocols**

General Society Properties

- Termination of conformance checking
  - aim: to identify classes of Societies (SOKB, ICs, G) for which, under *suitable syntactic conditions*, any execution of SCIFF terminates, given each possible history
  - classes must be expressive enough to represent certain significant protocols, while guaranteeing termination of SCIFF.
- Well-definedness of Societies
  - aim: given a society (SOKB, ICs, G), to guarantee that, under certain hypotheses, this society will be well-defined, i.e., among its possible (closed) instances there exists at least one for which goal G is achieved.
- Idea: use background from LP, e.g.:
  - call-consistent normal LP => ∃ at least one (total) stable model
  - stratified normal LP => ∃ exactly one (total) stable model
  - acyclic ALP and query => termination

Examples

- **Not well-defined:**
  - Goal: g.
  - SOKB: g <- E(p), E(q).
  - ICS: E(p) -> EN(q).

  Or:
  - Goal: true.
  - ICS: H(p) -> EN(p).
    Not H(p) -> E(p).

- **Not terminating:**
  - ICS: E(p(X)) -> E(p(f(X))).

Agent Compliance (wrt ICs), [Alberti et alii ENTCS2003]

- **Negative compliance**: A group of agents is negatively compliant to a set of social integrity constraints *iff* its members never produce a social event which is expected not to happen;
- **Positive compliance**: A group of agents is positively compliant to a set of social integrity constraints *iff* its members never fail to produce social events which are expected to happen;
- **Strong compliance**: A group of agents is strongly compliant to a set of social integrity constraints *iff* it is both negatively and positively compliant.

- *Compliance* is a property which can be verified *on-the-fly* (in open environments)

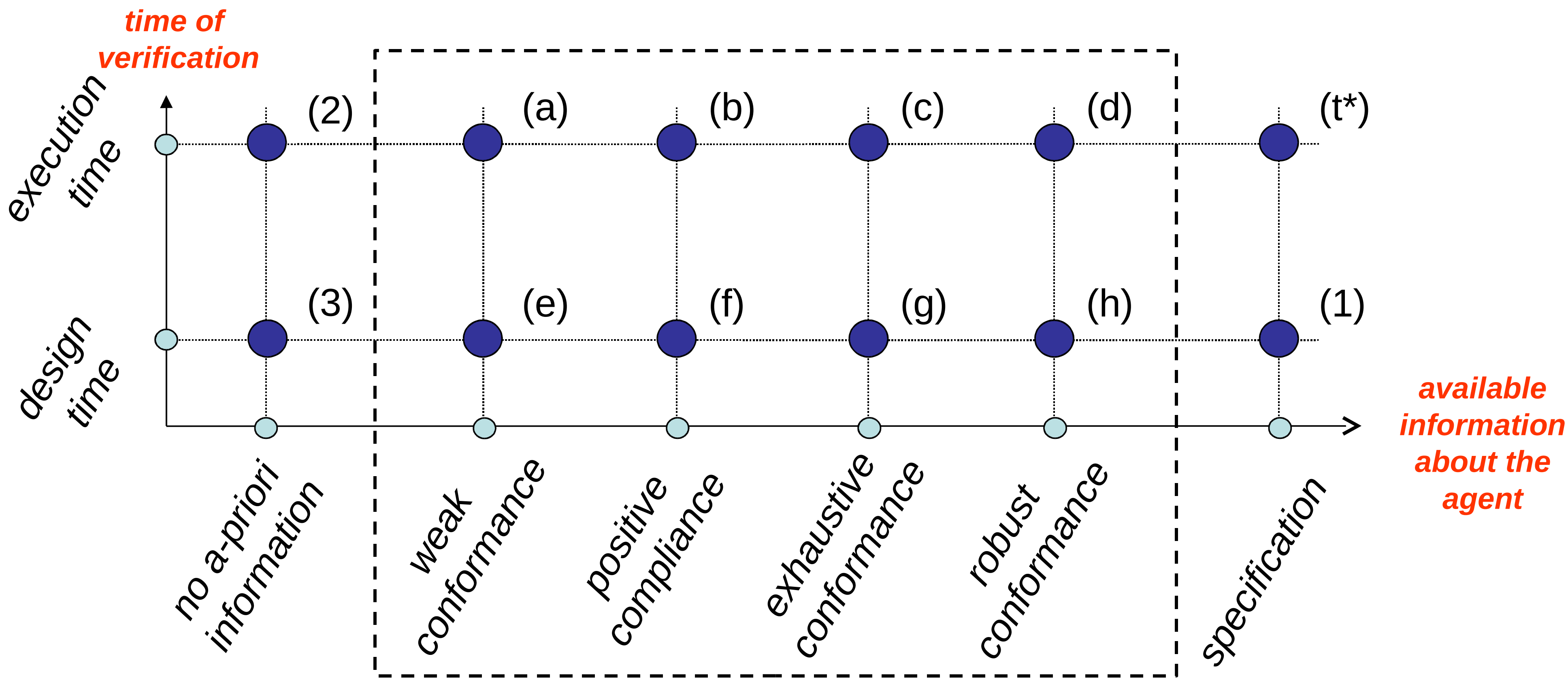
Agent Conformance (wrt Protocol) [Endriss et alii, AAMAS03]

- **Weak conformance** An agent is weakly conformant to a protocol P *iff* it never utters any illegal dialogue move (wrt. P)(negative?).

- **Exhaustive conformance** An agent is exhaustively conformant to a protocol P *iff* it is weakly conformant to P and it will utter at least one legal output move for any legal input of P it receives (strong?).

- An additional notion of conformance (*robust* conformance) is related to the behaviour of computees in the event of illegal incoming messages (FIPA not-understood).

Degrees of verification



Proving specific properties

- Related work: model checking.
  - Prove static properties on mechanism
  - E.g. Needham-Schroeder: protocol is prone to man-in-the-middle breaks
  - Need full knowledge about the mechanism
- Using SCIFF (open questions):
  - how to use model checking techniques in an open environment (partial information)?
  - how to extend SCIFF to achieve results comparable with those achieved by model checking techniques? (possible?)
  - idea: can SCIFF be used to generate compliant *histories* of events? (which formalism to synthesize compliant histories?)

Collaboration?

- **Torino: Logic-based protocols & DyLOG**
  - mentalistic approach (agents) vs. social approach (Society infrastructure)
  - conformance:
    - investigate on the relationship between the different notions of conformance/compliance
  - translation AUML-> DyLog:
    - It would be interesting to investigate if the approach used for the translation AUML ->DyLog could be extended for translating AUML specification into ICS
  - from the implementation point of view: integration of DyLog and SOCS-SI

Ambito Applicativo

- **Analizzare e validare** i modelli e i linguaggi definiti in diversi scenari applicativi.

- Realizzazione di interazioni tra agenti basate su **dialoghi**, con particolare attenzione alla **negoziazione** per **l'allocazione di risorse**

- Esempio di test in SOCS: aste combinatorie, Net-Bill

- Sistemi di supporto alla **diagnosi medica** e di **verifica di protocolli** in campo medico (**linee guida**).

Demo!

- **A simplified auction scenario will be presented.**
- **Auction protocol is defined thorugh ICs. Amongst them:**

(1) If a computee makes a bid, the auctioneer is expected to answer by either saying win or lose before a deadline d

$$\mathbf{H}(tell(Bidder,Auctioneer,bid(Item,Price),T) \rightarrow$$
$$\mathbf{E}(tell(Auctioneer,Bidder,win(Item,Price),T'), T' \leq T+d \vee$$
$$\mathbf{E}(tell(Auctioneer,Bidder,lose(Item,Price), T'), T' \leq T+d$$

(2) Once the auctioneer awards a bidder ("win"), the auctioneer is expected not to acknowledge the same bidder with "lose"

$$\mathbf{H}(tell(Auctioneer,Bidder,win(Item,Price), T) \rightarrow$$
$$\mathbf{EN}(tell(Auctioneer,Bidder,lose(Item,Price), Tr), Tr \geq T$$

Demo!

- In the first run, a “compliance” history is shown.
- Agent *f* open an auction in order to get a taxi to a station
- Three taxi (*taxi1*, *taxi2*, *taxi3*) make a bid each.
- *f* notifies *taxi1* “win”, and to *taxi2* and *taxi3* it notifies “lose”

Demo!

- **In the second run, a “wrong” interaction is shown.**
- **Same scenario as before, but this time the auctioneer *f* does not notify *taxi2* and *taxi3* “lose”**
- **As soon as the history is declared “closed” (no more events can happen anymore), SOCS-SI detects the violation due to the ICs:**

H(tell(Bidder,Auctioneer,bid(Item,Price),T) $\rightarrow$

E(tell(Auctioneer,Bidder,**win**(Item,Price),T’), T’$\leq$T+d

$\vee$

E(tell(Auctioneer,Bidder,**lose**(Item,Price), T’), T’ $\leq$T+d

Goal Achievability

- **Society can be goal-directed**
- **Given an instance of a society S_{HAP} with (open/closed) history, a goal G is achievable in S_{HAP} iff there exists an (open/closed) consistent fulfilled set of expectations EXP s.t.:**

$$SOKB \cup \mathbf{EXP} \cup \mathbf{HAP} \models G$$

- **We write:**

$$S_{HAP} \mid\approx_{\mathbf{EXP}} G$$

i.e.,

$$Comp(SOKB \cup \mathbf{EXP}) \cup \mathbf{HAP} \cup CET \models G$$

Operational Semantics

- Based on IFF
- Data structure

$$T = \langle R, CS, PSIC, EXP, HAP, FULF, VIOL \rangle$$

Where

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

Derivation

- **Initial Node**

$$T_0 = <\{G\}, \emptyset, IC_s, \emptyset, S_{\text{HAP}i}, \emptyset, \emptyset>$$

(may start with a non-empty history **HAP**$_i$)

- **Derivation $T_0 \rightarrow T_1 \rightarrow \ldots \rightarrow T_n$ (quiescence)**
- **Successful derivation:**
  - **Final node: $T_n = <\emptyset, CS, PSIC, \text{EXP}, \text{HAP}_f, FULF, \emptyset>$**
  - **written $S_{\text{HAP}i} \mid\!\sim_{\text{EXP} \cup \text{FULF}}^{\text{HAP}f} G$**

Transitions

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

Soundness

- **Soundness**. Given a society instance S_{HAPi}, if

$$S_{\text{HAP}i} \mid\!\sim_{\text{EXP} \cup \text{ FULF}}^{\text{HAP}f} G$$

with expectation answer (EXP $\cup$ FULF,σ) then

$$S_{\text{HAP}f} \mid\!\approx_{(\text{EXP} \cup \text{ FULF})\sigma} G\sigma$$

- Proved under
  - allowedness conditions
  - without abducibles quantified $\forall$
  - Extendable to quantified $\forall$ (sketched, see lemma to abducibles quantified $\forall$)

Soundness: scheme of proof

- Based on Soundness of IFF
  - In both cases of *open* and *closed* history

Prolog/CHR-based Implementation of SCIFF

- (Attributed) Variables
  - Quantification (exist, forall) in attributes
  - Ad-hoc constraint solver for unification
- Implementation of Data Structures
  - R as the Prolog resolvent
  - CS as CLP stores (CLPFD, CLPB)
  - PSIC,EXP,HAP,FULF,VIOL implemented as CHR constraints
- Implementation of Transitions
  - Most (propagation, fulfillment/violation, consistency…) as CHR rules
  - CLP: delegated to the solvers