Multi-Agent Programming

Brian Logan[1]

School of Computer Science
University of Nottingham

Midlands Graduate School
8th – 12th April 2013

[1] Slides on Normative Organisations are from an AAMAS 2012 tutorial on *Logics and Multi-agent Programming Languages* given jointly with Natasha Alechina, Nils Bulling and Mehdi Dastani

Course Overview

Lecture 1: *Programming agents*
BDI model; PRS and other BDI languages

Lecture 2: *Programming multi-agent systems*
Coordination in MAS; agent communication languages & protocols; programming with obligations and prohibitions

Lecture 3: *Logics for MAS*
LTL, CTL; Rao and Georgeff's BDI logics; Coalition Logic, ATL

Lecture 4: *Verification of MAS*
A tractable APL and BDI logic: SimpleAPL and PDL-APL

Lecture 2: Programming Multi-Agent Systems

Outline of this lecture

- coordination in multi-agent systems
- coordination in MAS composed of benevolent agents
    - agent communication languages and protocols
- coordination in MAS composed of self-interested agents
    - normative organisations, obligations and prohibitions

Multi-agent systems

Multi-agent systems are a promising approach to constructing complex software systems which are:

- **Open:** agents dynamically enter and exit the system
- **Autonomous:** agents pursue their own objectives
- **Encapsulated:** internal state and operation of agents is not visible to other agents (or the MAS)
- **Heterogeneous:** agents can have different capabilities and be implemented in different ways (e.g., different agent programming languages)

Applications of multi-agent systems

distributed problem solving

- each agent has only restricted capabilities or knowledge in relation to the (shared) problem to be solved
- e.g., scheduling meetings, design of industrial products

solving distributed problems

- the agents have similar capabilities but the problem is distributed
- e.g., controlling a communications or energy distribution network

What is a multi-agent system?

- a multi-agent system is a system in which there are several agents situated in the same environment which cooperate at least part of the time
- cooperation can either be implicit (e.g., emergent) or explicit
- most forms of explicit cooperation require some kind of communication between the agents
- more complex forms of cooperation often require additional components as part of the MAS to coordinate the behaviour of the agents

Interactions in multi-agent systems

- if the agents are not aware of or simply ignore each other, there isn't very much interesting to say
- if they always *compete* with each other, it is more interesting, but the agents don't form a system in anything other than the ecological sense (e.g., artificial life)
- for a multiagent system to be possible the agents must *cooperate* about some things
- e.g., even if the agents compete for resources, they must cooperate about how the resources are to be allocated

Competition and cooperation in MAS

- the balance between competition and cooperation depends on the degree to which the goals of the agents overlap
- e.g., agents representing different organisations in an electronic market will typically have competing goals (to maximise the profit of their organisation)
- however they must cooperate to ensure that the market (e.g., auction) works effectively
- *mechanism design* is concerned with designing interaction protocols in which the agents have no incentive not to cooperate

Benevolent vs. self-interested agents

benevolent agents implicitly or explicitly share one or more *common (system or organisational) goals*

- e.g., when the agents are 'owned' by the same organisation or individual
- agents work to achieve the overall objectives of the system, even when these conflict with the agent's own goals

self-interested agents do not share a common goal

- e.g., they are designed to represent the interests of different organisations or individuals
- agents co-operate because it helps them achieve their own goals

Benevolent agents

- all the agents in the MAS cooperate to achieve one or more system or organisational goals
- the agents co-operate to perform some task that a single agent can't do on its own
  - because a single agent doesn't have all the capabilities or knowledge required to perform the task
  - because a single agent would be too slow
- note that there may still be elements of competition, e.g., if the agents compete for the organisation's resources
- mechanisms are still required to ensure that resources and tasks are allocated appropriately

Coordination in multiagent systems

- the overall objective of a multiagent system can be achieved by coordinating (regulating) the observable/external behaviour of the agents
- many agent-oriented programming languages and platforms (e.g., Jason, 2APL) support
  - instantiation of multiple agents
  - constructs to implement basic ('cooperative') coordination, e.g., communication, access to shared resources or environments, task allocation etc.
- implementing coordination mechanisms for more open, less 'cooperative' MAS requires additional languages or components

Programming coordination in MAS

- several approaches in the literature:
  - languages and artefacts defined in terms of coordination concepts such as synchronization, shared-space, channels, sensing, e.g., Linda, CARTAGO, ReSpecT, EIS, etc.
  - organizational models, normative systems, and electronic institutions defined in terms of social and organisational concepts, e.g., ISLANDER/AMELI, PowerJava, $\mathcal{M}$oise^{+}, 2OPL, etc.
- in addition to programming individual agents, to implement a MAS, we need to be able to program such coordination mechanism(s)

Agent communication languages & protocols

Communication in multiagent systems

- an important strand of work in multi-agent programming (and logics for MAS) is the design and analysis of *agent communication languages*
- most agent communication languages are based on a very simplified notion of speech acts
- ACLs typically define a set of *performatives* (`tell`, `ask` etc.) and their syntax
- examples:

  KQML (Knowledge Query and Manipulation Language) — DARPA, 1990s

  FIPA (Foundation for Intelligent Physical Agents) ACL began in 1995, standardised in 1999

KQML

- defines format of messages
- 41 performatives or message types, e.g., `ask-if` and `tell`
- other components of a message are for example ontology (for the terminology used)
- does not define content
- has semantics (pre- and postconditions, in a language with belief, knowledge, wanting and intending modalities) by Labrou and Finin, IJCAI'97
- KQML criticised by Cohen and Levesque for lacking message types to express commitment

FIPA ACL

- similar to KQML
- 20 performatives: for example, `agree`, `cancel`, `confirm`, `disconfirm`, `inform`, `not-understood`, `query-if`, `refuse`, `accept-proposal`, `reject-proposal`, `request`...
- also has formal semantics in multi-modal logic (based on Cohen and Levesque, see Bretier and Sadek in LNAI 1193)

Contract net protocol

The contract net protocol is a way of achieving efficient co-operation through task sharing in networks of (possibly heterogeneous, autonomous) agents

- **task announcement:** an agent which generates (or receives) a task broadcasts a description of the task to some or all of the agents
- **bid response:** agents respond to the task announcement with a bid
- **task allocation:** the agent which announced the task allocates it to one or more of the bidding agents
- **expediting:** the agent to which the task was allocated carries it out

Task announcement

- task manager sends a task announcement to some or all agents
- task announcement contains information about the task to be performed:
  - *eligibility specification:* the criteria an agent must meet in order to be eligible to submit a bid
  - *task abstraction:* brief description of the task to allow potential bidders to evaluate level of interest
  - *bid specification:* description of the expected form of a bid for the announced task

Bidding

- on receipt of a task announcement, an agent determines if it is eligible for the task based on:
  - the task's eligibility specification
  - the agent's hardware and software resources
  - its current commitments
- eligible agents send a bid to the task manager containing the information in the bid specification, e.g., when they will be able to complete the task, how much it will cost, etc.

Task allocation

- bids are stored by the task manager until a deadline is reached
- if no (acceptable) bids are received by the deadline, task is re-announced
- otherwise the manager then awards the task to one or more bidders
- bidders who have be awarded the task confirm that they are still able to undertake it (situation may have changed between bid and award)
- otherwise part or all of the task is re-announced

Task processing

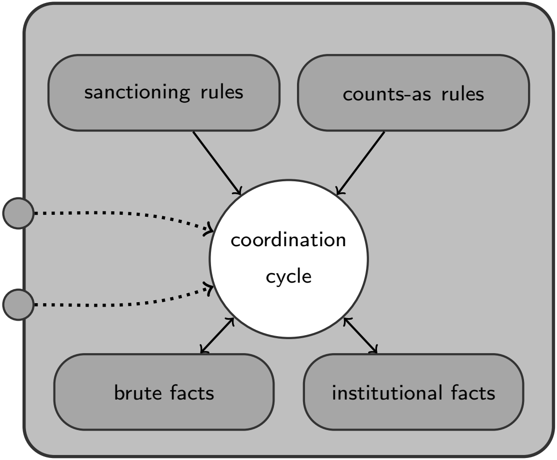
- award messages contain a complete specification of the task to be executed
- successful bidder(s) (contractors) *must attempt to expedite the task*
- this may result in the generation of new sub-tasks which the bidder then manages . . .
- when the task is complete, contractors send their manager a report message containing the result of the task

Applications

- contract net has become one of the most popular frameworks for task sharing in multi-agent systems (e.g., FIPA-OS)
  - originally used to allocate tasks over a distribute network of sensors (benevolent agents)
  - later extended to self-interested agents in electronic markets
- many variants — e.g., agents respond with offers of tasks to swap for the announced task

Normative organisations

Counts-as & sanctioning rules



- brute facts model the domain specific state
- agents modify brute facts by performing actions
- brute state is normatively assessed by counts-as rules
- counts-as rules link brute state to institutional facts
- normative judgments and role enactments constitute institutional facts
- judgment might lead to sanctions (punishments and rewards)
- coordination cycle determines order in which constructs are applied

Normative multi-agent organisation example

- to program a normative organisation we must specify: the roles that can be played by agents in the organisation, the initial brute state and the effects of actions on the brute facts
- e.g. a program for a simplified implementation of a conference management system could be:

Roles: `chair, reviewer, author`

Brute Facts: `phase(closed)`

Effect Rules:

```
{rea(C,chair), phase(closed)}
  open(C)
{not phase(closed), phase(abstracts)}
  ⋮
{rea(R,reviewer), phase(review), assigned(R,P)}
  uploadReview(R,P)
{review(R,P)}
```

Normative multi-agent organisation example (2)

- we also need to specify how the brute state gives rise to institutional facts, and the effects of normative judgements
- counts-as rules normatively judge the brute state, i.e., they produce normative or institutional facts, e.g.:

```
{paper(A,PId), pages(PId) > 15}
  =>
{viol(PId,pagelimit)}
```

- sanctioning rules specify the consequences for the brute state of a particular normative assessment, e.g.:

```
{viol(PId,pagelimit)}
  =>
{rejected(PId)}
```

Normative multi-agent organisation example (3)

Roles: `chair, reviewer, author`

Brute Facts: `phase(closed)`

Effect Rules:

```
{rea(C,chair), phase(closed)}
  open(C)
{not phase(closed), phase(abstracts)}
    .
    .
    .
{rea(R,reviewer), phase(review), assigned(R,P)}
uploadReview(R,P)
{review(R,P)}
```

Counts-As Rules:

```
{paper(A,PId), pages(PId) > 15}
 =>
{viol(PId,pagelimit)}
```

Sanctioning Rules:

```
{viol(PId,pagelimit)}
 =>
{rejected(PId)}
```

Problems with counts-as rules

- despite (or perhaps because of) their simplicity counts-as rules have some limitations when it comes to expressing norms:
- difficult to handle conditional and temporal aspects, e.g., *when you validate your ticket, you have 55 minutes to complete your journey*
- obligations and prohibitions are not explicitly specified:
  - obfuscates the meaning of program code
  - obligations and prohibitions cannot be communicated to (norm-aware) agents
- an alternative approach is to use conditional norms:
  - obligations and prohibitions are explicit
  - we can express conditions and deadlines

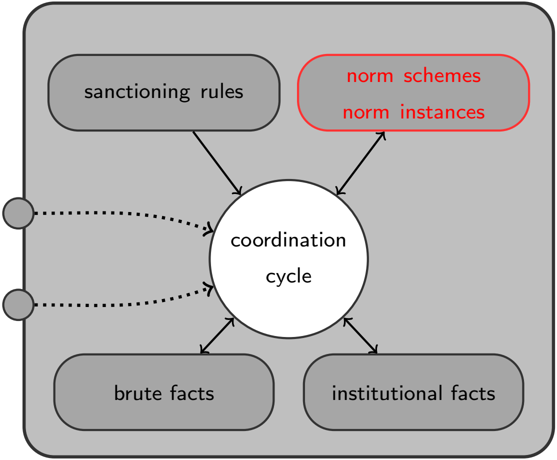
Counts-as rules vs. conditional norms

- the normative component of an organisation is typically about specifying obligations, prohibitions and permissions
- counts-as rules can only express obligations, prohibitions and permissions *implicitly*

Op	a state in which p does *not* hold necessarily counts as a violation
Fp	a state in which p holds necessarily counts as a violation
Pp	a state in which p holds *not* necessarily counts as a violation

- which obligations, prohibitions an permissions w.r.t. q can we "derive from" the counts-as rule `{not q} => {viol}`?

Conditional norms



- brute facts model the domain specific state
- agents modify brute facts by performing actions
- ideal brute state described by norm schemes (conditional obligations and prohibitions)
- norm schemes instantiate norm instances (detached obligations and prohibitions);
- violations and role enactment represented by institutional facts
- norm violation may lead to sanctions;
- coordination cycle determines order in which constructs are applied

Example norm schemes

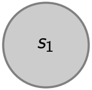
Norms:

```
reviewdue(R):                                   % label
< phase(review) and assigned(R,P),              % condition
  O(review(R,P)),                               % obligation
  phase(collect)>                               % deadline

minreviews(P):
< phase(submission) and paper(P),
  O(nrReviews(P) >= 3),
  phase(collect)>

pagelimit(PId):
< phase(submission) and abstract(A,PId),
  F(pages(PId) > 15),
  phase(review)>
```

Evolution of Obligations 1

Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```

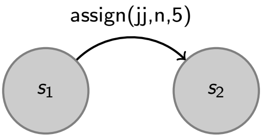


brute facts	:	`{ phase(review), paper(5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ }`

Evolution of Obligations 2

Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```

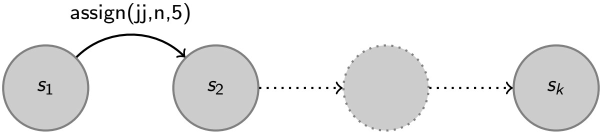


brute facts	:	`{ phase(review), paper(5), assigned(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ (reviewdue(n),O(review(n,5),phase(collect)) }`

Evolution of Obligations 3

Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```

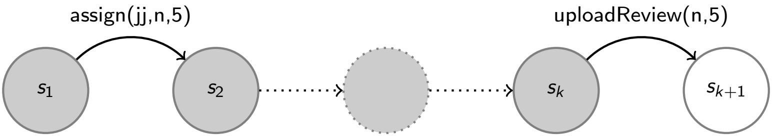


brute facts	:	`{ phase(review), paper(5), assigned(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ (reviewdue(n),O(review(n,5),phase(collect)) }`

Evolution of Obligations 4

Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```

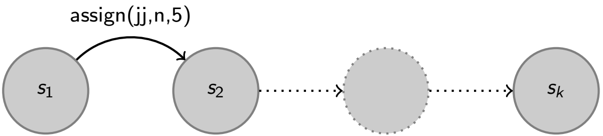


brute facts	:	`{ phase(review), paper(5), assigned(n,5), review(n,5) }`
inst. facts	:	`{ rea(jj,chair), obey(reviewdue(n)) }`
instances	:	`{ }`

Evolution of Obligations 3 (alternative history)

Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```

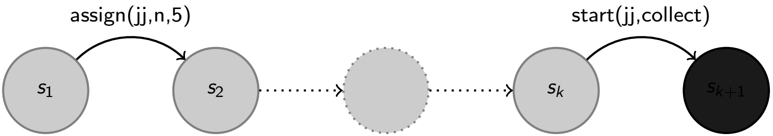


brute facts	:	`{ phase(review), paper(5), assigned(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ (reviewdue(n),O(review(n,5),phase(collect)) }`

Evolution of Obligations 4a

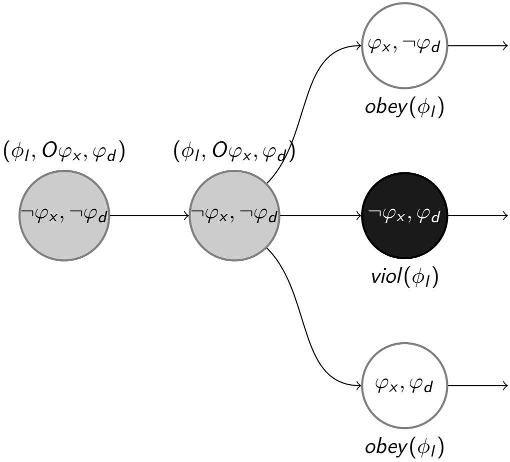
Recall norm scheme:

```
reviewdue(R):
< phase(review) and assigned(R,P), O(review(R,P)), phase(collect) >
```



brute facts	:	`{ phase(collect), paper(5), assigned(n,5) }`
inst. facts	:	`{ rea(jj,chair), viol(reviewdue(n)) }`
instances	:	`{ }`

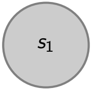
Behavior of an Obligation Summarized



Evolution of Prohibitions 1

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

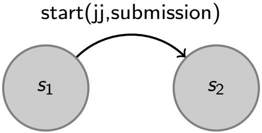


brute facts	:	`{ phase(abstract), abstract(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ }`

Evolution of Prohibitions 2

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

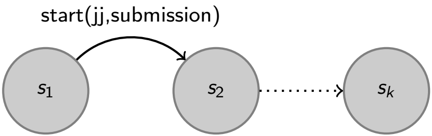


brute facts	:	`{ phase(submission), abstract(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ (pagelimit(5), F(pages(5) > 15, phase(review)) }`

Evolution of Prohibitions 3

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

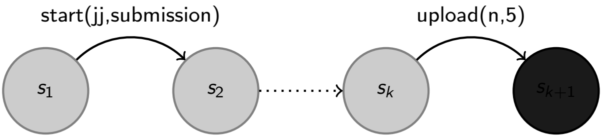


brute facts : `{ phase(submission), abstract(n,5) }`
inst. facts : `{ rea(jj,chair) }`
instances : `{ (pagelimit(5), F(pages(5) > 15, phase(review)) }`

Evolution of Prohibitions 4

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

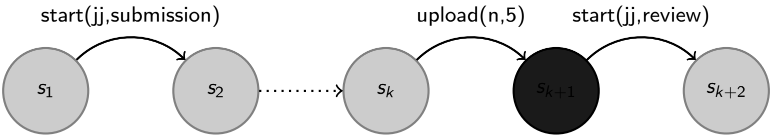


brute facts	:	`{ phase(submission), abstract(n,5), paper(5), pages(5) = 17 }`
inst. facts	:	`{ rea(jj,chair), viol(pagelimit(5)) }`
instances	:	`{ (pagelimit(5), F(pages(5) > 15, phase(review)) }`

Evolution of Prohibitions 5

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

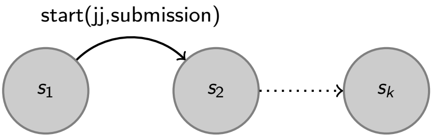


brute facts	:	`{ phase(review), abstract(n,5), paper(5), pages(5) = 17 }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ }`

Evolution of Prohibitions 3 (alternative history)

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

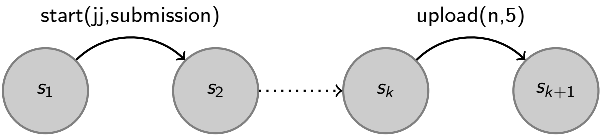


brute facts	:	`{ phase(submission), abstract(n,5) }`
inst. facts	:	`{ rea(jj,chair) }`
instances	:	`{ (pagelimit(5), F(pages(5) > 15, phase(review)) }`

Evolution of Prohibitions 4a

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

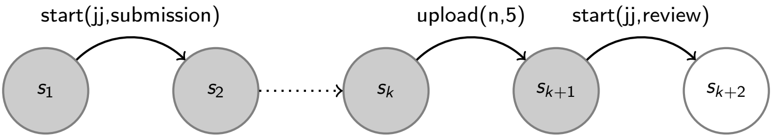


brute facts	:	`{ phase(submission), abstract(n,5), paper(5), pages(5) = 15 }`
inst. facts	:	`{ rea(jj,chair), viol(pagelimit(5)) }`
instances	:	`{ (pagelimit(5), F(pages(5) > 15, phase(review)) }`

Evolution of Prohibitions 5a

Recall norm scheme:

```
pagelimit(P):
< phase(submission) and abstract(A,PId), F(pages(PId) > 15), phase(review)>
```

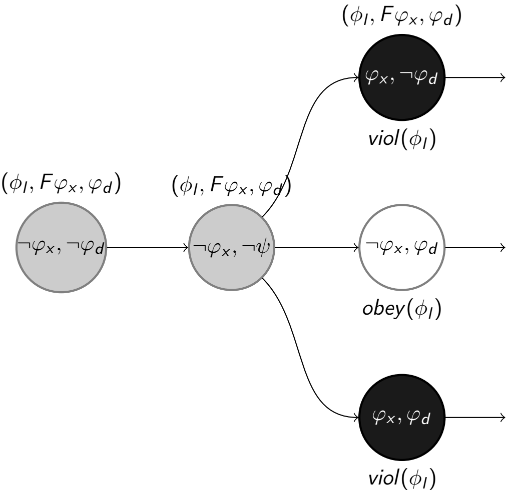


brute facts : `{ phase(review), abstract(n,5), paper(5), pages(5) = 15 }`

inst. facts : `{ rea(jj,chair), obey(pagelimit(5)) }`

instances : `{ }`

Behavior of a Prohibition Summarized



Norm-aware agents

- explicit representation of obligations and prohibitions makes it easier to develop *norm-aware* agents
- an agent is norm-aware if it can deliberate on its goals, norms and sanctions before deciding which plan to select and execute
- a norm-aware agent is able to (deliberately) *violate norms* (accepting the resulting sanctions) if it is in the agent's overall interests to do so
- e.g., if meeting an obligation would result in an important goal of the agent becoming unachievable

Summary

- coordination between benevolent agents can be programmed using ACLs and protocols defined in terms of speech acts
- only works if all agents (are programmed to) follow the protocol
- simple coordination for self-interested agents can be programmed using counts-as rules
- difficult to specify conditional obligations and prohibitions with deadlines using counts-as rules
- conditional norms are more complex, but allow conditional obligations and prohibitions with deadlines to be explicitly expressed

The next lecture

Logics for MAS