

(General issues in logics of agents, and)
Interpreted Dynamic Logic and Application to
Normative Systems

(part of a collaboration with
Andreas Herzig, Emiliano Lorini & Fred Moisan)

Logics for multi-agent systems

social sciences \Leftrightarrow computer science

- social software [Parikh 2001]: give to *societies of agents* what Turing's theory of computation is to computers.

Why?

- ← transform social sciences into formal sciences
- ⇒ use the insight of social sciences for distributed computing
 - negotiation
 - coordinated actions
 - knowledge, beliefs
 - preferences
 - obligations
 - ...

I am going to try to explain...

... various typical issues of **knowledge representation** and **logics of agents** more specifically:

- modelling a system (and trying to justify it)
- formalizing a property (and trying to justify it)
- characterizing a class of systems syntactically
- decision problems and their computational complexity
 - reasoning
 - (synthesising)
 - verifying

We are going to illustrate these issues with a formalism to reason about and verify social procedures involving interacting agents within a normative system.

Long term objective is to reason about **social reality** [Searle 1995].

Criteria of a “good” formal framework

Language:

- expressive
- intuitive
- succinct

Models:

- intuitive
- compact

Decision problems (model checking / satisfiability checking / synthesis)

- low complexity

A right balance needs to be found.

Our solution in dynamic logics

- We propose PDL^{\leftarrow} : an instantiation of propositional dynamic logic PDL [Harel et al. 2000].
- We step away from relational (possible worlds) semantics.
 - unpractical from a modelling perspective
- We adopt a “database perspective” [Shoham 2009].
 - **represents information**: a model is merely a list of what is true
 - **provides ‘services’ to store, retrieve and update information**: we have ‘exogenous’ rules of model update
- We illustrate it on:
 - capabilities
 - dynamics of permissions
 - constitutive rules

Outline

- 1 Basic logic
- 2 Capabilities and permissions
- 3 Constitutive rules

PDL[←]

$\mathbb{P} = \{p, q, \dots\}$ is a countable set of propositional variables. The language of PDL[←] is made up of events π and formulas φ and is defined by the following BNF:

$$\begin{aligned}\pi & ::= +p \mid -p \mid \pi; \pi \mid \pi \cup \pi \mid \pi^* \mid \varphi? \\ \varphi & ::= p \mid \top \mid \perp \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle \pi \rangle \varphi\end{aligned}$$

where p ranges over \mathbb{P} .

The set of all events is noted \mathcal{E} .

Some notation

Just as in PDL, $\pi_1; \pi_2$ is sequential composition, $\pi_1 \cup \pi_2$ is nondeterministic composition, π^* is iteration, and $\varphi?$ is the test of φ .

The formula $\langle \pi \rangle \varphi$ reads “there is an execution of π such that φ is true afterwards”.

The logical connectives \wedge , \rightarrow and \leftrightarrow have the usual meaning. The formula $[\pi]\varphi$ abbreviates $\neg\langle \pi \rangle\neg\varphi$. The event skip abbreviates $\top?$ (“nothing happens”).

First element of semantics: Assignments

The set of **assignments** is $\mathcal{A} = \{+p : p \in \mathbb{P}\} \cup \{-p : p \in \mathbb{P}\}$.

An assignment modifies the truth value of the propositional variable p :

- $+p$ sets p to true;
- $-p$ sets p to false.

The truth conditions are the customary ones for \top , \perp , negation and disjunction. The dynamic operators are interpreted by means of an update relation on valuations.

$$V \models p \quad \text{iff} \quad p \in V$$

$$V \models \langle \pi \rangle \varphi \quad \text{iff} \quad \text{there is a } V' \text{ such that } VR_{\pi}V' \text{ and } V' \models \varphi$$

R_{π} is a binary relation on valuations that allows to interpret events and that is defined together with \models by mutual recursion:

$$R_{+p} = \{(V, V \cup \{p\}) : V \text{ is a valuation}\}$$

$$R_{-p} = \{(V, V \setminus \{p\}) : V \text{ is a valuation}\}$$

$$R_{\pi_1; \pi_2} = R_{\pi_1} \circ R_{\pi_2}$$

$$R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$$

$$R_{\pi^*} = (R_{\pi})^*$$

$$R_{\varphi?} = \{(V, V) : V \models \varphi\}$$

A formula φ is satisfiable if and only if $V \models \varphi$ for some valuation V , and φ is valid if and only if $\neg\varphi$ is not satisfiable.

Complexity

Theorem

The problem of PDL^{\leftarrow} model checking is PSPACE-complete.

Proof (PSPACE-hardness).

We can reduce QSAT. From,

$$\Phi = \exists x_1 \forall x_2 \cdots \exists x_{m-1} \forall x_m \varphi$$

let

$$\begin{aligned} \Phi^{PDL^{\leftarrow}} = & \langle +x_1 \cup -x_1 \rangle [+x_2 \cup -x_2] \cdots \\ & \langle +x_{m-1} \cup -x_{m-1} \rangle [+x_m \cup -x_m] \varphi \end{aligned}$$

and (arbitrarily) $V = \emptyset$.

We can check that Φ is true iff $V \models \Phi^{PDL^{\leftarrow}}$; and V and $\Phi^{PDL^{\leftarrow}}$ are of size polynomial wrt. the size of Φ . □

Proof (PSPACE-easiness).

Every possible update of V by a complex event π can be reached by a sequence that is admitted by π and that is of length at most exponential in the length of π .

Based on that one can prove that membership of a couple of valuations (V, V') in R_π can be decided in polynomial space. Finally one can prove that a formula φ can be evaluated in space polynomial in the size of φ ; in particular, when evaluating $\langle \pi^* \rangle \varphi$ one may decide in polynomial space whether one of the (at most $2^{(|\pi^*|)^2}$) valuations V' is accessible from V . □

Theorem

The problem of PDL^{\leftarrow} satisfiability checking is PSPACE-complete.

Axiomatization

Prop. logic

(Prop) φ when φ is a propositional tautology

PDL[←] specif.

$\langle +p \rangle p$	\leftrightarrow	\top if $p \in \mathbb{P}$
$\langle -p \rangle p$	\leftrightarrow	\perp if $p \in \mathbb{P}$
$\langle \pm p \rangle q$	\leftrightarrow	q if $q, p \in \mathbb{P}$ and $q \neq p$
$\langle \pm p \rangle \top$	\leftrightarrow	\top
$\langle \pm p \rangle \neg \varphi$	\leftrightarrow	$\neg \langle \pm p \rangle \varphi$

PDL axioms

(K)	$\langle \pi \rangle (\varphi \vee \psi)$	\rightarrow	$\langle \pi \rangle \varphi \vee \langle \pi \rangle \psi$
(Union)	$\langle \pi \cup \pi' \rangle \varphi$	\leftrightarrow	$\langle \pi \rangle \varphi \vee \langle \pi' \rangle \varphi$
(Sequ)	$\langle \pi; \pi' \rangle \varphi$	\leftrightarrow	$\langle \pi \rangle \langle \pi' \rangle \varphi$
(Test)	$\langle \psi? \rangle \varphi$	\leftrightarrow	$\psi \wedge \varphi$
(LFP)	$\langle \pi^* \rangle \varphi$	\leftrightarrow	$\varphi \vee \langle \pi \rangle \langle \pi^* \rangle \varphi$
(Ind)	$\langle \pi^* \rangle \varphi$	\rightarrow	$\varphi \vee \langle \pi^* \rangle (\neg \varphi \wedge \langle \pi \rangle \varphi)$

Rules

(MP)	if $\vdash \varphi$ and $\vdash \varphi \rightarrow \psi$	then	$\vdash \psi$
(Nec)	if $\vdash \varphi$	then	$\vdash \langle \pi \rangle \varphi$

Outline

- 1 Basic logic
- 2 Capabilities and permissions**
- 3 Constitutive rules

Let us suppose that for every $p \in \mathbb{P}$ and $i \in \mathbb{A}$, the set \mathbb{P} contains the propositional variables

- $A_i(+p)$: “ i has the ability to make p true”;
- $A_i(-p)$: “ i has the ability to make p false”;
- $P_i(+p)$: “ i is authorized to make p true”;
- $P_i(-p)$: “ i is authorized to make p false”.

We apply these constructions recursively, considering that p itself may be a special propositional variable.

We can now express e.g. that i is allowed to enable j to set p to false, written $P_i(+A_j(-p))$.

Comparing literature: propositional control and transfer of control

A CL-PC formula $\diamond_J^A \varphi$ (see [van der Hoek and Wooldridge 2005]) can be translated into

$$\Phi(\varphi) \wedge \langle \text{skip} \cup (\bigvee_{i \in J} A_i(+p_1)?; +p_1) \cup (\bigvee_{i \in J} A_i(-p_1)?; -p_1) \rangle \dots \\ \langle \text{skip} \cup (\bigvee_{i \in J} A_i(+p_n)?; +p_n) \cup (\bigvee_{i \in J} A_i(-p_n)?; -p_n) \rangle \varphi$$

where $\mathbb{P}_\varphi = \{p_1, \dots, p_n\}$ and $\Phi(\varphi)$ describes some CL-PC constraints.

The logic DCL-PC extends CL-PC by delegation events $i \xrightarrow{p} j$ [van der Hoek et al. 2010] of i transferring to j his control over p . Such an event corresponds to the following event in PDL^\leftarrow :

$$A_i(+p)?; A_i(-p)?; \neg A_i(+p); \neg A_i(-p); +A_j(+p); +A_j(-p)$$

Example: Managing water resources

There are **two farmers** i_1 and i_2 working in a certain area close to a town called Thirstytown who need water in order to irrigate their fields. In this area there are **three different exploitable water basins** 1, 2 and 3. Only water basins 1 and 2 can be safely used by the farmers; basin 3 provides drinkable water to the population of Thirstytown, and if it is exploited for irrigation then Thirstytown will fall short of water. There are two other actors in this scenario: i_3 is the **chief of the Water Authority** which has the jurisdiction over the area, and i_4 is a **local policeman** working in Thirstytown.

Modelling the scenario

We start by giving a meaning to some proposition variables:

- for every propositional variables $\{p_1, p_2, p_3\}$:
 - p_h means that “the level of water in the basin h is high”
 - $\neg p_h$ means that “the level of water in the basin h is low”
- for every farmer $i_k \in \{i_1, i_2\}$ and for every propositional variable p_h with $h \in \{1, 2, 3\}$
 - $A_{i_k}(\neg p_h)$ expresses that basin h is physically connected to the field of farmer i_k so that i_k is able to exploit the water of the basin h ,
 - $P_{i_k}(\neg p_h)$ expresses that i_k is authorized to exploit the water of basin h .

We could present the scenario as model checking or **theorem proving**.

$$\varphi_1 = \bigwedge_{k \in \{1,2\}} (P_{i_k}(-p_1) \wedge P_{i_k}(-p_2) \wedge \neg P_{i_k}(-p_3))$$

$$\varphi_2 = A_{i_1}(-p_1) \wedge A_{i_1}(-p_2) \wedge \neg A_{i_1}(-p_3)$$

$$\varphi_3 = \neg A_{i_2}(-p_1) \wedge \neg A_{i_2}(-p_2) \wedge A_{i_2}(-p_3)$$

$$\varphi_4 = \bigwedge_{k \in \{1,2\}, h \in \{1,2,3\}} (A_{i_3} (+P_{i_4} (+A_{i_k} (-p_h))) \wedge A_{i_3} (+P_{i_4} (-A_{i_k} (-p_h))))$$

$$\varphi_5 = \bigwedge_{k \in \{1,2\}, h \in \{1,2,3\}} (P_{i_3} (+P_{i_4} (+A_{i_k} (-p_h))) \wedge P_{i_3} (+P_{i_4} (-A_{i_k} (-p_h))))$$

$$\varphi_6 = \bigwedge_{k \in \{1,2\}, h \in \{1,2,3\}} (A_{i_4} (+A_{i_k} (-p_h)) \wedge A_{i_4} (-A_{i_k} (-p_h)) \wedge \\ \neg P_{i_4} (+A_{i_k} (-p_h)) \wedge \neg P_{i_4} (-A_{i_k} (-p_h)))$$

Initial situation *INIT* = $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6$:

- 1 The farmers can legally only exploit basins 1 and 2 but not basin 3.
- 2 Farmer i_1 has access to basins 1 and 2, but not to basin 3.
- 3 Farmer i_2 has access to basin 3, but not to basin 1 and 2.
- 4 The chief of the Water Authority is *able* to authorize the policeman of Thirstytown to add and remove connections between basins and fields .
- 5 The chief of the Water Authority is *authorized* to authorize the policeman of Thirstytown to add and remove connections between basins and fields.
- 6 The policeman of Thirstytown is able to add and remove the connections between basins and fields, but is not authorized to.

The *GOAL* situation is defined by the conjunction of the following two facts:

- 1 Every farmer has exclusive control over at least one basin in $\{1, 2\}$: each farmer's field is connected to at least one water basin in $\{1, 2\}$ which is not connected to the other farmer's field. Formally:

$$\bigvee_{h \in \{1,2\}} (A_{i_1}(\neg p_h) \wedge \neg A_{i_2}(\neg p_h)) \wedge \bigvee_{h \in \{1,2\}} (A_{i_2}(\neg p_h) \wedge \neg A_{i_1}(\neg p_h))$$

- 2 No field is connected to basin 3: $\neg A_{i_1}(\neg p_3) \wedge \neg A_{i_2}(\neg p_3)$.

The problem to be solved is to **verify the executability** of some procedure such that, given the initial situation *INIT*, it ensures that the situation *GOAL* will be achieved.

Let $\pi_{i,h} = +P_{i_4}(+A_i(-p_h)); +P_{i_4}(-A_i(-p_h))$ be the event of the policeman getting authorization to connect/disconnect the field of farmer i to basin h . Consider the following sequence of events.

- 1 The chief of the Water Authority authorizes the policeman to modify the connections between basins and fields. Formally:

$$\pi_1 = (A_{i_3}(\pi_{i_1,1}) \wedge P_{i_3}(\pi_{i_1,1}))?; \pi_{i_1,1} ; \dots ; \\ (A_{i_3}(\pi_{i_2,3}) \wedge P_{i_3}(\pi_{i_2,3}))?; \pi_{i_2,3}$$

- 2 The policeman removes the connection between basin 1 and i_1 's field. Formally:

$$\pi_2 = (A_{i_4}(-A_{i_1}(-p_1)) \wedge P_{i_4}(-A_{i_1}(-p_1)))?; -A_{i_1}(-p_1)$$

- 3 The policeman adds the connection between basin 1 and i_2 's field. Formally:

$$\pi_3 = (A_{i_4}(+A_{i_2}(-p_1)) \wedge P_{i_4}(+A_{i_2}(-p_1)))?; +A_{i_2}(-p_1)$$

- 4 The policeman removes the connection between basin 3 and i_2 's field. Formally:

$$\pi_4 = (A_{i_4}(-A_{i_2}(-p_3)) \wedge P_{i_4}(-A_{i_2}(-p_3)))?; -A_{i_2}(-p_3)$$

It can be shown that:

$$\vdash \text{INIT} \rightarrow \langle \pi_1; \pi_2; \pi_3; \pi_4 \rangle \text{GOAL}$$

It says, given the initial situation *INIT*, the sequence of events $\pi_1; \pi_2; \pi_3; \pi_4$ is an executable and authorized (legal) procedure which ensures that *GOAL* will be achieved.

We may also quantify over an agent's actions: the valid formula

$$\vdash \text{INIT} \rightarrow \diamond_{\{i_4\}}^A (\neg A_{i_1}(\neg p_3) \wedge \neg A_{i_2}(\neg p_3))$$

expresses that the policeman has the ontic capability to guarantee that no farmer has access to basin 3.

Similarly:

- $\not\vdash \text{INIT} \rightarrow \diamond_{\{i_3\}}^{\text{AP}} \text{GOAL}$: i_3 cannot ensure *GOAL* alone.
- $\vdash \text{INIT} \rightarrow \diamond_{\{i_3, i_4\}}^{\text{AP}} \text{GOAL}$.

Outline

- 1 Basic logic
- 2 Capabilities and permissions
- 3 Constitutive rules**

Normative systems are based both on regulative as well as constitutive (i.e. non-regulative) components [Alchourron and Bulygin 1971].

According to Searle for instance “[. . .] regulative rules regulate antecedently or independently existing forms of behavior [. . .]. But constitutive rules do not merely regulate, they create or define new forms of behavior” [Searle 1969].

In Searle’s theory [Searle 1969, Searle 1995], constitutive rules are expressed by means of “counts-as” assertions of the form “X counts as Y in context C” where the context C refers to the normative system in which the rule is specified.

Constitutive rules relate “brute” physical facts, actions, etc. with institutional facts, actions, etc.

We add to the language a primitive $\alpha_1 \rightsquigarrow \alpha_2$ that reads that the occurrence of α_1 *counts as* the occurrence of α_2 , where α_1 and α_2 are assignments.

Example (road traffic regulations): $+70mph \rightsquigarrow +overspeed$ means that if a car runs at 70mph this counts as an overspeeding.

Technically, our modelling is inspired from the solution to the ramification problem in reasoning about actions [Thielscher 1997].

Definition

A model with constitutive rules is a tuple (V, C) such that V is a valuation and $C \subseteq \mathcal{A} \times \mathcal{A}$ is a *reflexive* and *transitive* relation over assignments that satisfies the following coherence constraint:

$$(Coh) \quad (\alpha, +q) \notin C \text{ or } (\alpha, -q) \notin C$$

When $(\alpha_1, \alpha_2) \in C$ then the occurrence α_1 counts as the occurrence of α_2 . The constraint *(Coh)* says that an event cannot have inconsistent ramifications.

Note that it follows from *(Coh)* together with reflexivity of C that $(+q, -q) \notin C$, $(+P_i(\alpha), -P_i(\alpha)) \notin C$, etc.

On the ontology of the “counts-as”

There is some disagreement whether the “counts-as” relation should satisfy transitivity.

In addition, some authors argue that the “counts-as” should satisfy contraposition [Grossi et al. 2006], while others have a different opinion on this matter [Jones and Sergot 1996].

In our framework, if desired, the constraint “if $(+p, +q) \in C$ then $(-q, -p) \in C$ ” could be added to C .

The truth condition for the “counts-as” construction is:

$$(V, C) \models \alpha_1 \rightsquigarrow \alpha_2 \quad \text{iff} \quad (\alpha_1, \alpha_2) \in C$$

We have to modify the definition of the relation R_π in order to take the relation C into account; we only change the case where π is an atomic event α :

$$R_\alpha = \{(V, (V \setminus \{q : (\alpha, -q) \in C\}) \cup \{q : (\alpha, +q) \in C\})\}$$

For example, the following schemas are valid:

$$\begin{aligned} &\alpha \rightsquigarrow \alpha \\ &(\alpha_1 \rightsquigarrow \alpha_2) \wedge (\alpha_2 \rightsquigarrow \alpha_3) \rightarrow (\alpha_1 \rightsquigarrow \alpha_3) \\ &\neg((\alpha \rightsquigarrow +q) \wedge (\alpha \rightsquigarrow -q)) \\ &\langle \alpha \rangle q \leftrightarrow ((\alpha \rightsquigarrow +q) \vee (q \wedge \neg(\alpha \rightsquigarrow -q))) \end{aligned}$$

The last equivalence is reminiscent of successor state axioms in the Situation Calculus [Reiter 2001]. It follows from it that

$$(\alpha \rightsquigarrow +q) \rightarrow \langle \alpha \rangle q.$$

For example, if driving at 80mph counts as overspeeding (i.e. $+80mph \rightsquigarrow +overspeed$) then, after starting to drive at 80mph, the driver will be over the speed limit (i.e. $\langle +80mph \rangle overspeed$).

The proofs of the complexity results for PDL^{\leftarrow} can be adapted straightforwardly to $\text{PDL}^{\leftarrow}_{\rightarrow}$.

Theorem

The problems of model checking and of satisfiability checking in $\text{PDL}^{\leftarrow}_{\rightarrow}$ are both PSPACE-complete.

Conclusions

- social software
- issues in logics of agents
- towards logics of social reality
 - more logic in higher-order obligations
 - agency / imperatives
 - dynamic constitutive rules