Common-sense reasoning as proto-scientific agent activity

Pierangelo Dell'Acqua*† and Luís Moniz Pereira†

* Department of Science and Technology - ITN
Linköping University, 601 74 Norrköping, Sweden
pier@itn.liu.se

† Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. We wish to model common-sense reasoning in situations where it contains some of the ingredients typical of proto-scientific reasoning, with a view to future elaboration and proof of concept. To model this proto-scientific narrative, we employ the integrative formal computational machinery we have been developing and implementing for rational cooperative epistemic agents. In our logic-based framework, agents can update their own and each other's theories, which are comprised of knowledge, active rules, integrity constraints, goals, abducibles, and preferences; they can engage in abductive reasoning involving updatable preferences; set each other goals; react to circumstances; plan and carry out actions; and revise their theories and preferences by means of concurrent updates on self and others.

1 Framework

1.1 Language

It is convenient to syntactically represent the theories of agents as propositional Horn theories. In particular, we represent default negation *not* A as a standard propositional variable. Propositional variables whose names do not begin with "*not*" and do not contain the symbols ":", "$\div$" and "$<$" are called *domain atoms*. For each domain atom A we assume a complementary propositional variable of the form *not* A. Domain atoms and negated domain atoms are called *domain literals*.

Communication is a form of interaction among agents. The aim of an agent β when communicating a message C to an agent α, is to make α update its current theory with C (i.e., to make α accept some desired for mental state by β). In turn, when α receives the message C from β, it is up to α whether or not incorporate C. This form of communication is formalized through the notion of projects and updates. Propositional variables of the form α:C (where C is defined below) are called *projects*. α:C denotes the intention (of some agent

β) of proposing the updating of the theory of agent α with C. Projects can be negated. A negated project of the form *not* α:C denotes the intention of the agent of not proposing the updating of the theory of agent α with C. Projects and negated projects are generically called *project literals*.

Propositional variables of the form $\beta \div C$ are called *updates*. $\beta \div C$ denotes an update with C in the current theory (of some agent α), that has been proposed by β. Updates can be negated. A negated update of the form *not* $\beta \div C$ in the theory of some agent α indicates that agent β does not have the intention to update the theory of agent α with C. Updates and negated updates are called *update literals*.

Preference information is used along with incomplete knowledge. In such a setting, due to the incompleteness of the knowledge, several models of a program may be possible. Preference reasoning is enacted by choosing among those possible models, through the expression of priorities amongst the rules of the program. Preference information is formalized through the notion of priority atoms. Propositional variables of the form $n_r < n_u$ are called *priority atoms*. $n_r < n_u$ means that rule r (whose name is n_r) is preferred to rule u (whose name is n_u). Priority atoms can be negated. *not* $n_r < n_u$ means that rule r is not preferred to rule u. Priority atoms and negated priority atoms are called *priority literals*.

Domain atoms, projects, updates and priority atoms are generically called *atoms*. Domain literals, project literals, update literals and priority literals are generically called *literals*.

Definition 1. *A* generalized rule *is a rule of the form* $L_0 \leftarrow L_1, \ldots, L_n$ *with* $n \geq 0$ *where every* L_i $(0 \leq i \leq n)$ *is a literal.*

Definition 2. *A* domain rule *is a generalized rule* $L_0 \leftarrow L_1, \ldots, L_n$ *whose head* L_0 *is a domain literal distinct from false and not false, and every literal* L_i $(1 \leq i \leq n)$ *is a domain literal or an update literal.*

Definition 3. *An* integrity constraint *is a generalized rule whose head is the literal false or not false.*

Integrity constraints are rules that enforce some condition on states, and they take the form of denials. To make integrity constraints updatable, we allow the domain literal *not false* to occur in the head of an integrity constraint. For example, updating the theory of an agent α with *not false* $\leftarrow$ *relaxConstraints* has the effect to turn off the integrity constraints of α if *relaxConstraints* holds. Note that the body of an integrity constraint can contain any literal. The following definition introduces rules that are executed bottom-up. To emphasize this aspect we employ a different notation for them.

Definition 4. *An* active rule *is a generalized rule whose head* Z *is a project literal and every literal* L_i $(1 \leq i \leq n)$ *in its body is a domain literal or an update literal. We write active rules as* $L_1, \ldots, L_n \Rightarrow Z$.

Active rules can modify the current state, to produce a new state, when triggered. If the body $L_1, \ldots, L_n$ of the active rule is satisfied, then the project (fluent) Z can be selected and executed. The head of an active rule is a project, either internal or external. An *internal project* operates on the state of the agent itself (self-update), e.g., if an agent gets an observation, then it updates its knowledge. *External projects* instead are performed on other agents, e.g., when an agent wants to update the theory of another agent.

To express preference information in logic programs we introduce the notion of priority rule.

Definition 5. *A* priority rule *is a generalized rule $L_0 \leftarrow L_1, \ldots, L_n$ whose head L_0 is a priority literal and every L_i $(1 \leq i \leq n)$ is a domain literal, an update literal, or a priority literal.*

Priority rules are also subject to updating.

Definition 6. *A* query *takes the form $?- L_1, \ldots, L_n$ with $n \geq 0$, where every L_i $(1 \leq i \leq n)$ is a domain literal, an update literal, or priority literal.*

We assume that for every project α:C, C is either a domain rule, an integrity constraint, an active rule, a priority rule or a query. Thus, a project can take one of the forms:

α:$(L_0 \leftarrow L_1, \ldots, L_n)$ $\quad$ α:$(L_1, \ldots, L_n \Rightarrow Z)$
α:$(\mathit{false} \leftarrow L_1, \ldots, L_n, Z_1, \ldots, Z_m)$ $\quad$ α:$(?{-}L_1, \ldots, L_n)$
α:$(\mathit{not\ false} \leftarrow L_1, \ldots, L_n, Z_1, \ldots, Z_m)$

Let $\mathcal{A}$ be a set of domain literals distinct from *false*. We call the domain literals in $\mathcal{A}$ *abducibles*. Abducibles can be thought of as hypotheses that can be used to extend the current theory of the agent in order to provide an "explanation" for given queries. Explanations are required to meet all the integrity constraints. Abducibles may also be defined by domain rules as the result of a self-update which adopts an abducible as a rule.

The reader can refer to [7, 8] for the declarative and procedural semantics of our framework of abductive logic-based agents, to [5] for a logic-based agent architecture, to [1] for a proof procedure of updating plus preferring reasoning, and to [6] for an asynchronous multi-agent system in which the interaction among agents is characterized by a transition rule system.

N.B.: In the sequel, rules with variables stand for the set of all their ground instances with respect to the Herbrand universe of the program.

1.2 Abductive Agents

The knowledge of an agent can dynamically evolve when the agent receives new knowledge, albeit by self-updating rules, or when it abduces new hypotheses to explain observations. The new knowledge is represented in the form of an updating program, and the new hypotheses in the form of a finite set of abducibles.

Definition 7. *An* updating program *U is a finite set of updates.*

An updating program contains the updates that will be performed on the current knowledge state of the agent. To characterize the evolution of the knowledge of an agent we need to introduce the notion of sequence of updating programs. In the remaining, let $S = \{1, \ldots, s, \ldots\}$ be a set of natural numbers. We call the elements $i \in S \cup \{0\}$ *states*. A *sequence of updating programs* $\mathcal{U} = \{U^s \mid s \in S\}$ is a set of updating programs U^s superscripted by the states $s \in S$.

Definition 8. *An* agent α at state s, *written as* Ψ^s_α, *is a pair* $(\mathcal{A},\mathcal{U})$, *where* $\mathcal{A}$ *is the set of abducibles and* $\mathcal{U}$ *is a sequence of updating programs* $\{U^1, \ldots, U^s\}$. *If* $s = 0$, *then* $\mathcal{U} = \{\}$.

An agent α at state 0 is defined by a set of abducibles $\mathcal{A}$ and an empty sequence of updating programs, that is $\Psi^0_\alpha = (\mathcal{A}, \{\})$. At state 1, α is defined by $(\mathcal{A}, \{U^1\})$, where U^1 is the updating program containing all the updates that α has received at state 0 either from other agents or as self-updates. In general, an agent α at state s is defined by $\Psi^s_\alpha = (\mathcal{A}, \{U^1, \ldots, U^s\})$, where each U^i is the updating program containing the updates that α has received at state $i - 1$.

1.3 Abductive stable models

In the remainder of the paper, by (2-valued) *interpretation* M we mean any consistent[1] set of literals. Given a generalized rule r of the form $L_0 \leftarrow L_1, \ldots, L_n$, we write $head(r)$ to indicate L_0 and $body(r)$ to indicate $L_1, \ldots, L_n$.

Definition 9. *Let* P *be a set of generalized rules and* M *an interpretation. The set of* default assumptions *is:*

$$Default(P, M) = \{not\ A \mid \nexists r \in P \text{ such that } head(r) = A \text{ and } M \models body(r)\}.$$

The knowledge of an agent α is characterized at the start (at state 0) by the set of all default assumptions *not* A (that is, by $Default(\{\}, M)$). Its knowledge can dynamically evolve when α receives new knowledge, via a sequence of updating programs $\mathcal{U} = \{U^1, \ldots, U^s\}$. Intuitively, the evolution of knowledge may be viewed as the result of, starting with the set of all default assumptions, updating it with U^1, updating next with U^2, and so on. The role of updating is to ensure that the rules contained in these newly added updates are in force, and that previous rules are still valid (by inertia) as far as possible, i.e., they are in force. This rationale is at the basis of the notion of rejected rules, spelled out below.

A rule r proposed via an update in U^i by an agent β is rejected at state s by an interpretation M if there exists a rule r' proposed via a subsequent update in U^j by an agent α, such that the head of r' is the complement of the head of r, the body of r' is true in M and the update is not distrusted.

Definition 10. *Let* $\mathcal{U} = \{U^i \mid i \in S\}$ *be a sequence of updating programs and* M *an interpretation. The set of* rejected rules *at state* s *is:*

$$\begin{aligned} Reject(\mathcal{U}, s, M) = \{r \mid \exists\, (\beta \div r) \in U^i \text{ and } \exists\, (\alpha \div r') \in U^j \text{ such that } i < j \leq s, \\ head(r) = not\ head(r'), M \models body(r') \text{ and } M \not\models distrust(\alpha \div r')\} \end{aligned}$$

[1] A set M is *consistent* iff there exists no atom X such that $X \in M$ and *not* $X \in M$.

The idea behind the updating process is that newer rules reject older ones in such a way that contradictions can never arise between them. Thus, contradictions could only ever arise between rules introduced at the same state. Furthermore, an agent α can prevent any type of updates from an agent β via the use of *distrust* in the theory of α, e.g., $distrust(\beta \div C) \leftarrow liar(\beta)$.

As the head of an active rule is a project (and not a domain atom), active rules can only be rejected by active rules. Rejecting an active rule r makes r not triggerable even if its body is true in the model. Thus, by rejecting active rules, we make the agent less reactive.

Let $\Psi^s_\alpha = (\mathcal{A}, \mathcal{U})$ be an agent α at state s and $La \subseteq \mathcal{A}$ a set of abducibles. We write $\mathcal{U} + La$ to indicate the sequence of updating programs $\mathcal{U} \cup \{U^{s+1}\}$, where $U^{s+1} = \{\alpha \div L \mid \text{for every } L \in La\}$. That is, $\mathcal{U} + La = \{U^1, \ldots, U^s, U^{s+1}\}$.

Definition 11. *Let $\Psi^s_\alpha = (\mathcal{A}, \mathcal{U})$ be an agent α at state s and M an interpretation. Let $La \subseteq \mathcal{A}$ be a set of abducibles and $\mathcal{U}' = \mathcal{U} + La$ a sequence of updating programs. M is an* abductive stable model of agent α at state s with hypotheses *La iff:*

- *$false \notin M$*
- *$M = least(\mathcal{X} \cup Default(T, M) \cup \bigcup_{1 \leq i \leq s} U^i)$, where:*

$$
\begin{aligned}
T &= \{r \mid \exists\, (\beta \div r) \text{ in } \bigcup_{1 \leq i \leq s+1} U^i \text{ such that } M \not\models distrust(\beta \div r)\} \\
R &= Reject(\mathcal{U}', s+1, M) \\
\mathcal{X} &= T - R
\end{aligned}
$$

Note that the abducibles are treated as a *virtual update*. That is, to compute the abductive stable models the abducibles abduced by α at state s are treated as if they were internal updates of α at state $s+1$. The virtual update is only used to compute the abductive stable models, and there is no commitment to what α will receive as update at the next state $s+1$.

1.4 Preferred abductive stable models

While updates allow us to deal with a dynamically evolving world, where rules change in time, preferences allow us to choose among various possible models of the world and among possible incompatible reactions. In [2], two criteria are established to remove unpreferred generalized rules in a program: removing unsupported generalized rules, and removing less preferred generalized rules defeated by the head of some more preferred one. Unsupported generalized rules are rules whose head is true in the model and whose body is defeated by the model. Below we write $body^+(r)$ (resp. $body^-(r)$) to indicate the atoms (resp. the negated atoms) in the body of a rule r.

Definition 12. *Let P be a set of generalized rules and M an interpretation. The set of* unsupported generalized rules *of P and M is:*

$$Unsup(P, M) = \{r \in P \mid M \models head(r), M \models body^+(r) \text{ and } M \not\models body^-(r)\}.$$

Unpref(P, M) *is a set of* unpreferred generalized rules *of* P *and* M *iff:*

$$Unpref(P, M) = least(Unsup(P, M) \cup \mathcal{X})$$

where $\mathcal{X} = \{\ r \in P \mid\ \exists u \in (P - Unpref(P, M))$ *such that:*
$M \models n_u < n_r,\ M \models body^+(u)$ *and* [*not* $head(u) \in body^-(r)$ *or*
$(not\ head(r) \in body^-(u),\ M \models body(r))$] $\}$.

In other words, a generalized rule is unpreferred if it is unsupported or defeated by a more preferred generalized rule (which is not itself unpreferred), or if it attacks (i.e., attempts to defeat) a more preferred generalized rule. The following definition introduces the notion of preferred abductive stable model of an agent α at a state s with set of hypotheses *La*. Given a sequence of updating programs $\mathcal{U}$ and the hypotheses *La* assumed at state s by α, a preferred abductive stable model of α at state s with hypotheses *La* is a stable model of the program $\mathcal{X}$ that extends P to contain all the updates in $\mathcal{U}$, all the hypotheses in *La*, and all those rules whose updates are not distrusted but that are neither rejected nor unpreferred. The preferred abductive stable model contains also the selected projects.

Definition 13. *Let* $\Psi^s_\alpha = (\mathcal{A}, \mathcal{U})$ *be an agent* α *at state* s *and* M *an abductive stable model of* α *at state* s *with hypotheses La. Let* $\mathcal{U}' = \mathcal{U} + La$ *be a sequence of updating programs.* M *is a* preferred abductive stable model of agent α at state s with hypotheses *La iff:*

- $\forall r_1, r_2$: *if* $(n_{r_1} < n_{r_2}) \in M$, *then* $(n_{r_2} < n_{r_1}) \notin M$
- $\forall r_1, r_2, r_3$: *if* $(n_{r_1} < n_{r_2}) \in M$ *and* $(n_{r_2} < n_{r_3}) \in M$, *then* $(n_{r_1} < n_{r_3}) \in M$
- $M = least(\mathcal{X} \cup Default(T, M) \cup \bigcup_{1 \le i \le s} U^i)$, *where:*

$$T = \{r \mid \exists (\beta \div r) \text{ in } \bigcup_{1 \le i \le s+1} U^i \text{ such that } M \not\models distrust(\beta \div r)\}$$
$$R = Reject(\mathcal{U}', s+1, M)$$
$$\mathcal{X} = (T - R) - Unpref(T - R, M)$$

T is the set containing all the rules in updates that are trusted from α according to M, and R is the set of all the rules that are rejected at state s. $\mathcal{X}$ is the set of all the trusted rules that are neither rejected nor unpreferred.

Definition 14. *An* abductive explanation *for a query* Q *is any set* $La \subseteq \mathcal{A}$ *of hypotheses such that there exists a preferred abductive stable model* M *with hypotheses La and* $M \models Q$.

Note that at state s an agent α may have several abductive explanations for a query Q.

1.5 Agent Cycle

Every agent α can be thought of as a pair $\Psi_\alpha = (\mathcal{A},\mathcal{U})$, where $\mathcal{A}$ is a set of abducibles and $\mathcal{U}$ is a sequence of updating programs, equipped with a set of *inputs* represented as *updates*. The abducibles are used as explanations for proving the goals of the agent, and updates can be used to solve the goals as well as to trigger new goals. The basic "engine" of an agent α is an abductive logic programming proof procedure, executed via the cycle represented in Fig. 1.

Cycle(α,s,Ψ_α^{s-1},G), where $\Psi_\alpha^{s-1} = (\mathcal{A},\mathcal{U})$ and $\mathcal{U} = \{U^1,\ldots,U^{s-1}\}$.

1. Observe and record any input in the updating program U^s.
2. Select a goal g in $G \cup \mathit{Goals}(\alpha, U^s)$ and execute g. Let $\mathit{La} \subseteq \mathcal{A}$ be an abductive explanation of g, if g is provable; otherwise, let $\mathit{La} = \{\}$.
   Let $G' = G \cup \mathit{Goals}(\alpha, U^s) - \{g\}$.
3. Execute all the projects in $\mathit{ExecProj}(\mathit{La})$.
4. Cycle with $(\alpha,s+1,\Psi_\alpha^s,G')$, where $\Psi_\alpha^s = (\mathcal{A},\mathcal{U} \cup \{U^s\})$.

Fig. 1. *The agent cycle*

Step 1: The cycle of an agent α starts at state s by observing any inputs (updates from other agents) from the environment, and by recording them in the updating program U^s.

Step 2: A goal g is selected from $G \cup \mathit{Goals}(\alpha, U^s)$, where

$$\mathit{Goals}(\alpha, U^s) = \{?{-}g \mid \alpha{\div}(?{-}g) \in U^s\}.$$

Note that only the goals issued by the agent α itself are executed. The goals issued by other agents are treated as normal updates[2]. Then, g is executed. Here, we can use any abductive proof procedure, such as ABDUAL [3, 4].

Step 3: The executable projects are executed. The set *ExecProj* of executable projects of an agent depends on the kind of agent we want to model. For instance, in case of a cautious agent, the set of executable projects is:

$$\mathit{ExecProj}(\mathit{La}) = \{\, \beta{:}C \mid \text{for every preferred abductive stable model } M \text{ at state } s \text{ with hypotheses } \mathit{La}, \text{ it holds that } \beta{:}C \in M\}.$$

If an executable project takes the form $\beta : C$ (meaning that agent α intends to update the theory of agent β with C at state s), then (once executed) the update $\alpha \div C$ will be available as input to the cycle of the agent β.

[2] In this way, α retains control upon deciding on which goals (requested by other agents) to execute. For example, the theory of α may contain the active rule: $\beta{\div}(?{-}g), \mathit{Cond} \Rightarrow \alpha{:}(?{-}g)$ which states that if α has been requested to prove a goal $?{-}g$ by β and some condition *Cond* holds, then α will issue the internal project to prove the goal $?{-}g$.

Step 4: Finally, the agent cycles by increasing the state, by incorporating the updating program U^s into $\mathcal{U}$, and with the new list G' of goals.

Initially, the cycle of α is Cycle(α,1,Ψ^0_α,{}) with $\Psi^0_\alpha = (\mathcal{A}, \{\})$.

1.6 Sequential execution of actions

To express the sequential execution of actions we introduce the notion of sequence of abducibles.

Definition 15. *Let $\mathcal{A}$ be a set of abducibles. Then, a* sequence of abducibles *is inductively defined as follow:*

1. *every $a \in \mathcal{A}$ is a sequence of abducibles;*
2. *if $a \in \mathcal{A}$ and x is a sequence of abducibles, then $a \diamond x$ is a sequence of abducibles;*
3. *nothing else is a sequence of abducibles.*

Intuitively, a sequence $a \diamond b \diamond c$ of abducibles states to abduce first a, then b and finally c. This capability allows an agent to execute actions in sequence. For instance, if the abducibles a, b and c occur in three distinct active rules r_a, r_b and r_c, then the sequence $a \diamond b \diamond c$ permits executing the projects occurring in the heads of r_a, r_b and r_c sequentially.

Consider an agent α that has a plan p consisting of two actions, a followed by b. Suppose that p is executable if some condition *cond* holds. Let $\mathcal{A} = \{a, b\}$ be a set of abducibles. Such a plan can be expressed as:

$$p \leftarrow cond, a \diamond b$$
$$a, precA \Rightarrow \alpha : effectA$$
$$b, precB \Rightarrow \alpha : effectB$$

where *precX* and *effectX* indicate the precondition and the effect of an action X. By launching the execution of p, if *cond* holds then the sequence $a \diamond b$ of abducibles is executed. First, a is abduced and if *precA* holds the action A can be executed. The effects of executing A are expressed via the project $\alpha : effectA$. If instead *precA* does not hold, then the plan p cannot be accomplished. Once the execution of A is terminated, b is abduced and the action B executed.

Such a plan can be coded in our framework as:

$$p \leftarrow cond, start(p) \quad (1)$$
$$start(p) \Rightarrow \alpha : exec(p) \quad (2)$$
$$start(p) \Rightarrow \alpha : ?{-}a \quad (3)$$
$$stop(p) \Rightarrow \alpha : not\ exec(p) \quad (4)$$
$$exec(p), ta \Rightarrow \alpha : ?{-}b \quad (5)$$
$$exec(p), tb \Rightarrow \alpha : ?{-}tp \quad (6)$$
$$a, precA \Rightarrow \alpha : effectA \quad (7)$$
$$a, precA \Rightarrow \alpha : ?{-}ta \quad (8)$$
$$b, precB \Rightarrow \alpha : effectB \quad (9)$$
$$b, precB \Rightarrow \alpha : ?{-}tb \quad (10)$$

with the new set of abducibles $\mathcal{B} = \{start(p), stop(p), a, b, ta, tb, tp\}$. Launching the plan p, by means of the rule (1), has the effect of abducing $start(p)$, which in turn will trigger the active rules (2) and (3). The active rule (2) updates the theory of the agent α with $exec(p)$ to indicate that α is executing the plan p. The active rule (3), by launching the query $?{-}a$ has the effect of making the agent α abduce a at the next agent cycle. The active rules (5) and (6) model the sequencing of the actions a and b. If $precA$ holds, abducing a while proving the plan p will trigger the active rules (7) and (8). The effect of triggering them is that the project $\alpha : effectA$ will be executed and the goal $?{-}ta$ launched at the next agent cycle. As ta is an abducible, at the next cycle ta will be abduced. This indicates that the action a is terminated. The action b can then be executed. This is achieved by triggering the active rule (5) that will launch the goal $?{-}b$. At the next cycle b will be abduced and if $precB$ holds the active rules (9) and (10) will be triggered. The project $\alpha : effectB$ will be executed and the goal $?{-}tb$ launched. Abducing tb indicates that the action b is terminated and therefore the entire plan p has been accomplished (i.e., the active rule (10) is triggered).

Recall that abductions adopted at one state do not carry over to the next state. Nevertheless, that can be achieved if desired, either by update imposing the abduced literals as facts, or else by enforcing their abduction anew, via the updating of an abduction goal to that effect into the next state.

2 Modelling Proto-scientific Reasoning by Rational Agents

Next we illustrate how to model, with the above instruments, common-sense reasoning in situations where it contains some of the ingredients typical of proto-scientific reasoning, with a view to future elaboration, proof of concept, and extension of the approach to scientific reasoning itself. To do so, we construe an exemplificative narrative of a doctor/patient cooperative diagnostic situation development, involving a combination of a number of common rational abilities illustrative of proto-scientific reasoning and acting, which demand their joint exercise, both in an individual and a cooperative fashion, akin to scientific theory refinement. To model this proto-scientific narrative, we employ the integrative formal computational machinery we have been developing and implementing for rational cooperative epistemic agents, and recapitulated above. Indeed, in our logic-based framework, agents can update their own and each other's theories, which are comprised of knowledge, active rules, integrity constraints, goals, abducibles, and preferences; they can engage in abductive reasoning involving updatable preferences; set each other goals; react to circumstances; plan and carry out actions; and revise their theories and preferences by means of concurrent updates on self and others.

The narrative below involves an initial patient situation requiring causal explanation; plus his interactive recourse to a doctor, whose initial therapeutic theory, diagnoses, and diagnostic preferences, are conducive to his advising the patient; and furthermore, initiative is required by the patient about courses

of action to obtain prescribed medicine, and experimentation and observation of its effect; but meanwhile, unforeseen circumstances provide unexpected new information and action from a third agent, become pertinent for the problem at hand; as a result, the doctor's original theory is revised, in what regards his diagnostic preferences, in the light of the patient's experimentation, and the unexpected triggering of an unforeseen action by the third party. The example has been fully tested with our implementation.

2.1 Requiring causal explanation

John runs a small software house and likes working until late when needed. He drinks coffee and has been a heavy smoker from a long time. Recently, he got problems with sleeping. He would like to have a break from his work, perhaps a vacation, but he does not have any company. Thus, he keeps on working. John does not know what the cause is and decides to visit a doctor. He tells the doctor about his sleeping problems and asks him what is the cause. John answers any question of of the doctor.
(Recall that we write generalized rules containing variables as a shorthand for all their ground instances.)

$work$
$likeWork$
$sProblems$
$badHabits$
$longTimeBadHabits$
$explanation \leftarrow cause(P, X, sProblems)$
$sProblems, company \Rightarrow john : takeVacation$
$sProblems, not\ company, not\ explanation \Rightarrow doctor : sProblems$
$sProblems, not\ company, not\ explanation \Rightarrow doctor : ?{-}askReason(sProblems)$
$doctor \div (?{-}Q), Q \Rightarrow doctor : Q$
$doctor \div (?{-}Q), not\ Q \Rightarrow doctor : not\ Q$

Since the theory of John does not contain any priority rules, the preferred stable model of John at the current state is equivalent to the abductive stable model $M_1 = \{work, likeWork, sProblems, badHabits, longTimeBadHabits, doctor : sProblems, doctor : ?{-}askReason(sProblems)\}$. According to the definition of agent cycle, John will execute the two projects in M_1 (step 3), and then he will cycle (step 4). When the doctor will observe his inputs (step 1), he will receive the two updates:

$$john \div sProblems$$
$$john \div (?{-}askReason(sProblems))$$

Any time the doctor is asked a reason for a medical problem by a patient, the doctor must make a diagnosis. To do so, he must first collect the relevant information about the medical problem from the patient, then diagnose the cause of the problem and tells it to the patient together with the suggested treatment.

$$\mathcal{A} = \{collectRelInfo(\ldots), cause(\ldots), answer(\ldots), treatment(\ldots), start(\ldots), stop(\ldots)\}$$

$$
\begin{array}{l}
diagnosis(P, X, Y) \leftarrow \; collectRelInfo(P, Y) \diamond \\
\qquad\qquad\qquad\quad cause(P, X, Y) \diamond answer(P, cause(P, X, Y)) \diamond \\
\qquad\qquad\qquad\quad treatment(X, T) \diamond answer(P, treatment(X, T)) \\
relevant(work, sProblems) \\
relevant(likeWork, sProblems) \\
relevant(badHabits, sProblems) \\
relevant(longTimeBadHabits, sProblems) \\
P \div (?{-}askReason(Y)) \Rightarrow doctor : ?{-}diagnosis(P, X, Y) \\
collectRelInfo(P, Y), relevant(R, Y) \Rightarrow P : ?{-}R \\
answer(P, A) \Rightarrow P : A
\end{array}
$$

Since the doctor does not have any goal to execute at step 2, he starts executing his projects (step 3). At this step, the unique project in his preferred abductive stable model is $doctor : ?{-}diagnosis(john, X, sProblems)$. Thus, at the next cycle of the doctor, the plan to make a diagnosis will start (step 2) by collecting all the information relevant for the sleeping problems. This will make the doctor to ask John the following questions $john : ?{-}work$, $john : ?{-}likeWork$, $john : ?{-}badHabits$ and $john : ?{-}longTimeBadHabits$.

After the replies of John (recall that John answers every question of the doctor), the theory of the doctor will be updated with:

$$
\begin{array}{l}
john \div work \\
john \div likeWork \\
john \div badHabits \\
john \div longTimeBadHabits
\end{array}
$$

After having executed the first action of the diagnosis plan, the doctor must find out a cause and the corresponding treatment of the problem, and tell them to John.

The doctor has three hypotheses that may explain the John's sleeping problems: bad habits (like drinking coffee and smoking), stress, or insomnia. The doctor evaluates John with the help of John's medical history, and he diagnoses a chronic insomnia. The doctor discards bad habits since John has been drinking coffee and smoking for many years without the attending sleeping problems. The doctor prefers to diagnose chronic insomnia attributable to stress since John's stress may be positive stress due to the fact that John likes his work.

$$
\begin{array}{l}
cause(P, insomnia, sProblems) < cause(P, stress, sProblems) \leftarrow pStress(P) \\
cause(P, stress, sProblems) < cause(P, insomnia, sProblems) \leftarrow stress(P), not\ pStress(P) \\
pStress(P) \leftarrow stress(P), P \div likeWork \\
stress(P) \leftarrow P \div work \\
false \leftarrow cause(P, badHabits, sProblems), P \div longTimeBadHabits
\end{array}
$$

Being $pStress(john)$ true in the theory of the doctor, the doctor prefers the abductive explanation $cause(john, insomnia, sProblems)$ to the abductive explanation $cause(john, stress, sProblems)$. Since it does not satisfy the integrity constraints, the abductive explanation $cause(john, badHabits, sProblems)$ is excluded by the doctor.

As treatment for insomnia, the doctor can either prescribe sleeping pills or suggest John to have a rest. Sleeping pills being preferable to a vacation on the

assumption that John can continue to work by having the pills, the doctor prescribes them.

$treatment(P, insomnia, sPills) < treatment(P, insomnia, rest) \leftarrow cWork(sPills), P \div work$
$cWork(sPills)$

According to the diagnosis plan, the doctor will tell John about the cause and the treatment for his sleeping problems. This is achieved by abducing $answer(john, cause(john, insomnia, sProblems))$ and $answer(john, treatment(john, insomnia, sPills))$ which trigger the corresponding active rules whose projects are $john : cause(john, insomnia, sProblems)$ and $john : treatment(john, insomnia, sPills)$. Once the doctor executes the two projects, John will update his theory at the next cycle with:

$$doctor \div cause(john, insomnia, sProblems)$$
$$doctor \div treatment(john, insomnia, sPills)$$

2.2 Agent initiative to obtain prescribed medicines

John tries to get sleeping pills before going to bed. Since it is late, he thinks the pharmacy nearby is closed, and plans to go to another pharmacy downtown.

$$\mathcal{A}_2 = \{goToPharmacy, buyPills, go(\ldots)\}$$

$takePills \leftarrow doctor \div treatment(john, insomnia, sPills)$
$getPills \leftarrow goToPharmacy \diamond buyPills$
$choosePharmacy(f1) \leftarrow open(f1), not\ choosePharmacy(f2) \quad (r1)$
$choosePharmacy(f2) \leftarrow open(f2), not\ choosePharmacy(f1) \quad (r2)$
$r1 < r2 \leftarrow near(f1)$
$near(f1)$
$open(f2)$
$goTo(f1) \leftarrow choosePharmacy(f1), go(nearSquare)$
$goTo(f2) \leftarrow choosePharmacy(f2), go(square) \diamond go(center)$
$takePills, not\ havePills \Rightarrow john : ?{-}getPills$
$goToPharmacy \Rightarrow john : ?{-}goTo(X)$
$buyPills \Rightarrow john : havePills$

Since John must take the sleeping pills and he does not have them, by triggering the first active rule above, the internal project $john : ?{-}getPills$ is executed. This has the effect of launching the plan to get to a pharmacy and to buy the pills.

The theory of John contains a priority rule stating to prefer r_1 to r_2 if the pharmacy f_1 is near. As f_1 is not open, it cannot be chosen (i.e., the body of r_1 is false). In fact, the rule r_2 does not belong to the set of unpreferred rules (see Def. 12) since $body^+(r_1)$ is not true. Being $choosePharmacy(f1)$ false and $open(f2)$ true, John chooses f_2. To get to f_2, John must get to the square and to the center.

While he is going there, he notices lights on in the nearby pharmacy and he concludes that it is open. So he decides to interrupt his original plan and go to this nearby pharmacy. It being open, he buys the pills.

$$environment \div lightOn(f1)$$
$$lightOn(f1) \Rightarrow john : open(f1)$$
$$open(f1) \Rightarrow john : ?-stop(getPills)$$
$$open(f1) \Rightarrow john : ?-getPills$$

Note that the internal project $john : ?-stop(getPills)$ has the effect of making John abduce $stop(getPills)$. Thus, John will stop the execution of the plan and he will relaunch his goal to get to a pharmacy. Now, since f_1 is open, the rule r_1 is preferrable to the rule r_2. Thereby, John will go to the nearby pharmacy.

Note also that here we have encoded plans directly in the agents knowledge. In general, one can employ a planner that given a task produces sequences of actions to be executed. We have shown only actions performed in sequence, but other types of actions can be expressed in our framework as well, like parallel actions and sensing actions.

2.3 Patient's experimentation

John takes the sleeping pills. But his work implies coffee and stress, and the attempt fails. One day he meets his friend Pamela and tells her about his problems.

$$john \div sProblems$$
$$john \div takingPills$$
$$john \div stillsProblems$$

Pamela advises him to suspend the taking of sleeping pills, not to work so hard and to have some rest. She invites him for an exciting vacation to one of the carabbean islands.

$$friend(john)$$
$$P \div sProblems \Rightarrow P : rest$$
$$P \div takingPills, P \div stillsProblems \Rightarrow P : not\ takePills$$
$$friend(P) \Rightarrow P : invite(vacation, carabbean)$$

John decides to follow her piece of advice and to accept her invitation.

$$pamela \div invite(vacation, carabbean)$$
$$A \div invite(X, Y), female(A) \Rightarrow A : accept(X, Y)$$
$$female(pamela)$$

2.4 Doctor's original theory revised

Subsequently John can sleep and tells the doctor what happened.

$$john \div takingPills$$
$$john \div stillsProblems$$
$$john \div vacation$$
$$john \div not\ sProblemsAfterVacation$$

The doctor now revises his theory of preferences to suggest a vacation in the first place in the future.

$P \div takingPills, P \div stillsProblems \Rightarrow$
$\quad doctor : not\ (treatment(Q, insomnia, sPills) < treatment(Q, insomnia, rest))$
$P \div vacation, P \div not\ sProblemsAfterVacation \Rightarrow$
$\quad doctor : treatment(Q, insomnia, rest) < treatment(Q, insomnia, sPills)$

The doctor updates his priority rules via the two active rules above in such a way to suggest a different treatment to other patients with insomnia problems.

3 Conclusion

We believe to have shown that the application of proto-scientific reasoning in common-sense examples, modelled by collections of rational agents, is an avenue of research worth pursuing with a view to further the modelling of collaborative scientific theory development and refinement. Three worthwhile aspects we did not touch upon, but which are already well within reach of present formal machinery are rule induction, argumentation, and mutual debugging.

Acknowledgements

L. M. Pereira acknowledges the support of POCTI project 40958 "FLUX - FleXible Logical Updates".

References

1. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial Intelligence*, LNAI 2424, pages 62–74, Berlin, 2002. Springer-Verlag.
2. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmn, G. Brewka, and L. M. Pereira, editors, *Logics in AI, Procs. JELIA'00*, LNAI 1919, pages 345–360, Berlin, 2000. Springer.
3. J. J. Alferes, L. M. Pereira, and T. Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. Theory and Practice of Logic Programming, 2004. To appear.
4. J. J. Alferes, L. M. Pereira, and T. Swift. Well-founded abduction via tabled dual programs. In D. De Schreye, editor, *ICLP'99*. MIT Press, 1999.
5. P. Dell'Acqua, M. Engberg, and L. M. Pereira. An architecture for a rational, reactive agent. 11th Portuguese Conf. on Artificial Intelligence, 2003. To appear.
6. P. Dell'Acqua, U. Nilsson, and L. M. Pereira. A logic based asynchronous multi-agent system. Computational Logic in Multi-Agent Systems (CLIMA02). Electronic Notes in Theoretical Computer Science (ENTCS), Vol. 70, Issue 5, 2002.
7. P. Dell'Acqua and L. M. Pereira. Enabling agents to update their knowledge and to prefer. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence, 10th Portuguese Int. Conf. on Artificial Intelligence (EPIA'01)*, LNAI 2258, pages 183–190. Springer-Verlag, 2001.
8. P. Dell'Acqua and L. M. Pereira. Preferring and updating in abductive multi-agent systems. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents' World (ESAW 2001)*, LNAI 2203, pages 57–73. Springer-Verlag, 2001.