Prospective Logic Agents

Luís Moniz Pereira*
Centro de Inteligência Artificial - CENTRIA
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
E-mail: lmp@di.fct.unl.pt
Corresponding author

Gonçalo Lopes
Centro de Inteligência Artificial - CENTRIA
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
E-mail: goncaloclopes@gmail.com

Abstract: As we face the actual possibility of modelling agent systems capable of non-deterministic self-evolution, we are confronted with the problem of having several different possible futures for any single agent. This issue brings the challenge of how to allow such evolving agents to be able to *look ahead*, prospectively, into such hypothetical futures, in order to determine the best courses of evolution from their own present, and thence to prefer amongst them. The concept of prospective logic programs is presented as a way to address such issues. We start by building on previous theoretical background, on evolving programs and on abduction, to construe a framework for prospection and describe an abstract procedure for its materialization. We take on several examples of modelling prospective logic programs that illustrate the proposed concepts and briefly discuss the ACORDA system, a working implementation of the previously presented procedure. We conclude by elaborating about current limitations of the system and examining future work scenaria.

Keywords: Evolving Agents; Future Prospection; Abduction; Preferences; Logic Programming; Abductive Stable Models; ACORDA; XSB-XASP system; Intelligent Systems

1 INTRODUCTION

Continuous developments in logic programming (LP) language semantics which can account for evolving programs with updates [Alferes et al., 2002, Alferes et al., 2000] have opened the door to new perspectives and problems amidst the LP and agents community. As it is now possible for a program to talk about its own evolution, changing and adapting itself through non-monotonic self-updates, one of the new looming challenges is how to use such semantics to specify and model logic based agents which are capable of anticipating their own possible future states and of preferring among them in order to further their goals, prospectively maintaining truth and consistency in so doing. Such predictions need to account not only for changes in the perceived external environment, but need also to incorporate available actions originating from the agent itself, and perhaps even consider possible actions and hypothetical goals emerging in the activity of other agents.

While being immersed in a world (virtual or real), every proactive agent should be capable, to some degree, of conjuring up hypothetical *what-if* scenaria while attending to a given set of integrity constraints, goals, and partial observations of the environment. These scenaria can be about hypothetical observations (what-if this observation were true?), about hypothetical actions (what-if this action were performed?) or hypothetical goals (what-if this goal was pursued?). As we are dealing with non-monotonic logics, where knowledge about the world is incomplete and revisable, a way to represent predictions about the future is to consider possible scenaria as tentative evolving hypotheses which *may* become true, pending subsequent confirmation or disconfirmation on further observations, the latter based on the expected consequences of assuming each of the scenaria.

We intend to show how rules and methodologies for the synthesis and maintenance of abductive hypotheses, extensively studied by several authors in the field of Abductive Logic Programming [Kakas et al., 1998, Kowalski, 2006b, Poole, 2000, Poole, 1997], can be used for effective, yet defeasible, prediction of an agent's future. Note that we are considering in this work a very broad notion of abduction, which can account for any of the types of scenaria mentioned above. Abductive reasoning by such prospective agents also benefits greatly from employing a notion of simulation allowing them to derive the consequences for each available scenario, as the agents imagine the possible evolution of their future states prior to actually taking action towards selecting one of them.

It is to be expected that a multitude of possible scenaria become available to choose from at any given time, and thus we need efficient means to prune irrelevant possibilities, as well as to enact preferences and relevancy preorders over the con-

sidered ones. Such preference specifications can be enforced either a priori or a posteriori w.r.t hypotheses making. A priori preferences are embedded in the knowledge representation theory itself and can be used to produce the most interesting or relevant conjectures about possible future states. Active research on the topic of preferences among abducibles is available to help us fulfill this purpose [Dell'Acqua and Pereira, 2005, Dell'Acqua and Pereira, 2007] and results from those works have been incorporated in the presently proposed framework.

A posteriori preferences represent meta-reasoning over the resulting scenaria themselves, allowing the agent to actually make a choice based on the imagined consequences in each scenario, possibly by attempting to confirm or disconfirm some of the predicted consequences, by attributing a measure of interest to each possible model, or simply by delaying the choice over some models and pursuing further prospection on the most interesting possibilities which remain open. At times, several hypotheses may be kept open simultaneously, constantly updated by information from the environment, until a choice is somehow forced during execution (e.g. by using escape conditions), or until a single scenario is preferred, or until none are possible.

In prospective reasoning agents, exploration of the future is essentially an open-ended, non-deterministic and continuously iterated process, distinct from the one-step, best-path-takes-all planning procedures. First, the use of abduction can dynamically extend the theory of the agent during the reasoning process itself in a context-dependent way so that no definite set of possible actions is implicitly defined. Second, the choice process itself typically involves acting upon the environment to narrow down the number of available options, which means that the very process of selecting futures can drive an agent to autonomous action. Unlike Rodin's thinker, a prospective logic agent is thus proactive in its look ahead of the future, acting upon its environment in order to anticipate, pre-adapt and enact informed choices efficiently. These two features imply that the horizon of search is likely to change at every iteration and the state of the agent itself can be altered during this search.

The study of this new LP outlook is essentially an innovative combination of fruitful research in the area, providing a testbed for experimentation in new theories of program evolution, simulation and self-updating, while launching the foundational seeds for modeling rational self-evolving prospective agents. Preliminary research results have proved themselves useful for a variety of applications and have led to the development of the ACORDA[1] system, successfully used in modelling diagnostic situations [Lopes and Pereira, 2006]. This paper presents a more formal abstract description of the procedure involved in the design and implementation of prospective logic agents.Some examples are also presented as an illustration of the proposed system capabilities, and some broad sketches are laid out concerning future research directions.

[1]ACORDA literally means "wake-up" in Portuguese. The *ACORDA* system project page is temporarily set up at: `http://articaserv.ath.cx/`

2 LOGIC PROGRAMMING FRAMEWORK

2.1 Language

Let $\mathcal{L}$ be any first order language. A domain literal in $\mathcal{L}$ is a domain atom *A* or its default negation *not A*, the latter expressing that the atom is false by default (CWA). A domain rule in $\mathcal{L}$ is a rule of the form:

$$A \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where *A* is a domain atom and $L_1, \ldots, L_t$ are domain literals. An integrity constraint in $\mathcal{L}$ is a rule of the form:

$$\bot \leftarrow L_1, \ldots, L_t \ \ (t > 0)$$

where $\bot$ is a domain atom denoting falsity, and $L_1, \ldots, L_t$ are domain literals.

A (logic) program *P* over $\mathcal{L}$ is a set of domain rules and integrity constraints, standing for all their ground instances. Every program *P* is associated with a set of *abducibles* $\mathcal{A} \subseteq \mathcal{L}$, consisting of literals which (without loss of generality) do not appear in any rule head of *P*. Abducibles may be thought of as hypotheses that can be used to extend the current theory, in order to provide hypothetical solutions or possible explanations for given queries.

2.2 Preferring Abducibles

An abducible can be assumed only if it is a considered one, i.e. it is expected in the given situation, and moreover there is no expectation to the contrary [Dell'Acqua and Pereira, 2005, Dell'Acqua and Pereira, 2007].

$$consider(A) \leftarrow expect(A), not\ expect\_not(A).$$

The rules about expectations are domain-specific knowledge contained in the theory of the agent, and effectively constrain the hypotheses (and hence scenaria) which are available.

To express preference criteria among abducibles, we consider an extended first order language $\mathcal{L}^*$. A preference atom in $\mathcal{L}^*$ is one of the form $a \lhd b$, where a and b are abducibles. $a \lhd b$ means that the abducible a is preferred to the abducible b. A preference rule in $\mathcal{L}^*$ is one of the form:

$$a \lhd b \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where $a \lhd b$ is a preference atom and every $L_i (1 \leq i \leq t)$ is a domain or preference literal over $\mathcal{L}^*$.

Although the program transformation previously detailed in [Dell'Acqua and Pereira, 2005, Dell'Acqua and Pereira, 2007] accounted only for the possibility of mutually exclusive abducibles, we have extended the definition to allow for sets of abducibles, so we can generate *abductive stable models* [Dell'Acqua and Pereira, 2005, Dell'Acqua and Pereira, 2007] having more than a single abducible.

In fact, these preference rules can be compiled into the first language $\mathcal{L}$. Basically, each preference rule of the above form can be converted to the following defeating rule:

$$expect\_not(b) \leftarrow L_1, \ldots, L_t, \\ consider(a), b, not\ a \ \ (t \geq 0)$$

with the declarative reading that in all the models where a is considered, if b is abduced, then a must also be abduced. If the two abducibles are mutually exclusive, then the preference will always defeat one in favor of the other. However if both abducibles can coexist in the same model, then we at least guarantee that in all models where b is present, a is also present.

For a more detailed explanation of the adapted transformation, please consult the ACORDA project page, mentioned in the previous footnote.

3 PROSPECTIVE LOGIC AGENTS

We now present the abstract procedure driving evolution of a prospective logic agent. Although it is still too early to present a complete formal LP semantics to this combination of techniques and methodologies, as the implemented system is undergoing constant evolution and revision, it is to be expected that such a formalization will arise in the future, since the proposed architecture is built on top of logically grounded and semantically well-defined LP components. The procedure is illustrated in Figure 1, and is the basis for the implemented ACORDA system, which we will detail in Section 5.

Each prospective logic agent has a knowledge base containing some initial program over $\mathcal{L}^*$. The problem of prospection is then one of finding abductive extensions to this initial theory which are both:

- $relevant$ under the agent's current desires and goals
- $preferred$ extensions w.r.t. the preference rules in the knowledge base

We adopt the following definition for the relevant part of a program P under a literal L:

Definition 1 Let L, B, C be literals in $\mathcal{L}^*$. We say L directly depends on B iff B occurs in the body of some rule in P with head L. We say L depends on B iff L directly depends on B or there is some C such that L directly depends on C and C depends on B. We say that $Rel_L(P)$, the relevant part of P, is the logic program constituted by the set of all rules of P with head L or some B on which L depends on.

Given the above definition, we say that an abductive extension Δ of P (i.e. $\Delta \subseteq \mathcal{A}_P$) is $relevant$ under some query G iff all the literals in Δ belong to $Rel_G(P \cup \Delta)$. The first step thus becomes to select the desires and goals that the agent will possibly attend to during the prospective cycle.

3.1 Goals and Observations

Definition 2 An observation is a quaternary relation amongst the *observer*; the *reporter*; the observation name; and the *truth value* associated with it.

$$observe(Observer, Reporter, Observation, Value)$$

Observations can stand for actions, goals or perceptions. The $observe/4$ literals are meant to represent observations reported

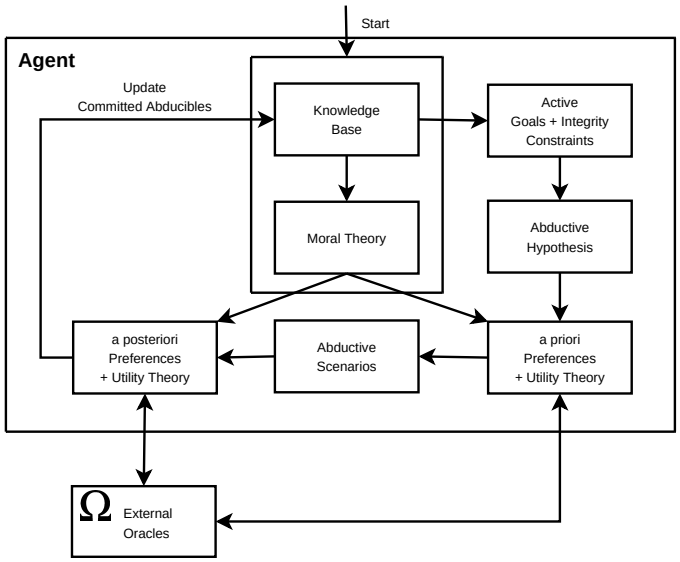

Figure 1: Prospective agent cycle.

by the environment into the agent or from one agent to another, which can also be itself (self-triggered goals). We also introduce the corresponding $on\_observe/4$ literal, which we consider as representing active goals or desires that, once triggered, cause the agent to attempt their satisfaction by launching the queries standing for the observations contained inside.

The prospection mechanism then polls for $on\_observe/4$ literals satisfied under the initial theory of the agent. In an abstract representation, we are interested in those $on\_observe/4$ literals which belong to the Well-Founded Model of the evolving logic program at the current knowledge state.

Definition 3 The set of active goals of initial program P is:

$$Goals(P) = \{G : on\_observe(agent, agent, G, true) \in WFM(P)\}$$

By adopting the more skeptic Well-Founded Semantics at this stage, we guarantee a unique model for the activation of $on\_observe/4$ literals. It should be noted that there can be many situations where more than one active goal is derived under the current knowledge theory of the agent. Since we are dealing with the combinatorial explosion of all possible abductive extensions, it is possible that, even if no combination of abducibles satisfies the entire conjunction of active goals, that at least a subset of those goals will be satisfied in some models. In order to allow for the generation of all these possible scenaria, we actually transform active goals into $tentative$ queries, encoded in the following form:

$$try(G) \leftarrow G \qquad try(G) \leftarrow not\ try\_not(G)$$
$$try\_not(G) \leftarrow not\ try(G)$$

In this way, we guarantee that computed scenaria will provide all possible ways to satisfy the conjunction of desires, or possible subsets of desires, allowing us then to apply selection rules to qualitatively determine which abductive extensions to adopt based on the relative importance or urgency of activated goals. Integrity constraints are also considered, so as to ensure the agent always performs transitions into valid evolution states.

These can also be triggered on the basis of possible abductive scenaria, as the next example will demonstrate.

Example 1 Prospecting the future allows for taking action before some expected scenaria actually happen. This is vital in taking proactive action, not only to achieve our goals, but also to prevent, or at least account for, catastrophic futures.

Consider a scenario where weather forecasts have been transmitted foretelling the possibility of a tornado. It is necessary to deal with this emergency beforehand, and take preventive measures before the event actually takes place. A prospective logic program that could deal with this scenario is encoded below.

$$\bot \leftarrow consider(tornado),\\ \quad not\ deal\_with\_emergency(tornado)$$

$$expect(tornado) \leftarrow weather\_forecast(tornado)\\ deal\_with\_emergency(tornado) \leftarrow\\ \quad consider(decide\_board\_up\_house)$$

$$expect(decide\_board\_up\_house) \leftarrow consider(tornado)\\ \bot \leftarrow decide\_board\_up\_house,\ not\ boards\_at\_home,\\ \quad not\ go\_buy\_boards$$

The first sentence expresses that, in case a tornado scenario is considered, the program should deal with the emergency. A possible way to deal with this emergency is deciding to board up the house. This hypothesis is only made available in the event of a tornado, since we do not want in this case to account for this decision in any other situation (we could change the corresponding $expect/1$ rule to state otherwise). The weather forecast brings about that a tornado is expected, and there being no contrary expectation to this scenario, the above program presents two possible predictions about the future. In one of the scenaria, the tornado is absent, but in the scenario where it is actually confirmed, the decision to board up the house follows as a necessity.

If we commit to the decision of boarding up the house, by assuming the tornado scenario is more relevant, and we do not have boards at home, it is necessary that we go and buy the boards. This is reflected by the second integrity constraint, which in fact would launch a subgoal for buying boards. As such, even if no goals were active, the possibility of considering certain scenaria can trigger integrity constraints, and also contextual abducibles which may in turn be used, once they are confirmed, to support activation of other goals.

3.2 Generating Scenaria

Once the set of active goals for the current state is known, the next step is to find out which are the relevant abductive extensions which are considered in the situation. They can be found by reasoning backwards from the goals into abducibles which come up under $consider/1$ literals. Each abducible represents a choice: the agent can either assume it true, or assume it false, meaning that it may potentially face a number of interpretations equal to all possible combinations of relevant abducibles. In practice, the combinatorial explosion of possible interpretations is contained and made tractable by a number of factors.

To begin with, the simple fact that all abducibles are constrained to the relevant part of the program under the active goals already leaves all the irrelevant abducibles out of the generation of scenaria. Secondly, the context-dependent rules presented in Section 2.2 for considering abducibles further excludes those abducibles which are not relevant to the actual situation of the agent. Furthermore, it is often the case that available abducibles are contradictory, i.e. considering an abducible actually precludes considering another one, for instance, when choosing between drinking coffee or drinking tea [Dell'Acqua and Pereira, 2005, Dell'Acqua and Pereira, 2007]. Finally, this step includes the application of a priori preferences in the form of contextual preference rules among the available abducibles.

In each possible interpretation, or scenario, thus generated, we also reason forwards from abducibles to obtain the relevant consequences of actually committing to each of them. Each abductive stable model is characterized by the abducible choices contained in it, but is in fact a whole model of the program sent to it. Information about each of the models will then be used to enact preferences over the scenarios *a posteriori*, taking into account the consequences in each scenario

3.3 Inspection Points

Consider a situation where an agent is thirsty and is deciding between having coffee and tea for a drink. In the scenario where the agent chooses to have tea, it can also consider the possibility of having scones. We say that the abducible $scones$ is constrained to scenarios where the abducible tea is present, but it cannot determine by itself the abduction of tea. The reasoning is that having tea is not a subgoal to solving scones, but rather having scones is a possibility which is open to consideration after we actually commit to drinking tea.

This semantics is not reducible to any of the previously presented constructs, so we introduce a new predicate, $inspect(X)$ for any domain literal $X \in \mathcal{L}$, intuitively meaning: solve X but do not allow any abductions to be triggered in any subgoals of X. X and its subgoals can merely consume abductions performed elsewhere. This predicate can be used to extract side-effects over generated models, but without interfering in model generation itself.

We present below the tea and scones example codified using the new $inspect/1$ predicate:

$drink \leftarrow tea$
$drink \leftarrow coffee$

$expect(tea)$
$expect(coffee)$
$expect\_not(coffee) \leftarrow blood\_pressure\_high$
$expect(scones) \leftarrow inspect(tea)$

$coffee \triangleleft tea \leftarrow sleepy$

$tea \leftarrow consider(tea)$
$coffee \leftarrow consider(coffee)$
$scones \leftarrow consider(scones)$

$\bot \leftarrow thirsty,\ not\ drink$

As described, the abducible $scones$ is only expected in scenaria where tea is a side-effect, but does not by itself provoke the abduction of tea.

3.4 Preferring a posteriori

Once each possible scenario is actually obtained, there are a number of different strategies which can be used to choose which of the scenaria leads to more favorable consequences. A possible way to achieve this was first presented in [Poole, 1997], using numeric functions to generate a quantitative measure of utility for each possible action. We allow for the application of a similar strategy, by making a priori assignments of probability values to uncertain literals and utilities to relevant consequences of abducibles. We can then obtain a posteriori the overall utility of a model by weighing the utility of its consequences by the probability of its uncertain literals. It is then possible to use this numerical assessment to establish a preorder among remaining models.

Although such numerical treatment of a posteriori preferences can be effective in some situations, there are occasions where we do not want to rely on probability and utility alone, especially if we are to attribute tasks of responsibility to such autonomous agents. In particular, it may become necessary to endow such agents with a set of behaviour precepts which are to be obeyed at all times, no matter what the quantitative assessments may say. This is the role of the moral theory presented in the figure. Although being clearly outside the scope of the presented work, we regard it as a growing concern which must be weighed as more intelligent and autonomous agents are built and put to use. A more detailed analysis of this moral perspective can be found in [Pereira and Saptawijaya, 2007].

Both qualitative and quantitative evaluations of the scenarios can be greatly improved by merely acquiring additional information to make a final decision.We next consider the mechanism that our agents use to question external systems, be they other agents, actuators, sensors or other procedures. Each of these serves the purpose of an *oracle*, which the agent can probe through observations of its own, of the form

$$observe(agent, oracle\_name, query, Value) \leftarrow$$
$$oracle, L_1, \ldots, L_t \quad (t \geq 0)$$

representing that the agent is performing the observation $query$ on the oracle identified by $oracle\_name$, whenever oracle observations are allowed (governed by the reserved toggle literal $oracle$) and given that domain literals $L_1, \ldots, L_t$ hold in the current knowledge state. Following the principle of parsimony, it is not desirable that the oracles be consulted ahead of time in any situation. Hence, the procedure starts by using its available local knowledge to generate the preferred abductive scenaria (i.e. the toggle is turned off), and then extends the search to include available oracles, by toggling $oracle$ on. Each oracle mechanism may in turn have certain conditions specifying whether it is available for questioning. At the next iteration, this toggle is turned off, as more consequences will be computed using the additional information.

Whenever the agent acquires additional information to deal with a problem at hand, it is possible, and even likely, that ensuing side-effects may affect its original search. Some considered abducibles may now be disconfirmed, but it is also possible that some new abducibles which were previously unavailable are now triggered by the information obtained by the oracle observations. To ensure all possible side-effects are accounted for, a second round of prospection takes place, by relaunching the whole conjunctive query. Information returned from the oracle may change the preferred scenaria previously computed, which can in turn trigger new questions to oracles, and so on, in an iterated process of refinement, which stops if no changes to the models have been enacted, and there are no new oracle questions to perform, or user updates to execute.

Even after extending the search to allow for experiments, it may still be the case that some abducibles are tied in competition to explain the active goals, e.g. if some available oracle was unable to provide a crucial deciding experiment. In this case, the only remaining possible action is to branch the simulation into two or more possible update sequences, each one representing an hypothetical world where the agent simulates commitment to the respective abducible. This means delaying the choice, and keeping in mind the evolution of the remaining scenaria until they are gradually defeated by future updates, or somehow a choice is enforced. Exactly how these branches are kept updated and eventually eliminated is not trivial, and this is why we purposefully leave undefined the procedure controlling the evolution of these branching prospective sequences. Another interesting possibility would be to consider those abductions common to all the models and commit to them, in order to prune some irrelevant models while waiting for future updates to settle the matter.

3.5 Prospective procedure

We conclude this section by presenting the full abstract procedure defining the cycle of a prospective logic agent.

Definition 4 Let P be an evolving logic program, representing the knowledge theory of an agent at state S. Let $oracle$ be the propositional atom used as a toggle to restrict access to additional external observations. A prospective evolution of P is a set of updates onto P computed by the following procedure:

1. Let O be the (possibly empty) set of all on_observe/4 atoms which hold at S.

2. Compute the set of stable models of the residual program derived by the evaluation of the conjunction $Q = \{G_1, \ldots, G_n, not \bot\}, n \geq 0$, where each G_i represents the goal contained in a distinct observe/4 literal obtained from the corresponding on_observe/4 in O.

3. If the set contains a single model, update the abductive choices characterizing the model onto P as facts, toggle the $oracle$ off and stop.

4. Otherwise, if $oracle$ currently holds and no new information from the oracles or from the scenaria is derived, for each abductive stable model M_i create a new branching evolution sequence P_i and update the abductive choices in M_i onto P_i. Execute the procedure starting from step 1 on each branching sequence P_i.

5. Otherwise, toggle the $oracle$ on and return to 2.

4 MODELLING PROSPECTIVE LOGIC AGENTS

4.1 Accounting for Emergencies

Example 2 Consider the emergency scenario in the London underground [Kowalski, 2006b], where smoke is observed, and we want to be able to provide an explanation for this observation. Smoke can be caused by fire, in which case we should also consider the presence of flames, but smoke could also be caused by tear gas, in case of police intervention. The tu literal in observation values stands for true or undefined.

$$
\begin{array}{l}
smoke \leftarrow consider(fire) \\
smoke \leftarrow consider(tear\_gas) \\
flames \leftarrow consider(fire) \\
eyes\_cringing \leftarrow consider(tear\_gas)
\end{array}
$$

$$
\begin{array}{l}
expect(fire) \\
expect(tear\_gas) \\
fire \triangleleft tear\_gas
\end{array}
$$

$$
\begin{array}{l}
\bot \leftarrow observation(smoke),\ not\ smoke \\
observation(smoke)
\end{array}
$$

$$
\begin{array}{l}
\bot \leftarrow flames,\ not\ observe(program, user, flames, tu) \\
\bot \leftarrow eyes\_cringing, \\
\qquad not\ observe(program, user, eyes\_cringing, tu)
\end{array}
$$

This example illustrates how an experiment can be derived in lieu of the consequences of an abduction. In order for fire to be abduced, we need to be able to confirm the presence of flames, which is a necessary consequence, and hence we trigger the observation to confirm flames, expressed in the second integrity constraint. Only in case this observation does not disconfirm flames are we allowed to abduce fire.

4.2 Automated Diagnosis

Prospective logic programming has a direct application in automated diagnosis scenaria, as was previously shown in [Lopes and Pereira, 2006]. Another illustration is that of a use case in ongoing research on diagnosis of self-organizing industrial manufacturing systems [Barata et al., 2007].

Example 3 Consider a robotic gripper immersed in a collaborative assembly-line environment. Commands issued to the gripper from its controller are updated to its evolving knowledge base, as well as regular readings from the sensor. After expected execution of its commands, diagnosis requests by the system are issued to the gripper's prospecting controller, in order to check for abnormal behaviour. When the system is confronted with multiple possible diagnosis, requests for experiments can be asked of the controller. The gripper can have three possible logical states: open, closed or something intermediate. The available gripper commands are simply $open$ and $close$. This scenario can be encoded as the initial prospective program below.

$$
\begin{array}{l}
open \leftarrow request\_open,\ not\ consider(abnormal(gripper)) \\
open \leftarrow sensor(open),\ not\ consider(abnormal(sensor))
\end{array}
$$

$$
\begin{array}{l}
intermediate \leftarrow request\_close, manipulating\_part, \\
\qquad\qquad not\ consider(abnormal(gripper)), \\
\qquad\qquad not\ consider(lost\_part) \\
intermediate \leftarrow sensor(intermediate), \\
\qquad\qquad not\ consider(abnormal(sensor))
\end{array}
$$

$$
\begin{array}{l}
closed \leftarrow request\_close,\ not\ manipulating\_part, \\
\qquad not\ consider(abnormal(gripper)) \\
closed \leftarrow sensor(closed) \\
\qquad not\ consider(abnormal(sensor))
\end{array}
$$

$$
\begin{array}{ll}
\bot \leftarrow open, intermediate & \bot \leftarrow open, closed \\
\bot \leftarrow closed, intermediate &
\end{array}
$$

$$
\begin{array}{l}
expect(abnormal(gripper)) \\
expect(abnormal(sensor)) \\
expect(lost\_part) \leftarrow manipulating\_part \\
expect\_not(abnormal(sensor)) \leftarrow \\
\qquad manipulating\_part, \\
\qquad observe(system, gripper, ok(sensor), true)
\end{array}
$$

$$
\begin{array}{l}
observe(system, gripper, Experiment, Result) \leftarrow \\
\qquad oracle, test\_sensor(Experiment, Result)
\end{array}
$$

$$
\begin{array}{l}
abnormal(gripper) \triangleleft abnormal(sensor) \leftarrow \\
\qquad request\_open,\ not\ sensor(open), \\
\qquad not\ sensor(closed) \\
lost\_part \triangleleft abnormal(gripper) \leftarrow \\
\qquad observe(system, gripper, ok(sensor), true), \\
\qquad sensor(closed) \\
abnormal(gripper) \triangleleft lost\_part \leftarrow \\
\qquad not\ (lost\_part \triangleleft abnormal(gripper))
\end{array}
$$

For each possible logical state, we encode rules predicting

that state from requested actions and from provided sensor readings. We consider that execution of actions may fail, or that the sensor readings may be abnormal. There are also situations where mechanical failure did not occur and sensor readings are also correct, but there was some other failure, like losing the part the robot was manipulating, by dropping it.

In this case, there is an available experiment to test whether the sensor is malfunctioning, but resorting to it should be avoided as much as possible, as it will imply occupying additional resources from the assembly-line coalition. As expected, evaluation is context-dependent on the situation. Consider this illustrative update set:

$$U = \{manipulating\_part, request\_close, sensor(closed)\}.$$

It represents the robot in the process of manipulating some part, receiving an order to close the gripper in order to grab it, but the sensor reporting the gripper is completely closed. This violates an integrity constraint, as the gripper should be in an intermediate state, taking hold of the part. At the start of a diagnosis, three abductive hypotheses are expected and considered,

$$\mathcal{A}_P = \{lost\_part, abnormal(gripper), abnormal(sensor)\}.$$

Without further information, abducible *abnormal(gripper)* is preferred to *lost_part*, but still no single scenario has been determined. Activating oracle queries, the system finds the experiment to test the sensor. If it corroborates closed, not only the abducible *abnormal(sensor)* is defeated, but also *abnormal(gripper)*, since *lost_part* is preferred. However, failure to confirm the sensor reading would result in no single scenario being abduced for this situation, and other measures would have to be taken.

4.3 Encoding Actions

Another interesting possibility in future prospection is to consider the dynamics of actions. To perform an action, a prospective agent needs not just to consider the necessary preconditions for executing it in the present, but also to look ahead at the consequences it will entail in a future state. These two verifications take place on different reasoning moments. While the preconditions of an action can be evaluated immediately when collecting the relevant abducibles for a given knowledge state, its postconditions can only be taken into consideration after the model generation, when the consequences of hypothetically executing an action are known.

The execution of an action can be encoded in EVOLP by means of *assert/1* rules, of the form:

$$assert(A) \leftarrow L_1, \ldots, L_t \quad (t \geq 0)$$

where *A* is a domain atom representing the name of the action and $L_1, \ldots, L_t$ are domain literals representing the preconditions for the action. The preconditions can themselves contain other $assert/1$ literals in their bodies, allowing lookahead into future updates. The postconditions of a given action can be encoded as integrity constraints on the name of the action and will be triggered during generation of the stable models.

Example 4 Consider an agent choosing an activity in the afternoon. It can either go to the beach, or to the movies, but not both, and it can only go see a movie after buying tickets to it. The abducibles in this case are $\mathcal{A}_P = \{go\_to\_beach, go\_to\_movies\}$. There is a single integrity constraint stating that tickets cannot be bought without money. In ACORDA syntax:

$$afternoon\_activity \leftarrow assert(beach)$$
$$afternoon\_activity \leftarrow assert(movies)$$

$$assert(beach) \leftarrow consider(go\_to\_beach)$$
$$assert(movies) \leftarrow tickets$$
$$assert(tickets) \leftarrow consider(go\_to\_movies)$$

$$expect(go\_to\_beach)$$
$$expect(go\_to\_movies)$$
$$\bot \leftarrow tickets, not\ money$$

The abduction of either go_to_beach or go_to_movies fulfills, respectively, the preconditions for the action $beach$ and the action $tickets$. The consequence of buying the tickets is that the precondition for going to the movies is fulfilled. However, that consequence may also trigger the integrity constraint if the agent does not have money. Fortunately, by simulating the consequences of actions in the next state, the agent can effectively anticipate that the constraint will be violated, and proceed to choose the only viable course of action, that is going to the beach.

5 IMPLEMENTING THE ACORDA SYSTEM

The basis for the developed ACORDA system is an EVOLP meta-interpreter on which we can evaluate literals for truth according to three- and two-valued semantics. Both this meta-interpreter and the remaining components were developed on top of XSB Prolog, an extensively used and stable LP inference engine implementation, following the Well-Founded Semantics (WFS) for normal logic programs.

The tabling mechanism [Swift, 1999] used by XSB not only provides a significant improvement in the time complexity of logic program evaluation, but also allows for extending WFS to other non-monotonic semantics. An example of this is the XASP interface (standing for XSB Answer Set Programming), which extends computation of the WFM, using Smodels [Niemelä and Simons, 1997] to compute two-valued models from the *residual program* resulting from querying the knowledge base [Castro et al.,]. This residual program is represented by delay lists, that is, the set of undefined literals for which the program could not find a complete proof, due to mutual dependencies or loops over default negation for that set of literals, detected by the XSB tabling mechanism. It is also possible to access Smodels by building up a clause store in which a normal logic program is composed, parsed and evaluated, with the computed stable models sent back to the XSB system.

This integration allows one to maintain the relevance property [Dix, 1995] for queries over our programs, something that

the Stable Models semantics does not originally enjoy. In Stable Models, by the very definition of the semantics, it is necessary to compute all the models for the whole program. Furthermore, since computation of all the models is NP-complete, it would be unwise to attempt it in practice for the whole knowledge base in a logic program, which can contain literally thousands of rules and facts and unlimited abducibles. In our system, we sidestep this issue, using XASP to compute the relevant residual program on demand, usually after some degree of transformation. Only the resulting program is then sent to Smodels for computation of possible futures. The XSB side of the computation also plays the role of an efficient grounder for rules sent to Smodels, that otherwise resorts to Herbrand base expansion, which can be considerably hastened if we can provide a priori the grounding of domain literals. Also, the stable models semantics is not cumulative [Dix, 1995], which is a prohibitive restriction when considering self-evolving logic programs, in which it is extremely useful to store previously deduced conclusions as lemmas to be reused.

6 CONCLUSIONS AND FUTURE WORK

As far as we know, the only other authors taking a similar LP approach to the derivation of the consequences of candidate abductive hypotheses are [Kowalski, 2006b, Kowalski, 2006a], and [Poole, 1997, Poole, 2000]. Both represent candidate actions by abducibles and use logic programs to derive their possible consequences, to help in deciding between them. However, they do not derive consequences of abducibles that are not actions, such as observations for example. Nor do they consider the possibility of determining the value of unknown conditions by consulting an oracle or by some other process.

Poole uses abduction, restricted to acyclic programs, to provide explanations for positive and negative goals. An explanation represents a set of independent choices, each of which is assigned a probability value. The probability of a goal can be found by considering the set of abductively generated possible worlds containing an abductive explanation for the goal. His main concern is to compute goal uncertainty, with a view to decision making, taking into account both the probabilities of the abductive assumptions and the utilities of their outcomes.

Kowalski argues that an agent can be more intelligent if it is able to reason pre-actively - that is to say, to reason forward from candidate actions to derive their possible consequences. These consequences, he recognizes, may also depend upon other conditions over which the agent has no control, such as the actions of other agents or unknown states of the environment. He considers the use of Decision Theory, like Poole, to choose actions that maximise expected utility. But he has not explored ways of obtaining information about conditions over which the agent does not have control, nor the use of preferences to make choices [Kowalski, 2007].

Compared with Poole and Kowalski, one of the most interesting features of our approach is the use of Smodels to perform a kind of forward reasoning to derive the consequences of candidate hypotheses, which may then lead to a further cycle of abductive exploration, intertwined with preferences for pruning and for directing search.

With branching update sequences we have begun to address the problem of how to arbitrarily extend the future lookahead within simulations. Independent threads can evolve on their own by commiting to surviving assumptions and possibly triggering new side-effects which will only take place after such commitment.Nevertheless, some issues in the management of these branching sequences must still be tackled, namely envolving coordination and articulation of information shared among threads belonging to a common trunk, as well as the control of the lifetime of each individual thread.

Preferences over observations are also desirable, since not every observation costs the same for the agent. For example, in the industrial manufacture example, the experiment for testing the sensor was costly, but additional and cheaper experiments could eventually be developed, and they should be preferred to the more expensive one whenever possible. Furthermore, abductive reasoning can be used to generate hypotheses of observations of events possibly occurring in the future along the lines of [Alberti et al., 2005].

Prospective LP accounts for abducing the possible means to reach an end, but the converse problem is also of great interest, that is, given the observations of a set of actions, abduce the goal that led to the selection of those actions. This would be invaluable in abducing the intentions of other agents from the sequence of actions they exhibit.

Although we are currently dealing only with prospection of the future, prospective simulations of the past can also be of interest to account for some learning capabilities based on counterfactual thought experiments. This means that we can go back to a choice point faced in the past and relaunch the question in the form "'What would happen if I knew then what I know now?"', incorporating new elements on reevaluating past dilemmas. This could allow for debugging of prospective strategies, identifying experiments that could have been done as well as alternative scenarios that could have been pursued so that in the future the same errors are not repeated.

REFERENCES

[Alberti et al., 2005] Alberti, M., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2005). Abduction with hypotheses confirmation. In *Proc. of the 19th Intl. Joint Conf. on Artificial Intelligence (IJCAI-05)*, pages 1545–1546.

[Alferes et al., 2002] Alferes, J. J., Brogi, A., Leite, J. A., and Pereira, L. M. (2002). Evolving logic programs. In et al., S. F., editor, *Procs. 8th European Conf. on Logics in Artificial Intelligence (JELIA'02)*, pages 50–61.

[Alferes et al., 2000] Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H., and Przymusinski, T. C. (2000). Dynamic updates of non-monotonic knowledge bases. *J. Logic Programming*, 45(1-3):43–70.

[Barata et al., 2007] Barata, J., Ribeiro, L., and Onori, M. (Forthcoming 2007). Diagnosis on evolvable production sys-

tems. In *Procs. of the IEEE Intl. Symp. on Industrial Electronics (ISIE'07)*, Vigo, Spain.

[Castro et al.,] Castro, L., Swift, T., and Warren, D. S. *XASP: Answer Set Programming with XSB and Smodels*. `http://xsb.sourceforge.net/packages/xasp.pdf`.

[Dell'Acqua and Pereira, 2005] Dell'Acqua, P. and Pereira, L. M. (2005). Preferential theory revision. In Pereira, L. M. and Wheeler, G., editors, *Procs. Computational Models of Scientific Reasoning and Applications*, pages 69–84.

[Dell'Acqua and Pereira, 2007] Dell'Acqua, P. and Pereira, L. M. (2007). Preferential theory revision (ext.). *J. of Applied Logic*.

[Dix, 1995] Dix, J. (1995). A classification theory of semantics of normal logic programs: i. strong properties, ii. weak properties. *Fundamenta Informaticae*, 22(3):227–255,257–288.

[Kakas et al., 1998] Kakas, A., Kowalski, R., and Toni, F. (1998). The role of abduction in logic programming. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press.

[Kowalski, 2006a] Kowalski, R. (2002-2006a). How to be artificially intelligent. `http://www.doc.ic.ac.uk/~rak/`.

[Kowalski, 2006b] Kowalski, R. (2006b). The logical way to be artificially intelligent. In Toni, F. and Torroni, P., editors, *Proceedings of CLIMA VI*, LNAI, pages 1–22. Springer Verlag.

[Kowalski, 2007] Kowalski, R. (2007). Private communication.

[Lopes and Pereira, 2006] Lopes, G. and Pereira, L. M. (2006). Prospective logic programming with ACORDA. In Sutcliffe, G., Schmidt, R., and Schulz, S., editors, *Procs. of the FLoC'06 Ws. on Empirically Successful Computerized Reasoning, 3rd Intl. J. Conf. on Automated Reasoning*, number 192 in CEUR Workshop Procs.

[Niemelä and Simons, 1997] Niemelä, I. and Simons, P. (1997). Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, pages 420–429, Berlin. Springer.

[Pereira and Saptawijaya, 2007] Pereira, L. M. and Saptawijaya, A. (2007). Modelling morality with prospective logic. In Neves, J. M., Santos, M. F., and Machado, J. M., editors, *Procs. 13th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'07)*, LNAI. Springer.

[Poole, 1997] Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56.

[Poole, 2000] Poole, D. (2000). Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44:5–35.

[Swift, 1999] Swift, T. (1999). Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240.