Evolution Prospection in Decision Making

Luís Moniz Pereira and Han The Anh*

Abstract

This work concerns the problem of modelling evolving prospective agent systems. Inasmuch a prospective agent [1] looks ahead a number of steps into the future, it is confronted with the problem of having several different possible courses of evolution, and therefore needs to be able to prefer amongst them to decide the best to follow as seen from its present state. First it needs a priori preferences for the generation of likely courses of evolution. Subsequently, this being one main contribution of this paper, based on the historical information as well as on a mixture of quantitative and qualitative a posteriori evaluation of its possible evolutions, we equip our agent with so-called evolution-level preferences mechanism, involving three distinct types of commitment. In addition, one other main contribution, to enable such a prospective agent to evolve, we provide a way for modelling its evolving knowledge base, including environment and course of evolution triggering of all active goals (desires), context-sensitive preferences and integrity constraints. We exhibit several examples to illustrate the proposed concepts.

Keywords: Evolving prospective agent, intelligent agent, evolution-level preferences, context-sensitive integrity constraints, abductive reasoning, logic programming, intelligent decision making techniques.

*Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa, 2829-516 Caparica, Portugal, Email: lmp@di.fct.unl.pt, h.anh@fct.unl.pt

1 Introduction

Continuous developments in logic programming (LP) language semantics which can account for evolving programs with updates [2, 3, 4] 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 consideration of possible actions and hypothetical goals emerging in the activity of other agents.

Prospective agent systems [1] address the issue of how to allow evolving agents to be able to look ahead, prospectively, into their hypothetical futures, in order to determine the best courses of evolution from their own present, and thence to prefer amongst those futures. In such systems, *a priori* and *a posteriori* preferences, embedded in the knowledge representation theory, are used for preferring amongst hypothetical futures, or scenarios. The *a priori* ones are employed to produce the most interesting or relevant conjectures about possible future states, while the *a posteriori* ones allow the agent to actually make a choice based on the imagined consequences in each scenario. ACORDA [1] is a prospective logic system that implements these features. It does so by generating scenarios, on the basis only of those preferred abductions able to satisfy agents' goals, and further selecting scenarios on the basis of the immediate side-effects such abductions have within them.

However, the above proposed preferences have only local influence, i.e. for example, immediate *a posteriori* preferences are only used to evaluate the one-state-far consequences of a single choice. They are not appropriate when evolving prospective agents want to look ahead a number of steps into the future to determine which decision to make from any

state of their evolution. Such agents need to be able to evaluate further consequences of their decisions, i.e. the consequences of the hypothetical choices abduced to satisfy their goals. Based on the historical information as well as quantitative and qualitative a posteriori evaluation of its possible evolutions, we equip an agent with a so-called evolution-level preferences mechanism.

For evolving agents, their knowledge base evolves to adapt to the outside changing environment. At each state, agents have a set of goals and desires to satisfy. They also have to be able to update themselves with new information such as new events, new rules or even change their preferences. To enable a prospective agent to evolve, we provide a way for modelling its evolving knowledge base, including the environment and course of evolution triggering of all active goals (desires), of context-sensitive preferences and of integrity constraints. To further achieve this, immediate *a posteriori* preferences are insufficient.

After deciding on which action to take, agents evolve by committing to that action. Different decision commitments can affect the simulation of the future in different ways. There are actions that, if committed to, their consequences are nevermore defeated and thus permanently affect the prospective future. There are also actions that do not have any inescapable influence on the future, i.e. committing to them does not permanently change the knowledge base, like the previously described "hard" commitments – they are "ongoing". They may be taken into account when, in some following future state, the agents need to consider some evolution-level preferences trace. Other action commitments are "temporary", i.e. merely momentary.

In addition, we specifically consider so-called inevitable actions that belong to every possible evolution. By hard committing to them as soon as possible, the agent can activate preferences that rule out alternative evolutions that are ipso facto made less relevant.

This paper is an extended version of the work [10].

The rest of the paper is organized as follows. Section 2 discusses prospective logic programs, describing the constructs involved in their design and implementation. Section 3 describes evolving prospective agents, including single-step and multiple-step look-ahead, and exhibits several examples for illustration. Section 4 provides the semantics for evolving prospective programs on top of the semantics for Abdual over Well-Founded semantics. Based on that, section 5 briefly describes the implementation of evolving prospective logic programming on top of Abdual. The paper ends with conclusions and directions for the future...

2 Prospective Logic Programming

Prospective logic programming enables an evolving program to look ahead prospectively into its possible future states, which may include rule updates, and to prefer among them to satisfy goals [1]. This paradigm is particularly beneficial to the agents community, since it can be used to predict an agent's future by employing the methodologies from abductive logic programming [5, 7] in order to synthesize, prefer and maintain abductive hypotheses.

We next describe constructs involved in our design and implementation of prospective logic agents and their preferred and partly committed but still open evolution, on top of Abdual [6] - a XSB-Prolog implemented system which allows computing abductive solutions for a given query.

2.1 Language

Let $\mathcal{L}$ be a first order language. A domain literal in $\mathcal{L}$ is a domain atom A or its default negation *not* A. The latter is used to express that the atom is false by default (Closed World Assumption). A domain rule in $\mathcal{L}$ is a rule of the form:

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

where A is a domain atom and $L_1, \ldots, L_t$ are domain literals. An integrity constraint in $\mathcal{L}$ is a rule with an empty head. A (logic) program P over $\mathcal{L}$ is a set of domain rules and integrity constraints, standing for all their ground instances.

2.2 Preferring abducibles

Every program P is associated with a set of abducibles $\mathcal{A} \subseteq \mathcal{L}$. These, and their default negations, can be seen as hypotheses that provide hypothetical solutions or possible explanations to given queries. Abducibles can figure only in the body of program rules.

An abducible A can be assumed only if it is a considered one, i.e. if it is expected in the given situation, and, moreover, there is no expectation to the contrary [9].

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

The rules about expectations are domain-specific knowledge contained in the theory of the program, and effectively constrain the hypotheses available in a situation. Handling preferences over abductive logic programs has several advantages, and allows for easier and more concise translation into normal logic programs (NLP) than those prescribed by more general and complex rule preference frameworks. The advantages of so proceeding stem largely from avoiding combinatory explosions of abductive solutions, by filtering irrelevant as well as less preferred abducibles [8].

To express preference criteria among abducibles, we envisage an extended language $\mathcal{L}^\star$. A preference atom in $\mathcal{L}^\star$ is of the form $a \lhd b$, where a and b are abducibles. It means that if b can be assumed (i.e. considered), then $a \lhd b$ forces a to be assumed too if it can. A preference rule in $\mathcal{L}^\star$ is of the form:

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

where $L_1, ..., L_t$ are domain literals over $\mathcal{L}^\star$. This preference rule can be coded as follows:

$$expect\_not(b) \leftarrow L_1, ..., L_n, not\ expect\_not(a), expect(a), not\ a$$

In fact, if b is considered, the consider–rule for abducible b requires $expect\_not(b)$ to be false, i.e. every rule with the head $expect\_not(b)$ cannot have a true body. Thus, $a \lhd b$, that is if its body in the preference rule holds, and if a is expected, and not counter-expected, then a must be abduced so that this particular rule for $expect\_not(b)$ also fails, and the abduction of b may go through if all the other rules for $expect\_not(b)$ fail as well.

A priori preferences are used to produce the most interesting or relevant conjectures about possible future states. They are taken into account when generating possible scenarios (abductive solutions), which will subsequently be preferred amongst each other a posteriori.

Example 1 *Consider a situation where Claire drinks either tea or coffee (but not both). Suppose that Claire prefers coffee over tea when sleepy, and doesn't drink coffee when she has high blood pressure. This situation is described by the program with abducibles $coffee$ and tee:*

$$\begin{array}{ll} drink \leftarrow tea & drink \leftarrow coffee \\ expect(tea) & expect(coffee) \\ expect\_not(coffee) \leftarrow blood\_high\_pressure & \\ \leftarrow tea, coffee & \\ coffee \lhd tea \leftarrow sleepy & \end{array}$$

This program has two abductive solutions, one with *tea* the other with $coffee$. Adding literal *sleepy* triggers the only *a priori preference* in the program, which defeats the solution where only *tea* is present (due to the impossibility of simultaneously abducing coffee). If later we add *blood_pressure_high*, coffee is no longer expected, and the transformed preference rule no longer defeats the abduction of *tea* which then becomes the single abductive solution, despite the presence of *sleepy*.

2.3 A posteriori Preferences

Having computed possible scenarios, represented by abductive solutions, more favorable scenarios can be preferred a posteriori. Typically, *a posteriori* preferences are performed by evaluating consequences of abducibles in abductive solutions. An *a posteriori* preference has the form:

$$A_i \ll A_j \leftarrow holds\_given(L_i, A_i),\ holds\_given(L_j, A_j)$$

where A_i, A_j are abductive solutions and L_i, L_j are domain literals. This means that A_i is preferred to A_j

a posteriori if L_i and L_j are true as the side-effects of abductive solutions A_i and A_j, respectively, without any further abduction. Optionally, in the body of the preference rule there can be any Prolog predicate used to quantitatively compare the consequences of the two abductive solutions.

2.4 Active Goals and Context Sensitive Integrity Constraints

In each cycle of its evolution the agent has a set of active goals or desires. We introduce the *on_observe*/1 predicate, which we consider as representing active goals or desires that, once triggered by the observations figuring in its rule bodies, cause the agent to attempt their satisfaction by launching the queries standing for them. The rule for an active goal AG is of the form:

$$on\_observe(AG) \leftarrow L_1, ..., L_t \ (t \geq 0)$$

where L_1,...,L_t are domain literals. During evolution, an active goal may be triggered by some events, previous commitments or some history-related information. We differentiate events that have temporary influence, i.e. affect only the current cycle and thus are entered into its knowledge base as facts and removed when the influence is finished, from ones that have permanent influence, i.e. affect every cycle issuing from the current one and thus are entered to the knowledge base as facts and stay there forever. Respectively, we provide two predicates, *event/1* and *asserts/1*.

When starting a cycle, the agent collects its active goals by finding all the $on\_observe(AG)$ that hold under the initial theory without performing any abduction, then finds abductive solutions for their conjunction.

Context sensitive integrity constraints When finding abductive solutions, all integrity constraints in the knowledge base must be satisfied. However, when considering an evolving agent, there is a vital need to be able to code integrity constraints dependent on time points and external changing environment. A context sensitive integrity constraint with the name $icName$ and a non-empty context is coded by using an active goal as follows:

$$on\_observe(not\ icName) \leftarrow L_1, ..., L_n \ (t \geq 0)$$

$$icName \leftarrow icBody$$

where L_1,...,L_t are domain literals which represent the triggering context of the integrity constraint. Whenever the context is true, the active goal $not\ icName$ must be satisfied, which implies that the integrity constraint $\leftarrow icBody$ must be satisfied. When the context is empty ($t = 0$) the integrity constraint becomes a usual one which always must be satisfied.

2.5 Levels of commitment

Each prospective cycle is completed by registering any surviving abductive solutions (represented by their abducibles) into the knowledge base and moving to the next cycle of evolution. Committing to each alternative abductive solution will create a new branch of the so-called evolution tree. The history of the evolution is kept by setting a time stamp for the abducibles that the agent commits to in each cycle.

As a program is evolving, the commitment can affect the future in different ways. Based on their influence, we classify commitments in three categories. Firstly, there are abducibles, representing actions or other options that, after committed to in a state, will not be subsequently defeated, i.e. a commitment to reverse the committed to actions is not allowed. This kind of commitment inscribes a permanent consequence on the future and therefore plays the role of a fact in the knowledge base for all future evolution states issuing from that state. Commitments of this sort are called *hard*. In addition, there are commitments that, when committed to in a state, can nevertheless be defeated by committing to their opposite abducibles at some future state, but will keep on affecting the future (by inertia) up until then. Commitments of this kind are called *ongoing*. Lastly, the weakest kind of commitments are those immediately withdrawn in the following state and so have direct influence only on the transition from the current state. They can have indirect influence when in

some future state the history of the evolution needs to be taken into account. We call this kind *temporary*.

3 Evolving prospective agents

Informally, an evolution of a prospective agent is a sequence of time stamped sets of commitments at each cycle of the evolution. The agent self-commits to abducibles, which are used to code available and preferred decision choices on all manner of options. Depending on the capabilities and need, at each time point in the evolution the agent acts just to satisfy the active goals and integrity constraints at hand, or needs to look ahead a number of steps into the future in order to satisfy its long-term and context triggered goals and constraints in a prospective way, taking into account its possible futures and evolution-sensitive reachable decision choices.

3.1 Single-step prospective agent

Each cycle ends with the commitment of the agent to an abductive solution. Alternative commitments can be explored by searching the space of evolutions.

Example 2 *Suppose agent John is going to buy an air ticket for traveling. He has two choices, either buying a saver or a flexible ticket. He knows the flexible one is expensive, but, if he has money, he does not wish a saver ticket because, if he bought it, he would not be able to change or return it in any circumstance. The saver ticket is one that, when committed to, the reverse action of returning is not allowed (a hard commitment thus). However, if John does not have much money, he is not expected to buy something expensive. Later, waiting for the flight, John finds out that his mother is ill. He wants to stay at home to take care of her, thus needing to cancel the ticket. This scenario can be coded as in Figure 1.*

Line 1 is the declaration of program abducibles, and of which of these are ongoing and hard commitments. The abducibles in the *abds/1* predicate not declared as ongoing or hard are by default temporary. Line 2 says there is unconditional expectation for each abducible declared.

```
1. abds([saver_ticket/0, flexible_ticket/0,
         cancel_ticket/0, lose_money/0]).
   ongoing_commitment([flexible_ticket]).
   hard_commitment([saver_ticket]).
2. expect(saver_ticket). expect(flexible_ticket).
   expect(cancel_ticket). expect(lose_money).
3. on_observe(ticket) <- travel.
   ticket <- saver_ticket.
   ticket <- flexible_ticket.
4. expensive(flexible_ticket).
5. expect_not(saver_ticket) <- have_money.
6. expect_not(X) <- empty_pocket, expensive(X).
7. empty_pocket <- buy_new_car.
   have_money   <- not empty_pocket.
8. on_observe(stay_home) <- mother_ill.
9. stay_home <- cancel_ticket.
   stay_home <- lose_money.
10.change_ticket <- mother_ill.
   on_observe(not saver_ticket_ic)
                <- change_ticket.
   on_observe(not cancel_ticket_ic)
                <- change_ticket.
   saver_ticket_ic  <- saver_ticket,
                       cancel_ticket.
   cancel_ticket_ic <- cancel_ticket, ticket.
11.Ai << Aj <- holds_given(cancel_ticket, Ai),
               holds_given(lose_money, Aj).
```

Figure 1: Ticket example

When John wants to travel, specified by entering *event(travel)*, i.e. the fact *travel* is temporarily added, which, in turn, triggers the only active goal *ticket*. *empty_pocket* is false and *have_money* is true, hence there is expectation to the contrary of *saver_ticket* but not of $flexible\_ticket$ (lines 5-6). Thus, there is only one abductive solution: $[flexible\_ticket]$. The cycle ends by committing to this abductive solution. Since $flexible\_ticket$ is an ongoing commitment, it will be added in every abductive solution of the following cycles until John knows that his mother is ill, entering $event(mother\_ill)$. The only active goal *stay_home* now needs to be satisfied.

In addition, the event mother being ill triggers *saver_ticket_ic* and *cancel_ticket_ic*, context-sensitive integrity constraints in line 10. There

is no expectation to the contrary of *cancel_ticket* and *lose_money*, and the ongoing commitment *flexible_ticket* is defeated, there being now three minimal abductive solutions: [*cancel_ticket*, *not flexible_ticket*], [*lose_money*, *not flexible_ticket*], [*lose_money*, *not cancel_ticket*].

In the next stage, *a posteriori* preferences are taken into account. Considering the only *a posteriori* preference in line 11, the two abductive solutions that include *lose_money* are ruled out since they lead to the consequence *lose_money*, which is less preferred than the one that leads to *cancel_ticket*.

In short, agent John bought a flexible ticket to travel, but later he can cancel the ticket to stay at home to take care of his mother because the flexible ticket is a defeasible ongoing commitment.

Next consider the same initial situation but suppose John just bought a new car, by entering *asserts*(*buy_new_car*). *empty_pocket* becomes true and *have_money* becomes false. Hence there is expectation to the contrary for *flexible_ticket* (line 7) and no expectation to the contrary for *saver_ticket* (line 6). Therefore, the only abductive solution is [*saver_ticket*]. Since *saver_ticket* is a hard commitment, it is not defeated and later on, during evolution, it will always be added to every abductive solution. Even when the mother is ill, *saver_ticket_ic* will prevent having *cancel_ticket* (line 10). Thus, the only abductive solution is the one including *lose_money*.

In short, John made a hard commitment by buying a saver ticket, and later on, when his mother is ill, he must relinquish the ticket and lose money to stay at home.

Example 3 (Sophie's choice) *One of the most discussed cases where the same moral precept gives rise to conflicting obligations is taken from William Styrons Sophies Choice [16]. Sophie and her two children are at a Nazi concentration camp. A guard confronts Sophie and tells her that one of her children will be allowed to live and one will be killed. But it is Sophie who must decide which child will be killed. Sophie can prevent the death of either of her children, but only by condemning the other to be killed. The guard makes the situation even more excruciating by informing Sophie that if she chooses neither, then both will be killed. With this added factor, Sophie has a morally compelling reason to choose one of her children. But for each child, Sophie has an apparently equally strong reason to save him or her. Thus the same moral precept gives rise to conflicting obligations. The described scenario can be coded with the program given in Figure 2.*

```
abds([letting_both_die/0,kill/1,flip_a_coin/0]).
1. expect(kill(_)).
   expect(flip_a_coin).
   expect(letting_both_die).
2. on_observe(decide) <- sophie_choice.
3. decide <- letting_both_die,
             not kill, not flip.
   decide <- choose, not flip.
   decide <- flip.
4. choose <- kill(child_1), not kill(child_2).
   choose <- kill(child_2), not kill(child_1).
   kill <- kill(child_1).
   kill <- kill(child_2).
   flip   <- flip_a_coin.
5. expect_not(kill(C)) <- special_reason(C).
6. expect_not(flip_a_coin) <-
             special_reason(child_1),
             not special_reason(child_2).
   expect_not(flip_a_coin) <-
             special_reason(child_2),
             not special_reason(child_1).
7. die(2) <- letting_both_die.
   die(1) <- choose.
   die(1) <- flip.
8. pr(feel_guilty, 1) <- kill(X).
   pr(feel_guilty, 0.5) <- flip_a_coin.
9. Ai << Aj <- holds_given(die(N), Ai),
        holds_given(die(K), Aj), N < K.
10.Ai << Aj <-
   holds_given(pr(feel_guilty, Pi),Ai),
   holds_given(pr(feel_guilty, Pj),Aj), Pi < Pj.
```

Figure 2: Sophie's choice example

Line 1 says that there is always expectation for every abducible declared. In addition, if Sophie has no special reason for any child, there is no expectation to the contrary of those abducibles (lines 5-6). Thus, the abductive solutions for the only active goal *decide* are:

$A_1 = [letting\_both\_die, not\ kill(child\_1),$
$\qquad not\ kill(child\_2), not\ flip\_a\_coin]$
$A_2 = [kill(child\_1), not\ kill(child\_2), not\ flip\_a\_coin]$
$A_3 = [kill(child\_2), not\ kill(child\_1), not\ flip\_a\_coin]$
$A_4 = [flip\_a\_coin]$

i.e. it is possible for Sophie to decide to let both of her children die, choose one on her own decision, or flip a coin to decide. Then, in the next stage, *a posteriori* preferences are taken into account to filter out the less preferred abductive solutions. Considering the *a posteriori* preference encoded in line 9, solution that includes letting both die is ruled out since it leads to the consequence of two children dying, which is less preferred than any of the (equally) preferred remaining solutions (all with the consequence of just one child dying) (line 7). From the three remaining solutions, the ones that kill some child are ruled out since their consequences are the greater probability of Sophie to feel guilty than the one of flip a coin (line 8), having taken into account the *a posteriori* preference in line 10.

In short, Sophie's final decision is to flip a coin since, according to this, only one child will die and she will feel less guilty about her decision.

Next consider the case when some special reason for a single child, e.g. *child 1*, is entered. Then the expectation to the contrary of *killing_child_1* (line 5) and of *flipping_a_coin* (line 6) are held. Therefore, only two abductive solutions, one including *letting_both_die* and one including $kill(child\_2)$ are available for Sophie to choose. Then, as above, by applying the *a posteriori* preference in line 9, the first one is ruled out. In other words, Sophie's final decision is to kill the child 2.

For a consideration of utility combined with probability within a similar future decision scenaria stance, see [11].

Time-Sensitive Preferences As an agent is evolving, its preferences may change depending on the time point in the evolution that the agent is being at, or even the whole history of the evolution or part of it has to be taken into account.

Example 4 *Suppose agent John has lunch everyday. He can either eat fast food or fruit. His favorite lunch is with fast food since he wants to save his time for work. However, he does not want to be fat, by keeping on having fast food in a number of days, e.g. three as in our example. Hence, if he is fat, he would prefer to have fruit although he has to waste a lot of time on cooking. This scenario can be coded with the program in Figure 3.*

```
1. abds([fast_food/0, fruit/0]).
2. expect(fast_food).   expect(fruit).
3. on_observe(lunch).
   lunch <- fast_food, not fruit.
   lunch <- fruit, not fast_food.
4. save_time <- fast_food.
   cooking <- fruit.
   waste_time <- cooking.
5. Ai << Aj <- \+ fat_prolog,
                 holds_given(save_time,Ai),
                 holds_given(waste_time,Aj).
   Ai << Aj  <-  fat_prolog,
                 holds_given(fruit,Ai),
                 holds_given(fast_food,Aj).
  beginProlog.
   :- import member/2 from basics.
6. fat_prolog :- times(fast_food,3).
7. times(X,N) :- current_state(S),
                 M is S - N + 1, M > 0,
                 have_from_to(X, M, S).
   have_from_to(X,M,S) :- M > S, !.
   have_from_to(X,M,S) :- timeStamp(As, M),
                 member(X,As), M1 is M+1,
                 have_from_to(X,M1,S).
  endProlog.
```

Figure 3: Having lunch example

Line 1 is the declaration of program abducibles. There is expectation to every abducible declared and no expectation to their contrary (line 2). At each cycle, there are two initial abductive solutions, i.e. the ones obtained before *a posteriori* preferences being performed, for the only active goal *lunch*: $[fast\_food, not\ fruit]$, $[fruit, not\ fast\_food]$. Next the *a posteriori* preferences in line 5 are taken into account to rule out the less favorable solutions. These preferences are time-sensitive and hold depending on a part of the evolution, namely, three days before

the day being considered. Notice that in the body of any *a posteriori* preference we use only Prolog code (*holds_given*/2 is a reserved Prolog predicate of our system). The nullary predicate fat_prolog is a Prolog predicate which is to say that John is fat after having kept on eating fast foot for three days (lines 6-7). Thus, if suppose that the original state is day 1, then in the first three days John will have fast food since only the first *a posteriori* preference in line 5 holds and rules out the abductive solution including $fruit$ that leads to the consequence of wasting time which is less preferred than the remaining one which leads to the consequence of saving time (line 4). On the fourth day, John realizes that he is fat, i.e. the predicate fat_prolog holds. This falsifies the first *a posteriori* preference in line 5 and triggers the second one which rules out the abductive solution including $fast\_food$. Thus, John has fruit on this day. Then, similarly, on the next three days John will have fast food again until he feels fat. In short, for this simple example, John has fruit on the day divided by 4 (4,8,12,...) and has fast foot for the other days.

Inevitable Actions There may be abducibles that belong to every initial abductive solution (before considering *a posteriori* preferences). These abducibles are called *inevitable* and will be committed to whatever the final abductive solution is. Realizing that actually committing to some abducible changes the knowledge base, and may trigger preferences that subsequently might help to rule out some irrelevant abductive solutions (or even to provide the final decision for the current active goals), our agent is equipped with the ability to detect the inevitable abducibles, committing to them. Doing the inevitable first can lead to further inevitables.

Example 5 *Suppose agent John wants to take some money. He can go to one of three banks: a, b or c. All the banks are at the same distance from his place. In addition, John needs to find a book for his project work. The only choice for him is to go to the library. At first, John cannot decide which bank to go to. After a moment, he realizes that in any case he must go to the library, so does it first. Arrived there he notices that bank c is now the nearest compared to the others. So he then decides to go to c. This scenario can be coded with the program in Figure 4.*

```
1. abds([lib/0, a/0, b/0, c/0]).
2. expect(lib). expect(a).
   expect(b). expect(c).
3. on_observe(take_money).
   take_money <- a, not b, not c.
   take_money <- b, not a, not c.
   take_money <- c, not b, not a.
4. on_observe(find_book).
   find_book <- lib.
5. Ai << Aj <- dif_distance, hold(dist(Di),Ai),
              hold(dist(Dj), Aj), Di < Dj.
6. dist(10) <- prolog(current_position(lib)), a.
   dist(5)  <- prolog(current_position(lib)), b.
   dist(0)  <- prolog(current_position(lib)), c.
   beginProlog.
7. dif_distance :- current_position(lib).
8. go_to(lib) :- commit_to(lib).
   current_position(C) :- go_to(C).
   endProlog.
```

Figure 4: Inevitable action example

There are two active goals *take_money* and $find\_book$, and hence, three strict abductive solutions (i.e. consider only positive abducibles) that satisfy them: $[a, lib]$, $[b, lib]$, $[c, lib]$. Since the abducible *lib* belongs to all abductive solutions, it is an inevitable one. Thus, the actual commitment to *lib*, i.e. the action of going to the library, is performed. This changes John's current position (line 8). John's new position is at different distances from the banks (line 7) which triggers the *a posteriori* preference in line 5. This preference rules out the abductive solutions including *a* and *b* since they lead to the consequences of having further distances in comparison with the one including *c* (line 6). In short, from this example we can see that actually committing to some inevitable action may help to reach a decision for a problem that could not determinedly and readily be solved without doing that first, for there were three equal options competing.

3.2 Multiple-step prospective agent

While looking ahead a number of steps into the future, the agent is confronted with the problem of having several different possible courses of evolution. It needs to be able to prefer amongst them to determine the best courses from its present state (and any state in general). The (local) preferences, such as the *a priori* and *a posteriori* ones presented above, are no longer appropriate enough, since they can be used to evaluate only one-step-far consequences of a commitment. The agent should be able to also declaratively specify preference amongst evolutions through their available historical information as well as by quantitatively or qualitatively evaluating the consequences or side-effects of each evolution choice. Respectively, we equip our agent with two kinds of evolution-level preferences: *evolution result a posteriori preference* and *evolution history preference*.

3.2.1 Evolution result a posteriori preference

A posteriori preference is generalized to prefer between two evolutions. An *evolution result a posteriori preference* is performed by evaluating consequences of following some evolutions. The agent must use the imagination (look-ahead capability) and present knowledge to evaluate the consequences of evolving according to a particular course of evolution. An *evolution result a posteriori preference* rule has the form:

$$E_i \lll E_j \leftarrow holds\_in\_evol(L_i, E_i), \\ holds\_in\_evol(L_j, E_j)$$

where E_i, E_j are evolutions and L_i, L_j are domain literals. This preference implies that E_i is preferred to E_j if L_i and L_j are true as side-effects of evolving according to E_i or E_j, respectively, without further abduction. Optionally, in the body of the preference rule there can be recourse to any Prolog predicate, used to quantitatively compare the consequences of the two evolutions for decision making.

As a generalized version of *a posteriori* preference, what an *evolution a posteriori* preference actually does is the same as considering its induced *a posteriori* preference at the last step of the prospection, i.e. we can, instead of using *evolution a posteriori* preferences, use the their local induced versions (change $holds\_in\_evol/2$ by $holds\_given/2$), conditioning that they are considered only at the last step of the future prospection.

Example 6 *During war time agent David, a good general, needs to decide to save one city, a or b, from an attack. He does not have enough military resources to save both. If a city is saved, citizens of the city are saved. Normally, a bad general, who just sees the situations at hand would prefer to save the city with more population, but a good general would look ahead a number of steps into the future to choose the best strategy for the war as a larger whole. Having already scheduled for the next day that it will be a good opportunity to make a counter-attack on one of the two cities of the enemy, either a small or a big city, the prior action of first saving a city should take this foreseen future into account. In addition, it is always expected a successful attack on a small city, but the (harder) successful attack on the big city would lead to a much better probability of making further wins in the war. It is expected to successfully attack the big city only if the person who knows the secret information about the enemy (John) is alive in the city to be saved beforehand. The described scenario is coded with the program in Figure 5.*

Line 1 is the declaration of abducibles. Save a city is an ongoing commitment since it has direct influence on the next state, but is defeasible. The context sensitive integrity constraint in line 4 implies that at most one city can be saved since the lack of resources is a foreseen event. Thus, there are two abductive solutions: $[save(a), not\ save(b)]$ and $[save(b), not\ save(a)]$.

If the general is a bad one, i.e. is a single-step prospective agent, the *a posteriori* preference in line 7 would be immediately taken into account and rule out the abductive solution including $save(a)$, since it leads to the saving of 1000 people, which is less preferred than the one including $save(b)$ which leads to the saving of 2000 people (lines 5-6). Then, on the next day, he can attack the small city, but leads to the consequence that the further wining of the whole conflict is very small.

```
1. abds([save/1, big_city/0, small_city/0]).
   on_going_commitment([save(_)]).
2. expect(save(_)).
3. on_observe(save_place) <- be_attacked.
   save_place <- save(a).
   save_place <- save(b).
4. on_observe(not save_atmost_one_ic)
               <- lack_of_resources.
   save_atmost_one_ic <- save(a), save(b).
5. save_men(P) <- save(City),
                  population(City, P).
   alive(X) <- person(X),
           live_in(X, City), save(City).
6. population(a, 1000).  population(b, 2000).
   person(john). live_in(john, a).
   knows(john, secret_inf).
7. Ai << Aj <- holds_given(save_men(Ni), Ai),
        holds_given(save_men(Nj),Aj), Ni > Nj.
8. on_observe(attack) <- good_opportunity.
   attack <- big_city.   attack <- small_city.
9. expect(small_city).
   expect(big_city) <- alive(Person),
                       knows(Person, secret_inf).
10.pr(win, 0.9) <- big_city.
   pr(win, 0.01) <- small_city.
11.Ei <<< Ej <- holds_in_evol(pr(win,Pi), Ei),
        holds_in_evol(pr(win,Pj), Ej), Pi > Pj.
   beginProlog.
12.:- assert(sched_events(1,[lack_resources])),
   assert(sched_events(2,[good_opportunity])).
   endProlog.
```

Figure 5: Saving a city example

Fortunately David is a good general, capable of prospectively looking ahead, at least two steps in the future. David sees three possible evolutions:
$E_1 = [[save(a), not\ save(b)],\ [big\_city, save(a)]]$
$E_2 = [[save(a), not\ save(b)],\ [small\_city, save(a)]]$
$E_3 = [[save(b), not\ save(a)],\ [small\_city, save(b)]$
In the next stage, the *evolution result a posteriori preference* in line 11 is taken into account, ruling out E_2 and E_3 since both lead to the consequence of a smaller probability to win the whole conflict when compared to E_1.

In short, the agent with better capability of looking ahead will provide a more rational decision for the long term goals.

Notice that the reserved prolog predicate *sched_events*/2 can be employed to declare the foreseen scheduled events, e.g. of lacking military resources on the first day and of having a good opportunity to make a counter-attack on the second day, as in line 12.

3.2.2 Evolution history preference

This kind of preference takes into account information from the history of evolutions. The information can be quantitative, such as having in the evolution a maximal or minimal number of some type of commitment, or having the number of commitments greater, equal or smaller than some threshold. It also can be qualitative, such as time order of commitments along an evolution. Such preferences can be used *a priori* upon the process of finding possible evolutions. However, if all preferences (of every kind) coded in the program have been applied but there is still more than one possible evolution, an interaction mode with the user is turned on to ask for user's additional preferences. Similarly, if no solution can satisfy the preferences, the user may be queried about which might be relaxed, or which relaxation option to consider. Now the *evolution history preferences* are used *a posteriori*, given by the user in a list, so as to choose the most cherished evolutions. An evolution history preference can exhibit one of these forms, where C is an abducible:

1. *max(C)/min(C)/greater(C,N)*: find the evolutions having number of commitments to C *maximal/minimal/greater* than N.

2. *smaller(C,N)/times(C,N)*: find the evolutions having number of commitments to C *smaller than/equal* to N.

3. *prec(C1,C2)/next(C1,C2)*: find the evolutions with commitment C1 *preceding/next to* C2 in time.

Example 7 *Agent John must finish a project. He has to schedule his everyday actions so that he can*

finish it on time. Everyday he either works or relaxes. He relaxes by going to the beach, to a movie or watching football. Being a football fan, whenever there is a football match on TV, John relaxes by watching it. The described scenario is coded in Figure 6.

```
1. abds([beach/0, movie/0, work/0, football/0]).
2. expect(beach).  expect(movie).  expect(work).
3. on_observe(everyday_act).
   everyday_act <- work.
   everyday_act <- relax.
   relax <- beach.  relax <- movie.
   relax <- football.
4. expect(football)  <- prolog(have_football).
   expect_not(beach) <- prolog(have_football).
   expect_not(work)  <- prolog(have_football).
   expect_not(movie) <- prolog(have_football).
5. on_observe(on_time).
   on_time <- deadline(Deadline),
           project_work(Days),
           prolog(working_days(Deadline, Days)).
   deadline(5).   project_work(2).
6. beginProlog.
   :- import member/2 from basics.
   have_football :- current_state(S),
                    member(S, [1,2]).
   working_days(Deadline, Days) :-
         assert(plan_pref(times(work, Days))),
         assert(plan_ending(Deadline)).
   endProlog.
```

Figure 6: Football example

In line 5 we can see how an *evolution history preference* is used *a priori* in the predicate $working\_days/2$. There are two reserved predicates $plan\_pref/1$ and $plan\_ending/1$ that allow for asserting *a priori* evolution history preferences and the necessary number of look ahead steps. At the beginning, the agent tentatively runs the active goals to collect all *a priori evolution preferences* and decide how many steps are needed to look ahead. In this case, the agent will look ahead five steps taking into account the *a priori evolution history preference times(work,2)*. There are six possible evolutions:
$E_1 = [[beach], [football], [football], [work], [work]]$,
$E_2 = [[movie], [football], [football], [work], [work]]$,
$E_3 = [[work], [football], [football], [beach], [work]]$,
$E_4 = [[work], [football], [football], [movie], [work]]$,
$E_5 = [[work], [football], [football], [work], [beach]]$,
$E_6 = [[work], [football], [football], [work], [movie]]$
Since there are several possible evolutions, the interaction mode is turned on for John to give a list of evolution history preferences. Suppose, he prefers the evolutions with maximal number of goings to the beach, entering the list $[max(beach)]$. Three possible evolutions E_1, E_3 and E_5 remain. John is asked again for preferences. Suppose he likes going to the beach after watching football, thereby entering $[next(football, beach)]$. Then the only possible evolution is now E_3.

4 Semantics

We provide semantics for evolving prospective logic programming on top of the semantics for Abdual over Well-Founded semantics (WFS) [6]. Since we do not use explicit negation, in section 2 only normal logic programs are considered. It may be skipped on first reading, without lost of continuity. To begin with, some basic definitions are recalled.

4.1 Preliminaries

4.1.1 Terminology

An *objective literal* is either an atom A, or the *explicit negation* of A, denoted $-A$. If an objective literal O is an atom A, the explicit conjugate of O ($conj_E(O)$) is the atom $-A$; otherwise if O has the form $-A$, the explicit conjugate of O is A.

A *literal* either has the form O, where O is an objective literal, or $not(O)$ the *default negation* of O. Default conjugates are defined similarly to explicit conjugates: the default conjugate ($conj_D(O)$) of an objective literal O is $not(O)$, and the default conjugate of $not(O)$ is O.

A *program* P (sometimes also called an extended program), formed over some countable language of function and predicate symbols $\mathcal{L}_P$, is a countable set of rules of the form $H \leftarrow Body$ in which H is an objective literal, and $Body$ is a possibly empty finite sequence of literals.

The closure of the set of literals occurring in P under explicit and default conjugation is termed $literals(P)$.

By a *three-valued interpretation* $\mathcal{I}$ of a ground program P we mean a subset of $literals(P)$. We denote as $\mathcal{I}_T$ the set of objective literals in $\mathcal{I}$, and as $\mathcal{I}_F$ the set of literals of the form $not(O)$ in $\mathcal{I}$. For a ground objective literal, O, if neither O nor $not(O)$ is in $\mathcal{I}$, the truth value of O is undefined.

The *information ordering* of interpretations is defined as follows. Given two interpretations, $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{I} \subseteq_{Info} \mathcal{J}$ if $\mathcal{I}_F$ is a subset of $\mathcal{J}_F$, and $\mathcal{I}_T$ is a subset of $\mathcal{J}_T$.

4.1.2 The WFS of Extended Programs

The well-founded model can be seen as a double iterated fixed point whose inner operators determine a set of true and false literals at each step. More explanations can be found in [6].

Definition 1 *For a ground program P, interpretation $\mathcal{I}$ of P and sets $\mathcal{O}_1$ and $\mathcal{O}_2$ of ground objective literals, define $Tx^P_{\mathcal{I}}(O_1) = \{O : \text{there is a clause } O \leftarrow L_1, ..., L_n \in P \text{ and for each } i, 1 \leq i \leq n, L_i \in \mathcal{I} \text{ or } L_i \in \mathcal{O}_1\}$ and $Fx^P_{\mathcal{I}}(\mathcal{O}_2) = \{O : conj_E(O) \in I \text{ or (for all clauses } O \leftarrow L_1, ..., L_m \in P \text{ there exists } i, 1 \leq i \leq m, conj_D(L_i) \in \mathcal{I} \text{ or } L_i \in \mathcal{O}_2)\}$*

Definition 2 *Let P be a ground program, then ω^P_{ext} is an operator that assigns to every interpretation $\mathcal{I}^1$ of P a new interpretation $\mathcal{I}^2$ such that*

$\mathcal{I}^2_T = lfp(Tx^P_{\mathcal{I}^1}(\emptyset))$
$\mathcal{I}^2_F = \{not(O) | O \in gfp(Fx^P_{\mathcal{I}^1}(objective\_literals(P)))\}$

Definition 3 (WFS for Extended Programs) *Let P be a ground extended program. WFS(P) is defined as the least fixed point, over the information ordering, of ω^P_{ext}.*

4.1.3 Three-Valued Abductive Frameworks

Definition 4 (Integrity Constraint - IC) *An* IC *for a ground program P has the form*

$$\bot \leftarrow L_1, \ldots, L_n$$

where each L_i, $1 \leq i \leq n$ is a literal formed over an element of $\mathcal{L}_P$.

Definition 5 (Abductive framework) *An abductive framework is a triple $< P, \mathcal{A}, I >$ where $\mathcal{A}$ is a finite set of ground objective literals of $\mathcal{L}_P$ called* abducibles*, such that for any objective literal O, $O \in \mathcal{A}$ iff $conj_E(O) \in A$, I is a set of ground integrity constraints, and P is a ground program such that (1) there is no rule in P whose head is in $\mathcal{A}$; and (2) $\bot/0$ is a predicate symbol not occurring in $\mathcal{L}_P$.*

Definition 6 (Abductive scenario) *A scenario of an abductive framework $\langle P, \mathcal{A}, I \rangle$ is a tuple $\langle P, \mathcal{A}, \mathcal{B}, I \rangle$, where $\mathcal{B}$, a set of literals formed over $\mathcal{A}$, is such that there is no $O \in \mathcal{B}$ such that $conj_E(O) \in \mathcal{B}$. $P_{\mathcal{B}}$ is defined as the smallest set of rules that contains for each $a \in \mathcal{A}$, the rule $a \leftarrow t$ (t denotes true) iff $A \in \mathcal{B}$; and $a \leftarrow u$ (u denotes undefined) otherwise.*

Definition 7 (Abductive solution) *An abductive solution of abductive framework $\sigma = \langle P, \mathcal{A}, I \rangle$ is a scenario $\sigma = \langle P, \mathcal{A}, \mathcal{B}, I \rangle$ such that $\bot$ is false in $M(\sigma) = WFS(P \cup P_{\mathcal{B}} \cup I)$.*

Definition 8 (Abductive solution for a query) *We say that $\sigma = \langle P, \mathcal{A}, \mathcal{B}, I \rangle$ is an abductive solution for a query Q if $M(\sigma) \models Q$. σ is minimal if there is no other abductive solution $\langle P, \mathcal{A}, \mathcal{B}', I \rangle$ for Q such that $WFS(\mathcal{B}') \subseteq_{info} WFS(\mathcal{B})$.*

4.2 Semantics for Evolving Prospective Logic Program

Definition 9 (PL program) *A prospective logic (PL) program is a tuple $\pi = \langle P, \mathcal{A}, I, PrA, PosA \rangle$ where $\langle P, \mathcal{A}, I \rangle$ is an abductive framework; PrA and $PosA$ are sets of a priori and a posteriori preferences, respectively.*

At each evolution step or cycle the agent has a set of active goals to satisfy. Those active goals are captured by those $on\_observe/1$ literals which belong to the Well-Founded model of the extended program induced by the PL program (without preferences and abduction).

Definition 10 (Active goals) *The set of all active goals of π is*

$$AGs = \{G \mid on\_observe(G) \in WFS(P \cup I)\}$$

We call the conjunctional goal of π

$$AG(\pi) = \bigwedge_{G \in AGs} G$$

Definition 11 (A priori abductive solution) *A priori abductive solution of PL program $\pi = \langle P, \mathcal{A}, I, PrA, PosA\rangle$ is an abductive solution of the abductive framework $\langle P'', \mathcal{A}, I'\rangle$ for the query $AG(\pi)$, where P'' and I' are constructed thus*

1. *Let P', I' be the program and set of integrity constrains obtained from P and I by replacing every abducible $a \in \mathcal{A}$ in all of their rules with $consider(a)$, respectively.*
2. *P'' is obtained by adding to P*
   - *for each abducible $a \in \mathcal{A}$ the rule*

     $$consider(a) \leftarrow expect(a), \; not\ expect\_not(a), a$$

   - *for each a priori preference $a \lhd b \leftarrow L1, ..., L_n$ in $PriA$ the rule*

     $$expect\_not(b) \leftarrow L_1, ..., L_n, \; not\ expect\_not(a), expect(a), not\ a$$

$\alpha(\pi)$ denotes the set of all a priori abductive solutions of π.

Definition 12 (Side-effect) *Literal L is a side-effect of (a priori) abductive solution A, i.e. (reserved) predicate $holds\_given(L, A)$ holds, in PL program $\pi = \langle P, \mathcal{A}, I, PrA, PosA\rangle$, iff there exists an abductive solution $\langle P'', \mathcal{A}, B, I'\rangle$ for query L such that $B \subseteq A$ (P'' and I' per Definition 11).*

Definition 13 *The a posteriori preference*

$$A_i \ll A_j \leftarrow holds\_given(L_i, A_i),\ holds\_given(L_j, A_j)$$

is said to be applicable to a given pair of abductive solutions (A_1,A_2) iff L_i, L_j are, respectively, side-effects of abductive solutions A_1, A_2, i.e. both $holds\_given(L_i, A_1)$ and $holds\_given(L_j, A_2)$ hold.

Having obtained the set of all or some a priori abductive solutions, the a posteriori preferences are taken into account to rule out the less relevant ones. The remaining abductive solutions are called a posteriori.

It is up to the user to specify a priori and a posteriori preferences that satisfy the application domain needs, and guarantee any order properties desired. A posteriori preferences are defined by user algorithms that prefer amongst abductive solutions for whatever reason, possibly including order of generation as not all solutions need to be generated before a preference choice is made. Also, preference may be postponed to a later prospection cycle, when more information is made available.

Each prospection cycle is completed by registering any retained a posteriori abductive solutions into the knowledge base and moving to the next cycle of evolution. The information about individually committed abducibles at each cycle is kept by setting time stamps. Since committing to an abductive solution does not change any a priori and a posteriori preferences of the PL program, we only need to consider the changing w.r.t. its abductive framework.

Definition 14 (Evolution trace) *Given PL program $\pi = \langle P, \mathcal{A}, I, PrA, PosA\rangle$, an evolution trace of length N originating from π is the sequence $(\pi_0, ..., \pi_N)$, where $\pi_0 = \pi$, $\pi_i = \langle P_i, \mathcal{A}_i, I_i, PrA, PosA\rangle (1 \leq i \leq N)$ as follows*

1. *let $T_0 = timestamp(0, \emptyset)$; $H := \emptyset$.*
2. *For $1 \leq i \leq N$, let $(\_, \_, AS, \_)$ be an arbitrary a posteriori abductive solution of π_{i-1} (if there is none, the evolution trace does not exist), where $AS = \{a_1, ..., a_n\} \cup \{not\ b_1, ..., not\ b_m\}$ with $\{a_1, ..., a_n\} \cup \{b_1, ..., b_m\} \subseteq \mathcal{A}$*
   - *Set $T_i = timestamp(i, AS \cup H)$.*
   - *For each k, $1 \leq k \leq n$, if a_k is a hard commitment then add the rule $a_k \leftarrow t$ to P_i and remove a_k from $\mathcal{A}_i$. Otherwise, if a_k is an ongoing one, store the current P_i and I_i, then delete all occurrences of a_k in the body of all rules in P_i and I_i. In both cases add a_k to H.*

- *For each l, $1 \leq l \leq m$, if b_l is a hard commitment then remove b_l from A_i and add the integrity constraint $\bot \leftarrow b_l$ to I_i. Otherwise, if b_l is an ongoing one, check whether it is committed to "recently", i.e. whether $b_l \in H$. If yes, remove b_l from H and restore b_l to the rules and integrity constraints it was removed from, by comparing with the stored ones.*

If in each cycle there is a set of events, say EV_i in cycle i, then we need to add the events EV_i to the component P_i of the PL program π_i, for $1 \leq i \leq N$. An event can be a fact or a rule. Thus, adding here means add that rule or fact to the mentioned program.

Definition 15 (Evolution tree) *Given PL program $\pi = \langle P, \mathcal{A}, I, PrA, PosA \rangle$, the sequence $E = \{AS_1, ..., AS_N\}$ where $T_i = timestamp(i, AS_i)(i = 1, ..., N)$(as in definition 14), is called an evolution of length N of π. The corresponding evolution trace $(\pi_0, ..., \pi_N)$ is dubbed evolution trace following E.*

Considering all possible evolution traces of length N originating from π, we obtain a set of evolutions of length N, dubbed the evolution tree of length N of π.

Definition 16 (EPL program) *Evolving prospective logic (EPL) program generalizes PL one by providing two kinds of evolution-level preferences. Formally, an EPL program is a tuple $\delta = \langle P, \mathcal{A}, I, PrA, EPosA, EHisA \rangle$ where $P, \mathcal{A}, I, PrA$ are as before; $EPosA$ is a set of evolution a posteriori preferences; $EHisA$ is a set of evolution history preferences.*

Definition 17 (Evolution side-effect) *Literal L is a side-effect of evolution $E = \{AS_1, ..., AS_N\}$, i.e. (reserved) predicate $holds\_in\_evol(L, E)$ holds, in EPL program $\delta = \langle P, \mathcal{A}, I, PrA, EPosA, EHisA \rangle$, iff L is a side-effect of abductive solution AS_N in PL program π_{N-1}, where $(\pi_0, ..., \pi_{N-1})$ is the evolution trace following evolution $E' = \{AS_1, ..., AS_{N-1}\}$, originating from PL program $\pi_0 = \langle P, \mathcal{A}, I, PrA, \emptyset \rangle$.*

Definition 18 *The evolution a posteriori preference*

$$E_i \lll E_j \leftarrow holds\_in\_evol(L_i, E_i), \\ holds\_in\_evol(L_j, E_j)$$

is said to be applicable to a given pair of evolutions (E_1, E_2) iff L_i, L_j, respectively, are side-effects of evolutions E_1, E_2, i.e. both $holds\_in\_evol(L_i, E_1)$ and $holds\_in\_evol(L_j, E_2)$ hold.

When looking ahead further into the future, say $N > 1$ steps, to satisfy the long-term goals, the local (or one cycle step) a posteriori preferences are not taken into account. The so-called favorite evolutions that achieve the long-term active goals and survive after applying all preferences are obtained as follows. First, the evolution tree is generated, without considering the a posteriori preferences, then its evolutions will be preferred amongst each other using evolution a posteriori preferences (the remaining evolutions are called a posteriori). If, afterwards, there is still more than one evolution, the evolution history preferences will be taken into account.

Definition 19 (A posteriori evolution) *Given EPL program $\delta = \langle P, \mathcal{A}, I, PrA, EPosA, EHisA \rangle$, suppose Q is the evolution tree of length N of the PL program $\pi = \langle P, \mathcal{A}, I, PrA, \emptyset \rangle$. The set of a posteriori evolutions of δ, denoted by $\tau(\delta)$, is, similarly to the definition of a posteriori abductive solutions, determined, for instance, by the following procedure (assume that the preferences in $EPosA$ have been sorted in decreasing order):*

Procedure 1 .
if $(|Q| \leq 1)$
  $\tau(\delta) := Q$.
else
  $Tau := Q$.
  $T = true$.
  for $p \in EPosA$
    while $((T = true)$ *and* $(|Tau| > 1))$
      for (E_1, E_2) *in* $Tau \times Tau$
        if p *is applicable to* (E_1, E_2)
          $Tau := Tau - \{E_2\}$.
          restart while-loop.
        end if
      end for
      $T = false$.
    end while
  if $(|Tau| = 1)$
    break for-loop.

end if
end for
$\tau(\delta) := Tau$.
end if-else

Definition 20 (Favorite evolution) *Given EPL program* $\delta = \langle P, \mathcal{A}, I, PrA, EPosA, EHisA \rangle$. *The set of favorite (final) evolutions of* δ *is obtained from the set of a posteriori evolutions* $\tau(\delta)$ *by considering in succession the preferences in* $EHisA$ *and keeping only evolutions that satisfy the preference.*

5 Implementation

5.1 Abdual

Our system is implemented on top of Abdual [6, 14], an implemented XSB-Prolog system that allows computing abductive solutions for given queries.

Briefly, Abdual is composed of two modules: the preprocessor which transforms the original program by adding its dual rules, plus specific abduction-enabling rules; and a meta-interpreter allowing for top-down abductive query solving. When solving a query, abducibles are dealt with by means of extra rules the preprocessor added to that effect. These rules just add the name of the abducible to an ongoing list of current abductions, unless the negation of the abducible was added before to the list, in that case failing in order to ensure abduction consistency.

5.2 Evolving program with Abdual

To enable an Abdual program to evolve, by being able to update with new rules or facts, first of all, the preprocessor needs to be changed to make the whole code dynamic, including the original program and additional dual and abduction-enabling rules. Adding a rule or fact to the knowledge base is then performed by adding and removing a set of rules so as to be compatible with the transformation of the preprocessor.

As a prospective agent is evolving, the information about committed abducibles at each cycle is kept by setting time stamps so that later on it can be used, e.g. by evolution-level preferences, to model evolution history related predicates or even to let the agent hypothetically return to the past. The time stamped commitments are asserted to the system without being transformed by the preprocessor. Recall that different kinds of commitment are treated differently based on their influence on the future: hard and ongoing commitments keep on affecting the future onwards from the committed to them state though ongoing ones can be defeated; and temporary ones has no direct influence. Thus, the hard and ongoing ones, besides being time stamped, are added as facts (after being preprocessed) to the knowledge base while temporary commitments are just time stamped. In addition, in order to make a hard commitment non-defeasible, the abducible that being a hard commitment, is removed henceforth from the list of declared abducibles.

So far we have described several different kinds of preference mechanisms. The *a priori* one is implemented by using the transformation provided in the section 2.2.

The *a posteriori* one is mainly employed by means of the reserved predicate $holds\_given(L, A)$, which is to check whether the domain literal L is a true side-effect of the abductive solution A, i.e. whether there is an abductive solution for query L that is included in A. This means if A was committed to, L would be true in the result knowledge base.

Similarly, as being a generalized version of *a posteriori preferences*, *evolution result a posteriori ones* is employed by means of the predicate $holds\_in\_evol/2$, which is implemented by tentatively following the evolution given in the second argument until the one-to-last cycle, then check if the literal given in the first argument is a true side-effect of the abductive solution in the last cycle.

6 Conclusions and Future Work

We have shown how to model evolving prospective logic program agent systems, including single-step and multiple-step ones. Besides declaratively specify-

ing local preferences such as *a priori* and *a posteriori* ones, in order to let a prospective agent look ahead a number steps into the future and prefer amongst their hypothetical evolutions, we provide new kinds of preference, at evolution level, that can evaluate long-term consequences of a choice as well as analyze different kinds of information about the evolution history, which is kept by annotating such information with time stamps for each evolution cycle. In addition, active goals triggered by external events, context-sensitive integrity constraints and context-sensitive preferences provide flexible ways for modelling the changing knowledge base of an evolving prospective agent. We exhibited several examples to illustrate all proffered concepts. By means of them, we have, to some degree, managed to show that multiple-step prospective agents are more intelligent than the single-step ones, in the sense that they are able to give more reasonable decisions for long-term goals. In addition, the decision making process at each cycle during an evolution of our agent was, in many cases, enhanced by committing to so-called inevitable abducibles.

There are currently several possible future directions to explore. First of all, in each cycle the agent has to satisfy a set of active goals and sometimes he cannot satisfy them all. There are goals more important than others and it is vital to satisfy them while keeping the others optional. The agent can be made more focussed by setting a scale of priorities for the active goals so it can focus on the most important ones. We can make a scale by using preferences over the $on\_observe/2$ predicates that are used for modelling active goals.

Similarly, since there are integrity constraints that must be satisfied and there are also ones that are less important, we can prefer amongst integrity constraints by making them all context-sensitive and then prefer amongst the $on\_observe/2$ predicates used for modelling them.

When looking ahead, the prospective agent has to search the evolution tree for the branches that satisfy his goals and preferences. From this perspective we can improve our system with heuristic search algorithms such as best-first search, i.e. the most promising nodes will be explored first.We also can improve the performance of the system by using multi-threading which is very efficient in XSB, from version 3.0 [15]. Independent threads can evolve on their own and they can communicate with each other to decide whether some thread should be canceled or kept evolving, based on the search algorithm used.

In multiple-step prospective agents, the a posteriori preferences are not taken into account since they, not being aware of what will happen next, possibly cut off the potentially good courses of evolutions even they seem to be bad at the time being considered. Thus, the generated evolution tree can be extensively large. One possible solution for this problem is that we can set some priorities for a posteriori preferences, and some really important ones can be considered.

In multi-agent setting, it is undoubtedly that the decision making process of one agent would be much more efficient if he is aware of the intentions of others. One of our current work is to equip evolving prospective agents with intention recognition mechanism, i.e. based on the observations of of a sequence of actions abduce the goal that lead to the selection of those actions.

In addition, reasoning under uncertainty can be enabled for evolving prospective agents by integrating probabilistic reasoning into the system. Since our system is implemented on top of XSB Prolog, we can easily make use of P-log [12], a declarative language based on logic formalism for probabilistic reasoning, an extended version of which was implemented in XSB Prolog [13].

On a more general note, it appears the practical use and implementation of abduction in knowledge representation and reasoning, by means of declarative languages and systems, has reached a point of maturity, and of opportunity for development, worthy the calling of attention of a wider community of potential practitioners.

7 Acknowledgements

We thank Alexandre Pinto for his useful discussions upon the launching of this project, José Julio Alferes for his help with Abdual. We also thank Alexandre Pinto and Pierangelo Dell'Acqua for their useful

comments on the previous version of this paper.

References

[1] L. M. Pereira, G. Lopes. *Prospective Logic Agents.* Progress in Artificial Intelligence, Procs. 13th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'07), pp. 73−86, Springer LNAI 4784, Guimarães, Portugal, December 2007.

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

[3] J. J. Alferes, F. Banti, A. Brogi and J. A. Leite. *The Refined Extension Principle for Semantics of Dynamic Logic Programming* , Studia Logica 79(1): 7-32, 2005.

[4] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. *Dynamic updates of non-monotonic knowledge bases.* J. Logic Programming, 45(1-3):4370, September/October 2000.

[5] 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 U. P.

[6] J. J. Alferes, L. M. Pereira, T. Swift. *Abduction in Well-Founded Semantics and Generalized Stable Models via Tabled Dual Programs*, Theory and Practice of Logic Programming. 4(4), 383-428, 2004.

[7] R. Kowalski.*The logical way to be artificially intelligent.* In F. Toni, P. Torroni (eds.), Procs. of CLIMA VI, LNAI Springer, 2006.

[8] L. M. Pereira, P. Dell'Acqua, G. Lopes, *On Preferring and Inspecting Abductive Models*, Invited paper in: A.Gill, T. Swift (eds.), Procs. 11th Intl. Symp. Practical Aspects of Declarative Languages (PADL'09), pp. 1-15, Springer LNCS 5418, Savannah, Georgia, USA, January 2009.

[9] P. Dell'Acqua, L. M. Pereira. *Preferential theory revision.* Journal of Applied Logic, 5(4):586-601, Elsevier, 2007.

[10] L. M. Pereira, H. T. Anh, *Evolution Prospection*, in: K. Nakamatsu (ed.), Procs. First KES Intl. Symposium on Intelligent Decision Technologies (KES-IDT'09), pp. 51–64, Springer Studies in Computational Intelligence vol.199, Himeji, Japan, April 2009.

[11] L. M. Pereira, C. K. Ramli, *Modelling Probabilistic Causation in Decision Making*, in: K. Nakamatsu (ed.), Procs. First KES Intl. Symposium on Intelligent Decision Technologies (KES-IDT'09), Springer Verlag book in Engineering Series, Himeji, Japan, April 2009.

[12] C. Baral, M. Gelfond, and N. Rushton. *Probabilistic reasoning with answer sets.* In Proceedings of LPNMR7, pages 21−33, 2004.

[13] H. T. Anh, C. K. Ramli, C. V. Damásio. *An implementation of extended P-log using XASP.* In Proceedings of 24th International Conference on Logic Programming, Udine, Italy, 2008.

[14] Neg-Abdual. Available at *http://centria.di.fct.unl.pt/∼lmp/software.html*, (2008).

[15] *XSB-PROLOG system freely available at: http://xsb.sourceforge.net.*

[16] W. Styron, *Sophie's Choice*, Bantam Books, New York, 1980.