Evolution Prospection

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.

1 Introduction

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,

Luís Moniz Pereira
Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa,
2829-516 Caparica, Portugal
e-mail: lmp@di.fct.unl.pt

Han The Anh
Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa,
2829-516 Caparica, Portugal
e-mail: h.anh@fct.unl.pt

K. Nakamatsu et al. (Eds.): New Advan. in Intel. Decision Techno., SCI 199, pp. 51–63.
springerlink.com

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 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.

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. 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 [2, 4] 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 [3] - a XSB-Prolog implemented system which allows computing abductive solutions for a given query.

2.1 Language

Let $\mathscr{L}$ be a first order language. A domain literal in $\mathscr{L}$ 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 $\mathscr{L}$ 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 $\mathscr{L}$ is a rule with an empty head. A (logic) program P over $\mathscr{L}$ 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 $\mathscr{A} \subseteq \mathscr{L}$. 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 [6].

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

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 [5].

To express preference criteria among abducibles, we envisage an extended language $\mathscr{L}^\star$. A preference atom in $\mathscr{L}^\star$ is of the form $a \lhd b$, where a and b are

abducibles. It means that if b is assumed (i.e. abduced), then $a \triangleleft b$ forces a to be assumed too (b can only be abduced if a is as well). A preference rule in $\mathscr{L}^\star$ is of the form: $a \triangleleft b \leftarrow L_1, ..., L_t \ (t \geq 0)$, where $L_1, ..., L_t$ are domain literals over $\mathscr{L}^\star$. *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.

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:

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

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 1. 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.

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

```
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).
```

Fig. 1 Ticket example

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.

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 2. 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 2.

```
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.
```

Fig. 2 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's choices.. 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:

$$Ei \lll Ej \leftarrow holds\_in\_evol(L_i, Ei), holds\_in\_evol(L_j, Ej)$$

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. 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.

Example 3. 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

```
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.
```

Fig. 3 Saving a city example

information about the enemy (John) is alive in the city to be saved beforehand. The described scenario is coded with the program in Figure 3.

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.

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.

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 4. 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 4.

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.

```
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.
```

Fig. 4 Football example

4 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 specifying 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 a new kind 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 and context-sensitive integrity constraints 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 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 [7]. 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.

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.

References

1. Pereira, L.M., Lopes, G.: Prospective Logic Agents. In: Neves, J., Santos, M.F., Machado, J.M. (eds.) EPIA 2007. LNCS (LNAI), vol. 4874, pp. 73–86. Springer, Heidelberg (2007)
2. Kakas, A., Kowalski, R., Toni, F.: The role of abduction in logic programming. Handbook of Logic in Artificial Intelligence and Logic Programming 5, 235–324 (1998)
3. Alferes, J.J., Pereira, L.M., Swift, T.: Abduction in Well-Founded Semantics and Generalized Stable Models via Tabled Dual Programs. Theory and Practice of Logic Programming 4(4), 383–428 (2004)
4. Kowalski, R.: The logical way to be artificially intelligent. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 1–22. Springer, Heidelberg (2006)
5. Pereira, L.M., Lopes, G., Dell'Acqua, P.: On Preferring and Inspecting Abductive Models. In: Gill, A., Swift, T. (eds.) PADL 2009. LNCS. Springer, Heidelberg (2009)
6. Dell'Acqua, P., Pereira, L.M.: Preferential theory revision. Journal of Applied Logic 5(4), 586–601 (2007)
7. XSB-PROLOG system freely, `http://xsb.sourceforge.net`