An ASP-based Approach to Dealing with Agent Perception Failure *

Francesco Buccafurri, Gianluca Caminiti and Domenico Rosaci **,
DIMET Dept., Facoltà di Ingegneria, Università "Mediterranea" di Reggio Calabria, via Graziella, loc. Feo di Vito, 89060 Reggio Calabria, ITALY
E-mail: {bucca, gianluca.caminiti, domenico.rosaci}@unirc.it

The paper presents a new modality extending Answer Set Programming in order to refine its capability of representing knowledge in case of possible perception failure. The aim of the proposal is mainly to present a new logical approach to deal with uncertainty arising from perception bugs, and to show its effectiveness in the context of agent reasoning representation.

The basic environment of Answer Set Programming, has been chosen to simplify the study of the above aspect, without limiting the possibility of using our approach in the wide scenario of agent-oriented logic languages.

Keywords: knowledge representation, answer set programming, non-monotonic reasoning

1. Introduction

Answer Set Programming (ASP) [7,2] is a language very suitable for representing human-like reasoning in case of possible incomplete information, and it is used as a core for many agent-oriented logic languages [3,11,10,6]. Moreover, logic programming is a framework very suitable also to represent dynamic environments provided that specific features concerning time, actions, inertia, etc., are included in the language [15,8,14,9].

Our paper falls in the context of the agent reasoning representation by logic programming. In this scenario, we are interested in a particular aspect not yet considered in logic-based languages for agents. We refer to the chain sense-think-act of agents, where beyond sensors a perception layer always may alter what actually sensors report. Perception is intended in a general way: it can be related to the view of a robot, or to the network layer, in case of remote source of knowledge for a software agent, and so on. The core problem of this paper is the following: How to represent agent reasoning that takes into account possible perception failure?

We reasonably assume that the above aspect is orthogonal to those addressed by agent-oriented logic languages (mentioned above). As a consequence, we study this problem in the simplest setting, that is pure Answer Set Programming.

It is well known that ASP negation by default combined with classical negation, allow us the representation of reasoning in presence of incomplete knowledge. However, our purpose is to face the case of missing information arising from the failure of the perception layer, by assuming that some status information detects such abnormal situation. In this case, we can realize that ASP does not permit a natural encoding, so that we propose a specific modality making the encoding natural and strongly declarative.

Consider the following example. An agent is approaching a traffic light (for simplicity suppose that the traffic light consists of just a green light that can be either *on* or *off*). He knows that in case he is able to perceive the status of the light, he can draw reliable information from the traffic light. Thus, if the light is *on*, then the agent crosses the way, otherwise he stops. Suppose now that the

*NOTICE: this is the authors version of a work that was accepted for publication in AI Communications. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in AI Communications, [VOL 21, ISSUE 1, (January 2008)]

**Corresponding author: Francesco Buccafurri, E-mail: bucca@unirc.it

ISSN 0921-7126, IOS Press.

agent is able to detect a situation of compromised perception, for example the presence of a thick fog. In this case, he knows that it might happen that light perceived as off does not actually mean that the agent has to stop (since the fog prevents the agent to see the light). Suppose again that the agent would keep an optimistic behaviour, in order not to loose time (for instance in case of emergency). In other words, he wants to cross the way also in the latter case.

If we try to represent the above reasoning by ASP, then we write the following rule:

$r_1 : cross \leftarrow not\ \neg on$

In words, the above rule means that the agent crosses the way if he has not evidence that the light is *off* (here, synonym of $\neg on$). The literal $\neg on$ could be derived through at least one rule where it appears in the head. Since we want to take into account possible incomplete knowledge arising from the weather conditions, we do not allows the derivation of $\neg on$ simply on the basis of the impossibility of deriving the truth of *on*. In other words, the program cannot contain a "closure rule" of the form:

$r_2 : \neg on \leftarrow not\ on$

The ASP encoding above works correctly only in case of fog. Indeed, whenever the view is clear, if the agent does not see the light *on*, he goes across anyway (since the literal $\neg on$ is not derived and, therefore, the rule r_1 allows the derivation of *cross*). Clearly, according to the above description, this does not correctly represent the behaviour of the agent, who in case of full perception gives to the traffic light strong reliability (thus, in case of light *off* he stops).

The aim of the paper is to introduce a machinery capable of dealing with the perception failure, allowing us to refine the representation of the reasoning in case of incomplete information in order to take into account some detected status of perception failure.

We implement such machinery by introducing the possibility of *underlining* literals.

In the example above, we apply this modality to the literal *on*, and represent the agent reasoning by the following rule:

$r_3 : cross \leftarrow \underline{on}$

The meaning is the following: In case of full perception the underlining is ignored, otherwise the meaning of the body is just that of the original ASP program (i.e., $not\ \neg on$). As a consequence, the agent, in case of fog, goes across also if he does not see the light on. But whenever the view is clear, since the underlying is ignored, he stops if he does not see the light on (exactly as we have required).

Observe that the application of this machinery to the literal *on* in the rule r_3, makes the condition required to derive the head *cross* less strict than the standard case (i.e., $cross \leftarrow on$). In words, the agent requires the condition *on*, in order to go across, but if he is not able to perceive such an information, he does without it.

Thus, our modality, applied to a positive literal occurring in the body of a rule, refines its meaning, by assuming a more optimistic behaviour (than the standard one) in case of perception-based indefiniteness of such a literal. Coherently, the same modality, if applied to a literal with negation as failure, gives a specular behaviour.

Indeed we expect that rule:

$r_4 : stop \leftarrow not\ \underline{on}$

has the same behaviour of the rule r_3, as it happens in ASP. Indeed, if the underlining in rules r_3 and r_4 is not considered, then we have that the meaning of the two rules coincides, as both heads and bodies represent a complementary knowledge. In particular, both rules state than the agent goes across only in case *on* is true.

The semantics of underlining clearly preserves this behaviour, so that according to the rule r_4 (with underlining), in case of perception failure (we recall that in the other case underlining is uninfluential), the body is true only if *on* is proven to be false (i.e., $\neg on$ is true). Differently from the standard case, the indefiniteness of *on* does not make true its negation by failure. In other words, the proposed modality allows us to refine negation by failure by introducing a more pessimistic behaviour than the standard case. Indeed, in case of perception failure we do not assume the falsity of the information carried out by the literal to which the negation is applied, as standard ASP does.

In the next section we present two more articulated examples in order to give the flavour of our proposal in an intuitive way. The formal description of syntax and semantics of our programs, called *U(nderlined)-programs*, is presented in Section 3. In Section 4 we give the translation of our semantics into answer sets and we prove that such a translation is correct. The relevance of the contribution of our proposal is discussed, by examples, in Section 5, while, in Section 6, we discuss about related work. Finally we draw our conclusions in Section 7.

2. Examples

2.1. The Robot

In this example, we consider a robot moving into a room having a chessboard floor, in order to achieve a target object. The robot can move either horizontally, vertically or diagonally, and it can rotate for aligning itself with a row, a column or a diagonal. There could be one obstacle in some square. The room boundaries are also viewed as obstacles. This situation is depicted in Figure 2-(A), where the symbols F, L and R represent the front, left and right side of the robot respectively. The robot relies on the following boolean sensors:

(1) *target_ahead*: The sensor is *true* whenever the target is entirely within the field of vision (supposed to be 90°) of a frontal video-cam;
(2) *target_front*: The sensor is *true* whenever the angle between the robot front direction and that pointing the target is less than 45°, also in case the obstacle is in the middle (we might imagine this sensor as a directional antenna);
(3) *target_achieved*: When this sensor is *true*, there is at most an empty square between the robot and the target;
(4) *obstacle_front*: The value *true* means that there is at most an empty square between the robot and the obstacle in front of it;
(5) *obstacle_left* (resp. *obstacle_right*): The value *true* means that the obstacle is in the next square on the left (resp. on the right).

Concerning the above sensors, we allow that some of them may be affected by (perception) failure (in particular, sensors (1) and (4)).

We consider some other events, each associated to an action that is performed by the robot whenever the event is inferred by the program. These are:

(6) *go_ahead*: The robot moves one position ahead;
(7) *redirect*: The robot rotates in order to maximize the alignment with the target, i.e. the robot chooses an allowed direction (i.e. horizontal, vertical, diagonal) such that is minimum the angle between that direction and the target one;
(8) *avoid_left* (resp. *avoid_right*): The robot moves sideways one position on the left (resp. on the right), in order to avoid the obstacle in front of it, provided that *go_ahead* is not derived (thus, the robot prefers to go ahead, if possible);
(9) *end*: The robot stops. This action has the highest priority.

The following ASP program $\mathcal{P}$, whose literals represent the above sensors and events, can be associated to the robot:

$$
\begin{aligned}
r_1 :\ & \neg obstacle\_front \leftarrow target\_ahead, \\
& \qquad\qquad\qquad\quad target\_front \\
r_2 :\ & \qquad redirect \leftarrow not\ target\_front \\
r_3 :\ & \qquad go\_ahead \leftarrow not\ obstacle\_front, \\
& \qquad\qquad\qquad\quad target\_front \\
r_4 :\ & \qquad avoid\_left \leftarrow target\_front, \\
& \qquad\qquad\qquad\quad obstacle\_front, \\
& \qquad\qquad\qquad\quad not\ obstacle\_left, \\
& \qquad\qquad\qquad\quad not\ avoid\_right \\
r_5 :\ & \qquad avoid\_right \leftarrow target\_front, \\
& \qquad\qquad\qquad\quad obstacle\_front, \\
& \qquad\qquad\qquad\quad not\ obstacle\_right, \\
& \qquad\qquad\qquad\quad not\ avoid\_left \\
r_6 :\ & \qquad\qquad end \leftarrow target\_achieved
\end{aligned}
$$

The meaning of the program rules follows:

r_1 : In case the robot both is aligned with and perceives the target, then it assumes that there is no obstacle in the frontal direction, thus allowing, by means of the following rule r_3, a sort of "contact navigation".
r_2 : The robot performs the action *redirect* in order to be aligned with the target.
r_3 : The action *go_ahead* is taken if the robot is aligned with the target and there is no evidence of obstacles.
r_4 : (resp. rule r_5): The robot performs the action *avoid_left* (resp. *avoid_right*) if (i) it is aligned with the target, (ii) an empty square separates the robot from the obstacle in front of it, (iii) there is no evidence of an obstacle in the next square on the left (resp. on the right) and, finally, (iv) the action *avoid_right* (resp. *avoid_left*) has not been chosen.
r_5 : If the target is far at most one empty square from the robot, then the action *end* is taken.

Moreover, the robot can assume three states, that we call *evaluate*, *run* and *stop*, on the basis of the following behaviour (see Figure 1).

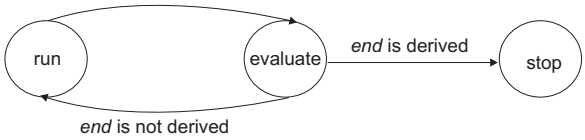

Fig. 1. The robot behaviour

In the state *evaluate* the robot applies its reasoning task, by first inserting in the program $\mathcal{P}$ a fact (i.e., a rule with empty body) for each true sensor and then evaluating the intended models (i.e. the answer sets) of the resulting program. If *end* is not derived, then the robot switches to the state *run*, where it nondeterministically chooses one of the answer sets computed in the previous state and then executes the action associated to the chosen answer set (according to the program, due to priorities stated in **(8)** and **(9)**, the action to execute is always unique). Next, it switches back to *evaluate*. Otherwise, in case *end* is derived in the state *evaluate*, the robot switches to the state *stop*, where it halts.

Since we admit that some sensor may be imperceptible, we want to study the behaviour of the robot in each of the following cases, either:

(a) all the sensors are perceptible,
(b) *target_ahead* is imperceptible,
(c) *obstacle_front* is imperceptible, or
(d) both *target_ahead* and *obstacle_front* are imperceptible.

Case (a). First, assume that the robot starts from the position 1 of Figure 2-(A) and all the sensors are perceptible (observe that in this case the presence of underlined literals in the program would be irrelevant).

The initial state of the robot is *evaluate*, and all the sensors are *false*. $\mathcal{P}$ admits only one answer set, namely *redirect* (throughout this example, for simplicity, we do not include sensor literals in the answer sets).

Thus, the robot switches to the state *run* where executes the action *redirect*, that is it rotates for aligning itself with the target. This way, the robot reaches the position 2 (see Figure 2-(B)), and then it switches to the state *evaluate*.

Now all the sensors but *target_front* are *false*. *target_front* is *true* since, in such a position, the target is within the aperture of the antenna. Again, $\mathcal{P}$ has only one answer set, which is *go_ahead*. Thus, the robot goes to the state *run*.

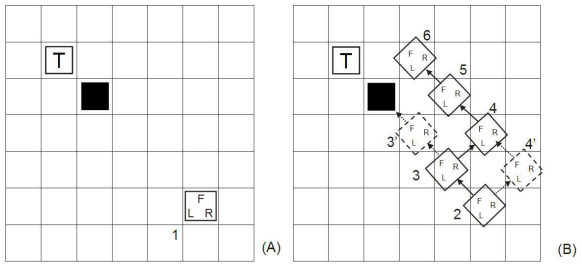

Fig. 2. An example of robot movement

Due to the execution of the action *go_ahead*, the robot moves one position ahead, reaching the position 3, and then it switches to the state *evaluate*. In this state the values of the sensors are the previous ones, with only the change of *obstacle_front*, which becomes *true*, since the distance between the robot and the obstacle is one square.

At this point the robot evaluates the answer sets of $\mathcal{P}$, that are $\{avoid\_left\}$ and $\{avoid\_right\}$, and switches to the state *run*. Then it nondeterministically chooses the answer set $\{avoid\_right\}$ and thus it executes the action *avoid_right*, moving sideways one square on the right, reaching the position 4, and switching then to the state *evaluate*.

Now $(*)$ the values of the sensors are *true* for both *target_ahead* and *target_front* and *false* for the others. Observe that, since the target belongs to the field of vision of the video-cam and it is aligned with the robot, the falsity of *obstacle_front* is derived. The program admits now one answer set, namely $\{go\_ahead\}$. Thus, the robot goes to the state *run*, where the corresponding action is executed.

Whenever the robot comes back to the state *evaluate*, the configuration $(*)$ above is replicated. As a consequence, the execution pattern following $(*)$ will be repeated one more time, until the robot reaches the position 6.

In this position, both *target_ahead* and *target_front* become *false*, since the target is not detected anymore by both the video-cam and the antenna, thus only *target_achieved* is *true*. In the state *evaluate* the robot computes the answer set of $\mathcal{P}$, that is $\{end,\ redirect\}$, and switches to the state *stop*.

Now, consider the imperceptibility cases (b), (c) and (d) mentioned above.

Case (b). Suppose the robot re-starts from the position 1 of Figure 2-(A) and the sensor *target_ahead* is imperceptible due to, say, fog. Ta-

Table 1
Robot behaviour (programs $\mathcal{P}$ and $\mathcal{P}''$) in case (b)

Pos	Evaluate	Run
1	{}	redirect
2	{target_front}	go_ahead
3	{target_front, obstacle_front}	avoid_right (avoid_left)
4	{target_front}	go_ahead
5	{target_front}	go_ahead
6	{target_achieved}	end

Table 2
Robot behaviour (program $\mathcal{P}$) in both case (c) and case (d)

Pos	Evaluate	Run
1	{}	redirect
2	{target_front}	go_ahead
3	{target_front}	go_ahead
3′	{target_front}	go_ahead
	crash!	

ble 1 describes the robot behaviour reporting, for each position achieved, the set of *true* sensors in the state *evaluate* and the action performed in the state *run*.

Likewise case (a), the robot is able to reach position 4 (see Figure 2-(B)). Now, observe that the intended meaning of rule r_3 is the following: the action *go_ahead* is performed either in case the robot proves the falsity of *obstacle_front* (by means of rule r_1) or in case there is no evidence of obstacles in front of it, provided that *target_front* is true. Since in this case rule r_1 is blocked, then the robot relies just on the fully working sensors *obstacle_front* and *target_front*, in order to proceed safely, thus performing a sort of "instrument navigation".

Thus, thanks to the repeated effect of rule r_3, the robot easily achieves the target.

Unfortunately, in case (c) the robot does not succeed in achieving the target, as shown by Table 2 and next explained.

Case (c). Now assume that the sensor *obstacle_front* is imperceptible. The robot starts from the position 1 of Figure 2-(A) and reaches position 3 of Figure 2-(B). The above assumption corresponds to having all the sensors but *target_front* false.

In the state *evaluate*, the robot computes the only answer set of $\mathcal{P}$, that is $\{go\_ahead\}$ $(**)$ (due to the rule r_3), thus after switching to the state *run*, it moves to a square next to the obstacle. Now, the robot switches to the state *evaluate*, replicating the situation $(**)$ and finally it goes ahead, crashing into the obstacle.

Table 3
Robot behaviour (program $\mathcal{P}'$) in both case (a) and case (c) (Robot behaviour (program $\mathcal{P}''$) in case (c))

Pos	Evaluate	Run
1	{}	redirect
2	{target_front}	avoid_right (avoid_left)
4′	{target_ahead, target_front}	go_ahead
4	{target_ahead, target_front}	go_ahead
5	{target_ahead, target_front}	go_ahead
6	{target_achieved}	end

Case (d). It is easy to see that Table 2 is applicable also to case (d).

In order to explain this anomaly, we observe that the program $\mathcal{P}$ does not take into account the information incompleteness arising from the sensors.

However, ASP is capable of dealing with such an issue, by means of the two forms of negation, suitably combined each other.

Indeed, in order to overcome the above problem, we try to change $\mathcal{P}$ in such a way that (i) the robot does not go ahead unless it has proved that there is no frontal obstacle and (ii) side movements are allowed also in case the frontal obstacle is not perceived. Thus, we rewrite rules r_3, r_4 and r_5 as follows:

$$
\begin{aligned}
r_3' : \quad & go\_ahead \leftarrow \neg obstacle\_front, \\
& \qquad\qquad target\_front \\
r_4' : \quad & avoid\_left \leftarrow target\_front, \\
& \qquad\qquad not \ \neg obstacle\_front, \\
& \qquad\qquad not\ obstacle\_left, \\
& \qquad\qquad not\ avoid\_right \\
r_5' : & avoid\_right \leftarrow target\_front, \\
& \qquad\qquad not \ \neg obstacle\_front, \\
& \qquad\qquad not\ obstacle\_right, \\
& \qquad\qquad not\ avoid\_left
\end{aligned}
$$

Observe that, in rule r_3', the NAF of *obstacle_front* has been replaced by $\neg$*obstacle_front*, in order to prevent the robot to go ahead unless the explicit falsity of *obstacle_front* (i.e. $\neg$*obstacle_front*) has been proved, provided that *target_front* is true.

Moreover, in rules r_4' and r_5', every no-NAF occurrence of the literal *obstacle_front* has been replaced as the NAF of the complementary literal. This results in blocking the body of rules r_4' and r_5' if $\neg$*obstacle_front* has been proved.

Now we show that such a solution is not satisfactory, since it does not work in both case (b) and case (d). Anyway, first we study case (a) and case (c).

Table 4
Robot behaviour (program $\mathcal{P}'$) in both case (b) and case (d) (Robot behaviour (program $\mathcal{P}''$) in case (d))

Pos	Evaluate	Run
1	{}	redirect
2	{target_front}	avoid_right (avoid_left)
4'	{target_front}	avoid_right (avoid_left)
endlessly moves to the right or to the left!		

Cases (a) and (c). Table 3 shows the behaviour of the robot, represented by program $\mathcal{P}' = \mathcal{P} \setminus \{r_3, r_4, r_5\} \cup \{r'_3, r'_4, r'_5\}$ in case (a). Note that in case (c) the robot has the same behaviour. According to the new program $\mathcal{P}'$, the robot succeeds in achieving the target.

Let us describe now cases producing anomalies.

Cases (b) and (d). In case (b) (resp. in case (d)) the robot behaviour is described by Table 4. Observe that, according to the program $\mathcal{P}'$, the robot can go ahead only if $\neg$*obstacle_front* has been proved (due to rule r_1), provided that the sensor *target_front* is true. Since in case (b) (resp. in case (d)) the rule r_1 is blocked, then the only move the robot is allowed to do is either *avoid_right* or *avoid_left*.

Now, although this is the intended robot behaviour in case (d) (since if both the sensors *target_ahead* and *obstacle_front* are imperceptible, we expect that the robot is not able to achieve the target), it holds that in case (b) the robot fails to reach the target.

At this point, we apply the underlined modality in order to fix the above problem.

To this aim we need to modify the rules r_3, r_4, and r_5 of $\mathcal{P}$ as follows:

$$\begin{aligned}
r''_3 :&\quad go_ahead \leftarrow not\ \underline{obstacle_front},\\
&\qquad\qquad\quad target_front\\
r''_4 :&\quad avoid_left \leftarrow target_front, \underline{obstacle_front},\\
&\qquad\qquad\quad not\ obstacle_left,\\
&\qquad\qquad\quad not\ avoid_right\\
r''_5 :&\ avoid_right \leftarrow target_front, \underline{obstacle_front},\\
&\qquad\qquad\quad not\ obstacle_right,\\
&\qquad\qquad\quad not\ avoid_left
\end{aligned}$$

The underlining applied to the rule r''_3 has an intuitive counterpart: If the robot is not able to perceive the presence of a frontal obstacle then it should not go ahead, unless $\neg$*obstacle_front* is derived, meaning that the sensor *target_ahead* is functional.

Moreover, the explanation of the underlining appearing in the rules r''_4 and r''_5 is that side movements of the robot are safe, thanks to the presence of the (perceptible) sensors *obstacle_left* and *obstacle_right*, although either the frontal view is compromised due to the imperceptibleness of *target_ahead* (case (b)) or *obstacle_front* is imperceptible (case (c)).

Now we show that, according to the new program $\mathcal{P}'' = \mathcal{P} \setminus \{r_3, r_4, r_5\} \cup \{r''_3, r''_4, r''_5\}$, our robot will behave as intended in all the considered cases.

Cases (a) and (b). In both case (a) and case (b) $\mathcal{P}''$ behaves as $\mathcal{P}$ (see Table 1), since all underlinings are ignored.

Cases (c) and (d). In both case (c) (see Table 3) and case (d) (see Table 4) $\mathcal{P}''$ behaves as $\mathcal{P}'$, since underlining in rule r''_3 (resp. in rules r''_4 and r''_5) strengthen (resp. soften) the requirements on *obstacle_front* expressed in the rule body.

Thus, the robot represented by $\mathcal{P}''$ achieves the target in all cases (a), (b) and (c). Finally, In case (d) the robot endlessly moves either to the left or to the right, without going ahead.

2.2. The Exploration Boat

An exploration boat has just performed some experiments in open sea and it is travelling back to the coast, after the sunset. A *lighthouse* has been built near the harbour, in order to show the presence of some danger. Moreover, an on-board *gps* device is capable of reporting that the boat is near to the coast.

Under normal weather conditions, the captain proceeds until the two following events occur: (1) he is alerted by the gps device (i.e. *gps* is true) and (2) he sees the lighthouse glowing (*lighthouse* is true).

This situation can be represented by an ASP program:

$$\begin{aligned}
r_1 :&\ stop_engine \leftarrow gps, lighthouse\\
r_2 :&\quad \neg gps \leftarrow out_of_order\\
r_3 :&\ \neg lighthouse \leftarrow radio_informed
\end{aligned}$$

Where rule r_1 describes the captain's behaviour and by means of rule r_2 (resp. rule r_3) he assumes the falsity of the information associated to the sensor *gps* (resp. *lighthouse*) in case it is out of order (resp. in case he has been informed by radio that it is turned off).

Now, suppose that either (i) the gps device is unreachable by satellite signals, or (ii) a thick layer of fog descends on the sea surface, thus cloaking the lighthouse.

In either case (i) or (ii) the captain wants to stop the engine, in order not to wreck the boat.

Unfortunately, the above ASP program does not correctly represent such a cautious behaviour in either the uncertain scenario (i) or (ii), since if either *gps* or *lighthouse* is not true, then *stop_engine* is not derived.

Moreover, if we rewrite the rule r_1 as

$$r_1' : stop\_engine \leftarrow not\ \neg gps, not\ \neg lighthouse$$

then we obtain that the new program behaves as intended only in case (i) or (ii). Otherwise (i.e. in case all sensors are perceptible) it fails since the boat does not proceed even whether it could (i.e. whenever both the gps device works and no radio information about the lighthouse is given).

The desired behaviour is obtained by encoding the rule r_1 by means of underlining in the following way:

$$r_1'' : stop\_engine \leftarrow \underline{gps}, \underline{lighthouse}$$

Indeed, if all sensors are perceptible, then the underlining can be ignored and the meaning of the program is the standard one: The engine is stopped only if the lighthouse is visible and the coast is nearby, whereas the boat proceeds otherwise. In case of sensor imperceptibility (i.e. either case (i), case (ii), or both), then the captain stops the engine only if at least one literal between $\neg gps$ and $\neg lighthouse$ is true.

3. Syntax and Semantics

In this section we recall some basic concepts regarding ASP programs [7] and we introduce the *underlined* logic programs (U-programs) and their semantics. For the sake of presentation we only refer, in this section, to variable-free (also said *ground*) programs – the extension to the general case is straightforward.

Recall that an *atom* is an expression $p(t_1, \cdots, t_n)$, where p is a *predicate* of arity n and $t_1, \cdots, t_n$ are constants.

A *literal* is either a *positive literal* a or a *negative literal* $\neg a$, where a is an atom and $\neg$ is the *classical negation* symbol. Given a literal a, its *complementary* literal $\neg a$ is defined as $\neg p$, if $a = p$ and p if $a = \neg p$. Given a set L of literals, we denote by $\neg L$ the set $\{\neg l \mid l \in L\}$. A set L of literals is said to be *consistent* if $L \cap \neg L = \emptyset$. Given a literal a we say that *not* a is the *negation as failure* (NAF) of a (observe that NAF of negative literals is allowed).

We recall that an (ASP) *rule* r is a formula $a \leftarrow b_1, \cdots, b_i, not\ b_{i+1}, \cdots, not\ b_m$ $(0 \leq i \leq m)$, where a and each b_i $(1 \leq i \leq m)$ are literals.

An *(ASP) program* is a finite set of rules.

Now, we introduce the notions of *underlined* literal, *U-rule* and *U-program*.

Definition 1. Given a positive literal a, we say that $\underline{a}$ is an *underlined* literal. Given an underlined literal $\underline{a}$, its complementary literal is the literal $\neg a$, while the NAF of $\underline{a}$ is *not* $\underline{a}$.

Observe that the application of the classical negation removes the underlining.

Definition 2. A *U-rule* r is a rule where each b_i $(1 \leq i \leq m)$ is a (possibly underlined) literal. Given a (U-)rule r, the literal a is called the *head* of r, the conjunction of literals $b_1, \cdots, b_i, not\ b_{i+1}, \cdots, not\ b_m$ is the *body* of r. $b_1, \cdots, b_i$ (resp. $not\ b_{i+1}, \cdots, not\ b_m$) is called the *positive part* (resp. the *NAF part*) of the body of r. We denote the set of (possibly underlined) literals appearing in the head, in the positive part, and in the NAF part of r by $head(r)$, $body^+(r)$, $body^-(r)$, respectively. We denote by $body(r)$ the set $body^+(r) \cup body^-(r)$.

Given a U-rule r, we define the two sets $\underline{body^+}(r) = \{a \mid \underline{a} \in body^+(r)\}$ and $\underline{body^-}(r) = \overline{\{a \mid \underline{a} \in body^-(r)\}}$.

Definition 3. A *U-program* is a finite set of U-rules.

A U-program with no underlined literal is referred to as an *independent* U-program. Observe that an independent U-program can be viewed as an ASP program.

Now we introduce the semantics of U-programs.

Definition 4. Given a (U-)program $\mathcal{P}$, we denote by $Lit^{\mathcal{P}}$ the set of literals occurring in $\mathcal{P}$.

Definition 5. Given a U-program $\mathcal{P}$, we define the set $s\text{-}Lit^{\mathcal{P}} \subseteq Lit^{\mathcal{P}}$ of *s-literals of* $\mathcal{P}$.

Literals in $s\text{-}Lit^{\mathcal{P}}$ represent boolean sensors which the program $\mathcal{P}$ depends on. As a consequence, we consider only U-programs where s-literals may occur in the head of the rules, provided that they are negative.

Remark. Thus we assume that in general a sensor a sends either the information $true$ or $false$, but the latter case does not coincide with explicitly stating the falsity of the event associated to a (which would be represented by the literal $\neg a$).

In other words, we allow possible incomplete information about sensors (in the sense that either they might give false negatives or they could not be correctly perceived), so that a sensor a sending the information $false$ just produces the success of any negation by failure on it, whereas the information $\neg a$ needs to be explicitly derived.

Definition 6. Given a U-program $\mathcal{P}$, a *U-interpretation* for $\mathcal{P}$ is a pair $\langle I, D\rangle$ where I is a consistent subset of $Lit^{\mathcal{P}}$, $D \subseteq s\text{-}Lit^{\mathcal{P}}$ and $I \cap D = \emptyset$.

The set D represents those s-literals that are associated to imperceptible sensors.

We recall now the definition of *answer set* of an ASP program $\mathcal{P}$, as introduced in [7]. We start by giving the definition of *interpretation* for a (U-) program $\mathcal{P}$ as a consistent subset of $Lit^{\mathcal{P}}$. Note that, according to the above definition, we want to limit our focus only on consistent answer sets[1].

First we need to introduce the *GL-transformation* of $\mathcal{P}$ w.r.t. an interpretation I for $\mathcal{P}$ as the *not*-free program $\mathcal{P}^I$ obtained from $\mathcal{P}$ by deleting (i) each rule $r \in \mathcal{P}$ such that $body^-(r) \cap I \neq \emptyset$; (ii) from each remaining rule r, every formula *not* a such that $a \in body^-(r)$.

An interpretation I for $\mathcal{P}$ is an *answer set* of $\mathcal{P}$ if I is the smallest subset of $Lit^{\mathcal{P}}$ such that for every rule $r \in \mathcal{P}^I$, if $body(r) \subseteq I$ then $head(r) \subseteq I$.

Now we define the intended models of our programs. First, we need to introduce a transformation whose aim is to produce an independent program.

Definition 7. Let $\mathcal{P}$ be a U-program and $\langle I, D\rangle$ a U-interpretation for $\mathcal{P}$. We define the *U-transformation* of $\mathcal{P}$ w.r.t $\langle I, D\rangle$ as the independent program $\mathcal{P}^{\langle I,D\rangle}$, obtained from $\mathcal{P}$ by executing the following operations:

(1) delete every rule $r \in \mathcal{P}$ such that $(\underline{body}^-(r) \cap D) \not\subseteq \neg I$;
(2) remove from the body of every remaining rule r every underlined literal $\underline{a}$ such that $a \in (\underline{body}^+(r) \cap D) \wedge a \notin \neg I$;
(3) remove from the body of each rule every underlining.

The meaning of the above operations is:

(1) Every rule including the NAF of underlined s-literal a whose associated information is imperceptible is removed, provided that, the falsity of the associated information has not been proved;
(2) The dependence on underlined literals is consistently eliminated, i.e., any underlined "imperceptible" s-literal is removed from any remaining rule, provided that, for such s-literals, the falsity of the associated information has not been proved;
(3) Underlining applied to both s-literals associated to perceptible sensors and non s-literals are meaningless.

Observe that the program so obtained is clearly an independent U-program. We are thus ready to introduce the definition of intended models of a U-program.

Definition 8. Given a U-program $\mathcal{P}$, a U-interpretation $\langle I, D\rangle$ for $\mathcal{P}$ is a *U-answer set* of $\mathcal{P}$ if I is an answer set of $\mathcal{P}^{\langle I,D\rangle}$ thought as ASP program.

Consider again the example of the robot described Section 2.1, and, in particular, the last version of the U-program $\mathcal{P}''$, consisting of the rules $\{r_1, r_2, r_3'', r_4'', r_5'', r_6\}$. Suppose that the robot occupies the position 2 in the state *evaluate* (Figure 2-(B)) and the sensor *obstacle_front* is imperceptible (observe that, as required by the definition of the state *evaluate*, a fact $target\_front \leftarrow$ is added to the program $\mathcal{P}''$ at this stage). In such a case we show that the U-interpretation $\langle I, D\rangle = \langle\{target\_front, avoid\_right\}, \{obstacle\_front\}\rangle$, is a U-answer set of $\mathcal{P}''$.

The U-transformation of $\mathcal{P}''$ is the ASP program $\mathcal{P}^{\langle I,D\rangle}$:

$$
\begin{aligned}
target\_front &\leftarrow \\
\neg obstacle\_front &\leftarrow target\_ahead, \\
&\qquad target\_front \\
redirect &\leftarrow not\ target\_front \\
avoid\_left &\leftarrow target\_front, \\
&\qquad not\ obstacle\_left, \\
&\qquad not\ avoid\_right \\
avoid\_right &\leftarrow target\_front, \\
&\qquad not\ obstacle\_right, \\
&\qquad not\ avoid\_left \\
end &\leftarrow target\_achieved
\end{aligned}
$$

[1] According to the original definition of answer sets [7], also the inconsistent set of all possible literals can be a valid answer set.

It's easy to see that I is an answer set of $\mathcal{P}^{\langle I,D\rangle}$.

4. Translation

In this section we give the translation from U-programs under the U-answer set semantics to ASP programs under the answer set semantics.

Moreover, we show that for any U-program $\mathcal{P}$, an ASP program $\Gamma(\mathcal{P})$ exists such that there is a one-to-one correspondence between the U-answer sets of $\mathcal{P}$ and the answer sets of $\Gamma(\mathcal{P})$.

This result allows us to compute the semantics of any U-program $\mathcal{P}$ by exploiting existing answer set solvers such as the DLV system [12], Smodels [16], etc.

Definition 9. Given a U-program $\mathcal{P}$, we define $\Gamma(\mathcal{P})$ as the ASP program obtained from $\mathcal{P}$ by executing Algorithm 1.

For each U-rule
r: $a \leftarrow b_1, \cdots, b_i, \; not \; b_{i+1}, \cdots, not \; b_m,$
$\underline{c_1}, \cdots, \underline{c_j}, \; not \; \underline{c_{j+1}}, \cdots, not \; \underline{c_n}$
for each underlined literal $\underline{c_k}$
(resp. NAF of underlined literal $not \; \underline{c_k}$)
$(1 \leq k \leq n)$:
1. replace $\underline{c_k}$ (resp. $not \; \underline{c_k}$)
by a new literal $f_{r,k} \notin Lit^{\mathcal{P}}$;
2. add the two rules:
$f_{r,k} \leftarrow c_k$
$f_{r,k} \leftarrow imp(c_k), not \; \neg c_k$
(resp. add the two rules:
$f_{r,k} \leftarrow not \; c_k, not \; imp(c_k)$).
$f_{r,k} \leftarrow \neg c_k, imp(c_k))$.

Algorithm 1. The translation Algorithm

Concerning Algorithm 1, observe that the predicate $imp()$ does not occur in $\mathcal{P}$. Moreover, for each s-literal $a \in s\text{-}Lit^{\mathcal{P}}$, the literal $imp(a)$ is used in $\Gamma(\mathcal{P})$ to represent that the sensor associated to a is imperceptible. Finally, it easy to see that the computational complexity of Algorithm 1 is $\mathcal{O}(n)$, where n is the number of underlinings occurring in $\mathcal{P}$.

Before stating the equivalence between the original U-program and its translation, we need to introduce some more definitions.

Definition 10. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$ for $\mathcal{P}$, we define $\mathcal{I}^{\langle I,D\rangle} = I \cup \{imp(a) \mid a \in D\}$.

In words, $\mathcal{I}^{\langle I,D\rangle}$ is obtained by adding to I an atom $imp(a)$ for each "imperceptible" s-literal a.

Moreover, given a U-program $\mathcal{P}$ and a set $D \subseteq s\text{-}Lit^{\mathcal{P}}$, we define both $\Gamma^D(\mathcal{P}) = \Gamma(\mathcal{P}) \cup \{imp(a) \leftarrow \mid a \in D\}$ and $AS^f(\Gamma^D(\mathcal{P}))$ as the set of all answer sets of $\Gamma^D(\mathcal{P})$, where every literal $f_{r,k}$ $(\forall r, k)$ is removed.

Given a U-program $\mathcal{P}$, we want to prove the equivalence between the set of U-answer sets of $\mathcal{P}$ and the set $AS^f(\Gamma^D(\mathcal{P}))$.

To this aim we need some preliminary definitions and results. Recall that, given a positive ASP program $\mathcal{P}$ and an interpretation I for $\mathcal{P}$, I is a (*Herbrand*) *model* of $\mathcal{P}$ if $\forall r \in \mathcal{P}, body(r) \subseteq I \Rightarrow head(r) \subseteq I$.

Given a U-program $\mathcal{P}$, we denote by $\mathcal{F}^{\mathcal{P}}$ the set including every literal $f_{r,k}$ $(\forall r, k)$ occurring in $\Gamma(\mathcal{P})$, by $Ind(\mathcal{P})$ the subset of $\mathcal{P}$ including only rules with no underlined literal, and by $Dep(\mathcal{P})$ the set $\mathcal{P} \setminus Ind(\mathcal{P})$, respectively.

Fact 1. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$ for $\mathcal{P}$, $\forall r \in Dep(\mathcal{P})$, the U-transformation of the U-program $\{r\}$ w.r.t. $\langle I, D\rangle$, $\{r\}^{\langle I,D\rangle}$, is such that either:
(i) $\{r\}^{\langle I,D\rangle} = \emptyset$, if $(\underline{body}^-(r) \cap D) \not\subseteq \neg I$, or
(ii) $\{r\}^{\langle I,D\rangle} = \{s\}$, where s is a classical rule such that
$head(s) = head(r)$,
$body^+(s) = body^+(r) \setminus ((\underline{body}^+(r) \cap D) \setminus (\underline{body}^+(r) \cap D \cap \neg I))$,
$body^-(s) = body^-(r)$.

Fact 2. $\forall r \in Dep(\mathcal{P})$ it is $\Gamma(\{r\}) = R^r \cup \{t^r\}$, such that:

$$
\begin{aligned}
R^r = &\{f_{r,k} \leftarrow c_k \mid c_k \in \underline{body}^+(r)\} \cup \\
&\{f_{r,k} \leftarrow imp(c_k), not \; \neg c_k \mid c_k \in \underline{body}^+(r)\} \cup \\
&\{f_{r,k} \leftarrow not \; c_k, not \; imp(c_k) \mid c_k \in \underline{body}^-(r)\} \cup \\
&\{f_{r,k} \leftarrow \neg c_k, imp(c_k) \mid c_k \in \underline{body}^-(r)\},
\end{aligned}
$$

where $f_{r,k} \notin Lit^{Dep(\mathcal{P})}$ and t^r is a rule such that:

(1) $head(t^r) = head(r)$,
(2) $body^+(t^r) = body^+(r) \setminus \{\underline{a} \mid a \in body^+(r)\} \cup \mathcal{F}^{\{r\}}$,
(3) $body^-(t^r) = body^-(r) \setminus \{\underline{a} \mid a \in body^-(r)\}$.

Before introducing the following result, recall that, given a U-program P and a U-interpretation $\langle I, D\rangle$, $P^{\langle I,D\rangle}$ is an independent program, i.e. it can be seen as an ASP program. As a consequence, $(P^{\langle I,D\rangle})^I$ denotes the GL-tranformation of $P^{\langle I,D\rangle}$ w.r.t. I.

Lemma 1. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$, for each rule $s' \in (Dep(\mathcal{P})^{\langle I,D\rangle})^I$ it holds that I is a model of $\{s'\}$ iff $\bar{I} = \mathcal{I}^{\langle I,D\rangle} \cup \mathcal{F}^{\{r\}}$ is a model of $(\Gamma^D(\{r\}))^{\bar{I}}$, where $r \in Dep(\mathcal{P}) \wedge (\{r\}^{\langle I,D\rangle})^I = \{s'\}$.

Proof. ($\Rightarrow$). In the following, we denote $\{r\}^{\langle I,D\rangle}$ by $\{s\}$.

By virtue of Fact 1, $\forall s' \in (Dep(\mathcal{P})^{\langle I,D\rangle})^I\ \exists r \in Dep(\mathcal{P}) \mid (\{r\}^{\langle I,D\rangle})^I = \{s'\}$. Now, according to the hypothesis, $body^+(s') \subseteq I \Rightarrow head(s') \subseteq I$. We have to prove that $\bar{I}$ is a model of $(\Gamma^D(\{r\}))^{\bar{I}}$.

By virtue of Fact 2, $(\Gamma^D(\{r\}))^{\bar{I}} = (Q \cup R^r \cup \{t^r\})^{\bar{I}}$, where $Q = \{imp(a) \leftarrow \mid a \in D\}$. Then, we must prove that $\bar{I}$ is a model of both (i) $Q^{\bar{I}}$, (ii) $(R^r)^{\bar{I}}$ and (iii) $\{t^r\}^{\bar{I}}$.

To prove items (i) and (ii) is trivial.

Now we prove item (iii): First, we denote $\{t^r\}^{\bar{I}}$ by $\{t'\}$. Observe that, according to the definition of the GL-transformation, $body^+(s') = body^+(s)$, $body^+(t') = body^+(t^r)$ and $head(s') = head(s) = head(t') = head(t^r) = head(r)$.

Moreover, by virtue of both Facts 1 and 2, it holds that $body^+(t^r) \subseteq body^+(s) \cup \mathcal{F}^{\{r\}}$. Thus, if $body^+(s) \subseteq I$, then $body^+(t^r) \subseteq body^+(s) \cup \mathcal{F}^{\{r\}} \subseteq I \cup \mathcal{F}^{\{r\}} \subseteq \bar{I} \Rightarrow head(t^r) = head(s) \subseteq I \subseteq \bar{I}$. Hence, it holds that $body^+(t') \subseteq \bar{I} \Rightarrow head(t') \subseteq \bar{I}$, i.e. $\bar{I}$ is a model of $\{t^r\}^{\bar{I}}$.

($\Leftarrow$). Assume that, by contradiction, $\exists s' \in (Dep(\mathcal{P})^{\langle I,D\rangle})^I \mid \{s'\} = (\{r\}^{\langle I,D\rangle})^I$ and I is not a model of $\{s'\}$. Thus, it is $body^+(s') \subseteq I \Rightarrow head(s') \not\subseteq I$.

By virtue of both Facts 1 and 2, it is easy to see that $body^+(t') \subseteq \bar{I} \Rightarrow body^+(s') \subseteq I$, where $\{t'\}$ denotes $\{t^r\}^{\bar{I}}$. According to the hypothesis and by virtue of Fact 2 it holds that $body^+(t^r) \subseteq \bar{I} \Rightarrow head(t^r) \subseteq \bar{I}$. Thus, since $head(s') = head(t')$, it holds that, according to the definition of $\bar{I}$, $head(t') \subseteq \bar{I} \Rightarrow body^+(s') \subseteq I \Rightarrow head(s') \subseteq I$.

Now, we have reached a contradiction. □

Lemma 2. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$, I is a model of $(Dep(\mathcal{P})^{\langle I,D\rangle})^I$ iff $\exists F \subseteq \mathcal{F}^{\mathcal{P}} \mid \bar{I} = \mathcal{I}^{\langle I,D\rangle} \cup F$ is a model of $(\Gamma^D(Dep(\mathcal{P})))^{\bar{I}}$.

Proof. ($\Rightarrow$). Assume $J = \{r \in Dep(\mathcal{P}) \mid \{r\}^{\langle I,D\rangle} \neq \emptyset\}$ and $Y = \{(\bigcup_{r\in J} \Gamma^D(\{r\}))^{\bar{I}}\}$, by virtue of Lemma 1 there exists $F = \bigcup_{r\in J} \mathcal{F}^{\{r\}}$ such that $F \subseteq \mathcal{F}^{\mathcal{P}} \wedge \bar{I}$ is a model of Y, where $\bar{I} = \mathcal{I}^{\langle I,D\rangle} \cup F$. Now we have to prove that $\forall u \in Z$, where $Z = (\Gamma^D(Dep(\mathcal{P})))^{\bar{I}} \setminus Y$, if $body^+(u) \subseteq \bar{I}$, then $head^+(u) \subseteq \bar{I}$.

First, observe that $\forall u \in Z$ there exists $r \in Dep(\mathcal{P})$ such that $\{r\}^{\langle I,D\rangle} = \emptyset$. Then, by virtue of Fact 1, there exists $f_{r,k} \in body^+(u)$ such that there exists $c_k \in (\underline{body}^-(r) \cap D)$. Moreover, by virtue of Fact 2, since $c_k \in D$, then $f_{r,k} \notin \bar{I}$. As a consequence, $body^+(u) \not\subseteq \bar{I}$.

($\Leftarrow$). Assume the ASP programs $J = \{r \in Dep(\mathcal{P}) \mid \{r\}^{\langle I,D\rangle} \neq \emptyset\}$ and $Y = \{(\bigcup_{r\in J} \Gamma^D(\{r\}))^{\bar{I}}\}$, by virtue of Lemma 1, if $\bar{I}$ is a model of Y, then I is a model of J^I. Now, observe that $Dep(\mathcal{P})^{\langle I,D\rangle} = J$. As a consequence it holds that $Y = (\Gamma^D(Dep(\mathcal{P})))^{\bar{I}}$. Thus, since $\bar{I}$ is a model of $(\Gamma^D(Dep(\mathcal{P})))^{\bar{I}}$, then $\bar{I}$ is a model of $(Dep(\mathcal{P})^{\langle I,D\rangle})^{\bar{I}}$. □

Lemma 3. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$, I is a model of $(\mathcal{P}^{\langle I,D\rangle})^I$ iff $\exists F \subseteq \mathcal{F}^{\mathcal{P}} \mid \bar{I} = \mathcal{I}^{\langle I,D\rangle} \cup F$ is a model of $(\Gamma^D(\mathcal{P}))^{\bar{I}}$.

Proof. Since $\mathcal{P} = Ind(\mathcal{P}) \cup Dep(\mathcal{P})$, it is also, according to the definition of the GL-transformation of $\mathcal{P}$ w.r.t. I, it holds that $\mathcal{P}^I = (Ind(P))^I \cup (Dep(P))^I$.

Now, it is $\Gamma^D(\mathcal{P}) = \Gamma(Ind(P)) \cup \Gamma^D(Dep(P))$. Thus, the GL-transformation of $\Gamma^D(\mathcal{P})$ w.r.t. I is $(\Gamma^D(\mathcal{P}))^I = (\Gamma(Ind(P)))^I \cup (\Gamma^D(Dep(P)))^I$.

Since the following hold: (1) $I \subseteq \bar{I}$ and (2) (according to the definition of $\Gamma()$) $\Gamma(Ind(\mathcal{P})) = Ind(\mathcal{P})$, then it is easy to see that I is a model of $(Ind(\mathcal{P}))^I$ iff $\bar{I}$ is a model of $(\Gamma(Ind(\mathcal{P})))^{\bar{I}}$.

Moreover, by virtue of Lemma 2, I is a model of $(Dep(\mathcal{P}))^I$ iff $\bar{I}$ is a model of $(\Gamma(Dep(\mathcal{P})))^{\bar{I}}$. Thus I is a model of $\mathcal{P}^I = (Ind(\mathcal{P}))^I \cup (Dep(\mathcal{P}))^I$ iff $\bar{I}$ is a model of $(\Gamma^D(\mathcal{P}))^{\bar{I}} = (\Gamma(Ind(\mathcal{P})))^{\bar{I}} \cup (\Gamma(Dep(\mathcal{P})))^{\bar{I}}$. □

Now we can state the equivalence between the set of U-answer sets of a U-program $\mathcal{P}$ and the set $AS^f(\Gamma^D(\mathcal{P}))$.

Theorem 1. Given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$ for $\mathcal{P}$, $\langle I, D\rangle$ is a U-answer set of $\mathcal{P}$ iff $\mathcal{I}^{\langle I,D\rangle} \in AS^f(\Gamma^D(\mathcal{P}))$.

Proof.

First, (i) observe that according to Definition 8, a given U-interpretation $\langle I, D\rangle$ is a U-answer set of $\mathcal{P}$ if $I \in AS(\mathcal{P}^{\langle I,D\rangle})$, and (ii) recall that $AS^f(\Gamma^D(\mathcal{P}))$ is the set including all answer sets of $\Gamma^D(\mathcal{P})$ where all literals $f_{r,k}$ $(\forall r, k)$ have been removed.

Now, it is easy to see that to prove Theorem 1 is equivalent to prove that given a U-program $\mathcal{P}$ and a U-interpretation $\langle I, D\rangle$, $I \in AS(\mathcal{P}^{\langle I,D\rangle})$ iff $\exists F \subseteq \mathcal{F}^{\mathcal{P}} \mid \bar{I} \in AS(\Gamma^D(\mathcal{P}))$, where $\bar{I} = \mathcal{I}^{\langle I,D\rangle} \cup F$.

($\Rightarrow$). Assume, by contradiction, that $\forall F \subseteq \mathcal{F}^{\mathcal{P}}, \bar{I} \notin AS(\Gamma^D(\mathcal{P}))$.

Then, it holds that either

(i) $\bar{I}$ is not a model of $(\Gamma^D(\mathcal{P}))^{\bar{I}}$, or
(ii) $\exists J \subset \bar{I} \mid J$ is a model of $(\Gamma^D(\mathcal{P}))^J$.

In case (i), by virtue of Lemma 3, it holds that I is not a model of $(\mathcal{P}^{\langle I,D\rangle})^I$, and then $I \notin AS(\mathcal{P}^{\langle I,D\rangle})$. Thus we have reached a contradiction.

In case (ii) it must be the case that $F \subseteq J$ because it is $\forall F \subseteq \mathcal{F}^{\mathcal{P}}$, $\bar{I} \notin AS(\Gamma^D(\mathcal{P}))$. Also it must be $\{imp(a) \mid a \in D\} \subseteq J$, because D is fixed.

Then it holds that $J = \mathcal{I}^{\langle I',D\rangle} \cup F$ such that $I' \subset I$ and, by virtue of Lemma 3, it is the case that I' is a model of $(\mathcal{P}^{\langle I',D\rangle})^{I'}$, then it must be that $I \notin AS(\mathcal{P}^{\langle I,D\rangle})$, which contradicts the hypothesis.

($\Leftarrow$). Assume, by contradiction, that the following holds: $I \notin AS(\mathcal{P}^{\langle I,D\rangle})$.

Thus it holds that either

(i) I is not a model of $(\mathcal{P}^{\langle I,D\rangle})^I$, or
(ii) $\exists J \subset I \mid J$ is a model of $(\mathcal{P}^{\langle J,D\rangle})^J$,

Concerning item (i), it is easy to see that $\bar{I} \notin AS(\Gamma^D(\mathcal{P}))$, which contradicts the hypothesis.

Concerning item (ii), by virtue of Lemma 3 $\bar{J} = (\mathcal{I}^{\langle J,D\rangle} \cup F) \subset \bar{I}$ is a model of $(\Gamma^D(\mathcal{P}))^{\bar{J}}$.

Thus $\bar{I} \notin AS(\Gamma^D(\mathcal{P}))$, which contradicts the hypothesis. □

5. Effectiveness of our Approach

The aim of this section is to show that our proposal gives a concrete benefit to the encoding capability of the ASP programmer in real cases. We start by observing that, as the translation above uses status information about imperceptibleness in the form of the predicate $imp(s)$, for a given sensor s, one could think that an encoding that uses directly ASP and assumes the availability of such a predicate leads the programmer to the same conclusion as the usage of U-programs. If so, our proposal would be trivial. Fortunately, this is not the case. Indeed, in general, the direct-ASP encoding using the predicate *imp*, produces programs little declarative. Thus, in other words, the language extension here proposed provides the programmer with an abstraction layer more than plain ASP, oriented to the case of possible imperceptible sensors. We show the above claim by a simple example, but the reader can easily generalize this argumentation by considering all (possibly complex) cases in which a number of underlined literals occur in the bodies of the rules. The example is the U-program $\mathcal{P} = \{r_1''\}$ shown in Section 2.2 (note that rules r_2 and r_3 are not relevant in this context since they are not U-rules):

$$r_1'' : stop\_engine \leftarrow \underline{gps}, \underline{lighthouse}$$

It is easy to see that any direct ASP encoding using the predicate *imp*, basically consists in some slight variation of the translation of the program $\mathcal{P}$ done according to Algorithm 1, that is the program $\Gamma(\mathcal{P})$:

$$\begin{aligned} stop\_engine &\leftarrow f_{r_1'',1}, f_{r_1'',2} \\ f_{r_1'',1} &\leftarrow lighthouse \\ f_{r_1'',1} &\leftarrow imp(lighthouse), not\ \neg lighthouse \\ f_{r_1'',2} &\leftarrow gps \\ f_{r_1'',2} &\leftarrow imp(gps), not\ \neg gps \end{aligned}$$

Evidently, $\Gamma(\mathcal{P})$ is not only intolerably less declarative than $\mathcal{P}$, but also time-wasting to be written. Moreover, we observe that the size of the translation grows with the number of underlined literals occurring in $\mathcal{P}$. Therefore, the above considerations allow us to confirm the effectiveness of our approach.

6. Related Work

The issue of action oriented perception by an agent/robot (possibly with noise from sensors) has been investigated by many authors from the area of Cognitive Robotics. Most of them focus on the integration of sensing actions into the situation calculus [17], event calculus [20,21] or similar logic-based frameworks [4,5].

A general introduction to perception can be found in [17].

Cognitive Robotics has typically adopted one of two views of perception [21]. On the one hand, perception is considered as a black-box process, whereby a sensing action turns raw data from sensors into fluents [13].

On the other hand, perception is a passive process, whereby the robot's model of the world is updated as a side-effect of its physical actions [20]. From the perspective of logic-based agents/robots, the issue of perception in a noisy environment has been covered in [19,18,1,20].

In [19], a logic program gives the consequence of choices made by agents that sense and act within the framework of independent choice logic. Though sensors may be noisy, unreliable or broken, agents consider only their output values and false positives are modelled by means of probabilities. In [18], a model for noisy sensors and actuators, based both on logic programming and a continuous version of probabilistic Horn abduction, is sketched. In [1], uncertainty is modelled by assigning a probability to the agent's beliefs about the state of the world. In [20], noise from sensors and actuators is captured using non-determinism, then abduction is used to supply possible explanations of incoming sensor data.

7. Conclusions

In this paper we extend ASP programs for taking into account possible perception failure of sensors. We have shown by examples that the language is very suitable for the above purpose. Moreover, we have defined the semantics of our language in an *answer-set* fashion and we have given a linear-time algorithm, allowing us to translate any U-program into an equivalent ASP program. This makes our programs easily executable on existing systems that efficiently implement answer set programming.

References

[1] F. Bacchus, J. Y. Halpern, and H. J. Levesque, Reasoning about noisy sensors and effectors in the situation calculus, *Artificial Intelligence* **111**(1–2) (1999), 171–208.

[2] C. Baral and M. Gelfond, Logic Programming and Knowledge Representation, *J. Log. Program.* **19/20** (1994), 73–148.

[3] C. Baral and M. Gelfond, Reasoning agents in dynamic domains, in: *Logic-Based Artificial Intelligence*, Kluwer, 2000, pp. 257–279.

[4] C. Baral and T. C. Son, Approximate reasoning about actions in presence of sensing and incomplete information, in: *Proc. of the Int. Symp. on Logic Programming (ILPS-97)*, MIT Press, 1997, pp. 387–404.

[5] G. De Giacomo, L. Iocchi, L. Nardi, and R. Rosati, Planning with sensing for a mobile robot, LNCS **1348** (1997), 156–168.

[6] M. Gelfond, Answer set programming and the design of deliberative agents., in: *Proc. of the Int. Conf. on Logic Programming (ICLP'04)*, LNCS, Springer, 2004, pp. 19–26.

[7] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing* **9** (1991), 365–385.

[8] M. Gelfond and V. Lifschitz, Representing action and change by logic programs, *Journal of Logic Programming* **17**(2-4) (1993), 301–321.

[9] R. A. Kowalski and F. Sadri, From logic programming towards multi-agent systems, AMAI **25**(3-4) (1999), 391–419.

[10] J. A. Leite, *Evolving knowledge bases*, IOS Press, 2003.

[11] J. A. Leite, J. J. Alferes, and L. M. Pereira, Minerva: A dynamic logic programming agent architecture, in: *Intelligent Agents VIII: 8th Int. Workshop (ATAL-2001)*, LNAI, Springer, 2002, pp. 141–157.

[12] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, The DLV System for Knowledge Representation and Reasoning, *ArXiv Computer Science e-prints* (2002), 11004–+.

[13] H. J. Levesque, What is planning in the presence of sensing?, in: *Proc. of AAAI-96*, 1996, pp. 1139–1146.

[14] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, GOLOG: A logic programming language for dynamic domains, *Journal of Logic Programming* **31**(1-3) (1997), 59–83.

[15] J. McCarthy and P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence* **4** (1969), 463–502.

[16] I. Niemelä and P. Simons, Smodels - an implementation of the stable model and well-founded semantics for normal logic programs, in: *Proc. of the 4th LPNMR*, LNCS, Springer, 1997, pp. 420–429.

[17] F. Pirri and A. Finzi, An approach to perception in theory of actions: Part I, *Linköping Electronic Articles in Computer and Inf. Science* **4**(41) (1999).

[18] D. Poole, Logic programming for robot control, in: *Proc. of the 14th IJCAI*, M. Kaufmann, 1995, pp. 150–157.

[19] D. Poole, Sensing and acting in the independent choice logic, in: *Extending Theories of Action: Formal Theory & Practical Applications: Papers from the 1995 AAAI Spring Symposium*, 1995, pp. 163–168.

[20] M. Shanahan, Noise, non-determinism and spatial uncertainty, in: *Proc. of AAAI-97*, 1997.

[21] M. Shanahan, A logical account of perception incorporating feedback and expectation, in: *Proc. of KR2002*, M. Kaufmann, 2002, pp. 1–13.