Logic Programs with Multiple Chances

Francesco Buccafurri and **Gianluca Caminiti** and **Domenico Rosaci** [1]

Abstract. In human-like reasoning it often happens that different conditions, partially alternative and hierarchically structured, are mentally grouped in order to derive some conclusion. The hierarchical nature of such knowledge concerns with the possible failure of a chance of deriving a conclusion and the necessity, instead of blocking the reasoning process, of activating a subordinate chance. Traditional logic programming (we refer here to Answer Set Programming) does not allow us to express such situations in a synthetic fashion, since different chances of deriving a conclusion must be distributed over different rules, and conditions enabling the switching among chances must be explicitly represented. We present a new language, relying on Answer Set Programming, which incorporates a new modality able to naturally express the above features. The merits of the proposal about the capability of representing knowledge are shown both by examples and by comparisons with other existing formalisms. A translation to plain ASP is finally provided in order to give a practical tool for computing our programs, since a number of optimized ASP evaluation systems are nowadays available.

1 INTRODUCTION

In human-like reasoning it often happens that different conditions are mentally grouped in order to derive some conclusion. Such conditions represent different chances, partially overlapped, exploitable as alternative ways for proceeding in the reasoning process. Moreover, following different kinds of ordering, like reliability, simplicity, cost, and so on, different chances are hierarchically structured, and the failure of a chance of deriving a conclusion, caused by uncertainty conditions, enables the application of a subordinate chance. This way, the reasoning process is not necessarily blocked by uncertainty. Consider for example the diagnostic medical reasoning. The doctor tries to recognize a classical clinical picture, in order to derive a diagnosis, but often such a picture is unclear and incomplete. In this case he typically adopts subordinate ways for reaching the same conclusion, like consulting, ex-adiuvantibus treatments, laboratory tests and so on. Traditional logic programming (we refer here to Answer Set Programming [8]) does not allow us to express such situations in a synthetic fashion, since different chances of deriving a conclusion must be distributed over different rules, and conditions enabling the switching among chances must be explicitly represented.

In this paper, we define a language extending Answer Set Programming (ASP) by including a new modality, allowing us to represent, in a compact and natural fashion, the multi-chances form of reasoning described above. Besides ASP, our programs, called *MC-programs*, are compared also with *Inheritance Logic Programming* [5] and *Nested Logic Programming* [12], that appear to us as the best candidates for representing multiple chances. The former is chosen among languages allowing default reasoning with exceptions and priorized rules [2, 15, 6, 4, 9], the latter is a powerful logic language allowing us to define rules with nested if-then-else constructs. A translation to plain ASP is finally provided in order to give a practical tool for computing our programs, since a number of optimized ASP solvers [10, 14] and optimization techniques [3] are available.

In order to give the flavor of our proposal, we illustrate the following simple MC-program, describing the diagnosis of appendicitis performed by a doctor:[2]

$$app \leftarrow \quad (\underline{fever}, pl, pf, \underline{nausea})[wbc\_test]$$

The structure of the above rule, which we call MC-rule, is the following. First, it embeds a standard logic rule, i.e., $app \leftarrow fever, pl, pf, nausea$, whose meaning is the usual one: If the conjunction $fever$, pl, pf, $nausea$ holds, where pl (resp. pf) means pain localization (resp. pain form), then the diagnosis app (appendicitis) is derived. The MC-rule contains also a subordinate condition, namely wbc_test (denoting the result of a laboratory test counting white blood cells), whose truth is required, in order to derive app, whenever the truth assignment of the elements of the conjunction $fever, pl, pf, nausea$ is recognized as *uncertain*. Such a machinery models the reasoning approach of the doctor, who adopts a subordinate strategy (i.e, the laboratory test) in order to derive the diagnosis, in case the clinical picture allows him neither to exclude such a diagnosis nor to conclude it (i.e., in case of an uncertain clinical picture).

More generally, through an MC-rule of the form $head \leftarrow body[sub\_cond]$, our purpose is to represent a logic rule like $head \leftarrow body$ which is not blocked every time $body$ fails (like in standard ASP), but enables the evaluation of the subordinate condition sub_cond only in case the value of $body$ reflects an uncertain situation. In our example, following a standard approach, we could certainly recognize as uncertain truth assignments just those where at least one literal among $fever, pl, pf, nausea$ is undefined and the others are true[3]. However, the notion of uncertainty we are using now is strictly related to the intended meaning of the rule. As a consequence, it might happen that other truth assignments are recognizable as uncertain, and thus, the occurrence of such assignments has to enable the evaluation of the subordinate condition wbc_test. In particular, truth assignments including some false value, might be good candidates for activating the evaluation of the subordinate condition, whenever such false values actually contain uncertain knowledge (for example, *false negatives* of a test result). Consider for example the literal $fever$. The doctor knows that in case of absence of fever (i.e.,

[1] DIMET, Università Mediterranea di Reggio Calabria, Italy, email: {bucca, gianluca.caminiti, domenico.rosaci}@unirc.it

[2] The same example is encoded in Section 4 both in Answer Set Programming and in Nested Logic Programming.

[3] Recall that, under ASP semantics, it may happen that an intended model contains neither a given literal nor its complementary literal. Throughout this paper the perspective we use in order to capture the above issue is allowing for literals three truth values: true, false and undefined.

whenever $fever$ is false) he cannot exclude the diagnosis of appendicitis (if the rest of the clinical picture holds). The same happens to the symptom $nausea$. Conversely, whenever the localization of the pain is not compatible with the diagnosis of appendicitis (i.e., pl is false) the diagnosis can be excluded (the same happens to the pain form). As a consequence, we should include into the set of truth assignments reflecting uncertain situations (corresponding in our example to an uncertain clinical picture), also those in which $fever$ and $nausea$ are false. This is done, in our formalism, by underlining such literals in the body of the rule. Observe that in case of absence of underlined literals, a conjunction of literals is recognized as uncertain exactly when it is undefined according to the standard meaning (i.e., at least one literal in the conjunction is undefined and the others are true). For instance, according to the above statements, whenever the patient has both a normal temperature, a clear localization of the pain compatible with the diagnosis of appendicitis, an unclear form of pain and, further, he does not have nausea, then the doctor recognizes an uncertain situation not allowing him to exclude the diagnosis. However, such an unclear clinical condition together with the positive result of the laboratory test wbc_test (which is the subordinate condition activated in case of uncertainty), allows the doctor to conclude the diagnosis. The language provides the programmer with another important tool. Some uncertainty configurations, among all possible ones, could be considered non-admissible on the basis of possible relationships among the elementary conditions of a conjunction. In our example, even though the conditions pl and pf may appear separately in admissible uncertainty configurations, the doctor thinks that they cannot be missing simultaneously. In other words, he considers very improbable that neither the localization nor the form of the pain appear in a definite way. This is modeled in our language by defining some *admissibility constraints*, associated to the conjunction of literals. In our example we have just one admissibility constraint $\{pl, pf\}$ encoding what we have described above. The set of admissibility constraints is inserted just after the conjunction on which they operate. The resulting MC-program is the following:

$$app \leftarrow \quad (\underline{fever}, pl, pf, \underline{nausea})\{\{pl, pf\}\}[wbc\_test] \qquad (1)$$

The plan of the paper is the following. Section 2 describes syntax and semantics of the language. In Section 3 the features of the language are shown by real-life examples. In Section 4 our capability of representing knowledge is compared with other existing approaches. Section 5 describes the translation of our language into Answer Set Programming. Finally, we draw our conclusions in Section 6.

2 SYNTAX AND SEMANTICS

In this section, after a brief recall of basic concepts about *Answer Set Programming* (ASP) [8], we introduce the syntax of logic programs *with multiple chances*, said *MC-programs*, and then we give its semantics, that is an adaptation of the notion of *answer sets*[4].

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, $\neg a$ is defined as $\neg p$, if $a = p$ and p if $a = \neg p$. A set L of literals is *consistent* if $\forall l \in L, \neg l \notin L$. Given a literal a, the formula $not\ a$ is the *negation as failure* (NAF) of a[5].

[4] For the sake of presentation, we refer to variable-free (also said *ground*) programs. The extension to the general case is straightforward.

[5] Observe that the NAF of negative literals is allowed.

Definition 1 Given $m \geq 0$ and $k \geq 0$, an *MC-formula* β is a formula $(\alpha_1[\gamma_1], \cdots, \alpha_m[\gamma_m])\{T_1, \cdots, T_k\}$, where (1) α_j $(1 \leq j \leq m)$ is (possibly the NAF of) a literal and may appear underlined (2) T_i $(1 \leq i \leq k)$ is a subset of $\{\alpha_j \mid (\alpha_j$ is either underlined, or is not the NAF of a literal$) \wedge 1 \leq j \leq m\}$, and (3) γ_j $(1 \leq j \leq m)$ is a (possibly empty) conjunction of MC-formulas. Each set T_i $(1 \leq i \leq k)$ is called *admissibility constraint* and the (possibly empty) set $\{T_1, \cdots, T_k\}$ of admissibility constraints of β is denoted by $S(\beta)$. γ_j $(1 \leq j \leq m)$ is said the *second-chance* of α_j.

The recursive definition above is a generalization of the standard definition of the conjunction of (possibly the NAF of) literals occurring in the body of an ASP logic rule, where both (1) a second-chance is (possibly) associated to each literal, (2) literals may appear underlined and (3) a set of admissibility constraints is associated to the conjunction. As we will explain in the following, the second-chance may be viewed like a substitution of the α_j to which is applied. Such a substitution is executed only when α_j is *uncertain* (the notion of uncertainty will be introduced in the following – the underlining is relating with this notion). Due to the recursive structure of the above definition, a second-chance is, in turn, a conjunction of MC-formulas, and thus may contain further subordinate chances (i.e., the nesting of chances is allowed). Intuitively, second-chances are subordinate conditions that are required, in order to satisfy the MC-formula, whenever the basic conditions α_is are uncertain. Actually, not all the possible configurations of values making α_is uncertain are allowed, in order to activate the substitution of the second-chance, but only those satisfying the admissibility constraints.

Given a conjunction of MC-formulas Δ, in favor of simplicity, we often write $(\alpha_1[\gamma_1], \cdots, \alpha_m[\gamma_m])\{T_1, \cdots, T_k\}[\Delta]$ to denote $(\alpha_1[\gamma_1[\Delta]], \cdots, \alpha_m[\gamma_m[\Delta]])\{T_1, \cdots, T_k\}$. In particular, if $\gamma_1 = \cdots = \gamma_m = \Delta$, we write $(\alpha_1, \cdots, \alpha_m)\{T_1, \cdots, T_k\}[\Delta]$ to denote $(\alpha_1[\Delta], \cdots, \alpha_m[\Delta])\{T_1, \cdots, T_k\}$. Clearly, square brackets of empty second-chances are omitted. Examples of MC-formulas are $\beta_1 = (a, not\ b)[c]$ (denoted also by $(a[c], not\ b[c])$), where $S(\beta_1) = \emptyset$ and $\beta_2 = (a, b)\{\{a, b\}\}[c]$, (denoted also by $(a[c], b[c])\{\{a, b\}\}$), where $S(\beta_2) = \{\{a, b\}\}$.

Definition 2 An *MC-rule* r is a formula $a \leftarrow \beta_1, \cdots, \beta_n$ $(n \geq 0)$, where a is a positive literal and β_i is an MC-formula, for each $1 \leq i \leq n$. The set $\{a\}$, denoted by $head(r)$, is called the *head* of r, and the set $\{\beta_1, \cdots, \beta_n\}$, denoted by $body(r)$, is called the *body* of r. An *MC-program* is a finite set of MC-rules.

Note that since an MC-formula is a generalization of a standard conjunction of (the NAF of) literals, a standard ASP rule is a special case of an MC-rule r. Informally, an MC-rule is a logic rule allowing in the body the substitution of some literal with its second-chance (under uncertainty conditions). An example of MC-rule is the expression (1) reported in Section 1. Another example of MC-rule is: $a \leftarrow b[c[d]], f[not\ g]$.

We introduce now the intended models of our semantics. First we need some preliminary definitions. Given an (MC-)program P, Lit^P is the set of literals occurring in P. An *interpretation* I of an (MC-)program P is a consistent subset of Lit^P. A literal a is *true* w.r.t. I if $a \in I$, it is *false* w.r.t. I if $\neg a \in I$. A literal a is *undefined* w.r.t. I if it is neither true nor false w.r.t. I. Given a literal a, a formula $not\ a$ is *true* w.r.t. I if $a \notin I$, it is *false* w.r.t. I otherwise (observe that the NAF of a literal cannot be undefined w.r.t. a given interpretation). From now on in this section, consider given an (MC-)program P and an interpretation I of P.

We introduce now a basic notion of our framework, that is the notion of *uncertainty* of an element of an MC-formula. Indeed the mechanism of substitution of an element of an MC-formula with its second-chance is founded on this property (this will be explained in detail in the following).

Given an MC-formula $\beta = (\alpha_1[\gamma_1], \cdots, \alpha_m[\gamma_m])\{T_1, \cdots, T_k\}$, we say that α_j ($1 \leq\ j \leq m$) is *uncertain* w.r.t. the interpretation I if either (i) $\alpha_j = a \mid a \in Lit^P \wedge \{a, \neg a\} \cap I = \emptyset$, (ii) $\alpha_j = \underline{a} \mid a \in Lit^P \wedge a \notin I$, or (iii) $\alpha_j = \underline{not\ a} \mid a \in Lit^P \wedge a \in I$.

The definition of uncertainty introduced above arises from the following reasoning. We expect that an element α_j is uncertain if it is undefined. This is captured by the item (i) of the definition. Note that, correctly, this item does not include the case of the NAF of a literal, since such a formula cannot be undefined. Now, when we want to interpret as uncertain also the false value of α_j, we require that α_j is underlined (items (ii) and (iii)). Thanks to this mechanism, also the NAF of a literal can be uncertain, whenever it appears in an underlined element $\alpha_j = \underline{not\ a}$, and a is true.

Consider again the MC-formula β introduced above. $S(\beta)$ is *true* w.r.t. I if for each $T_i \in S(\beta)$ ($1 \leq\ i \leq k$), there exists $\alpha \in T_i$ such that α is not uncertain w.r.t. I. In words, an admissibility constraint T_i states that literals occurring in it cannot appear simultaneously in uncertainty configurations. Now we define the intended models of our semantics. First, we introduce a transformation for MC-programs which produces an ASP program.

Definition 3 We define the *MC-transformation* of the program P w.r.t the interpretation I as the ASP program $\bar{P}^I$, obtained from P by executing Algorithm 1.

Algorithm 1 (The MC-transformation algorithm)
repeat
  for each MC-rule $r \in P$ **do**
    if $\exists \beta \in body(r) \mid S(\beta)$ is false w.r.t. I **then** delete r
    else for each $\beta = (\alpha_1[\gamma_1], \cdots, \alpha_m[\gamma_m])\{T_1, \cdots, T_k\} \in body(r)$ **do**
      delete $S(\beta)$
      for each $1 \leq\ j \leq m$ **do**
        if α_j is uncertain w.r.t. I **then** remove from α_j the underlining (if any)
          if γ_j is not empty **then** replace $\alpha_j[\gamma_j]$ by γ_j
          end if
        else remove from α_j the underlining (if any); replace $\alpha_j[\gamma_j]$ by α_j
        end if
      end for
    end for
  end if
end for
until $\forall r \in P, \forall \beta \in body(r)\ \exists a \in Lit^P \mid \beta = a \vee \beta = not\ a$

The MC-transformation of P consists of both (i) deleting all MC-rules of P whose body includes some MC-formula with false admissibility constraint, (ii) deleting all admissibility constraints from the remaining MC-formulas, (iii) for each uncertain element α_j in the body of every MC-rule, removing the underlining (if any), and replacing α_j by its second-chance (if any); (iv) for each remaining element α_j, removing the underlining (if any) and discarding its second-chance. The loop ends when only (possibly the NAF of) literals occur in P.

Definition 4 An interpretation J of the program P is an *answer set* of P if J is an answer set[6] of $\bar{P}^J$.

For instance, consider the following MC-program P:

[6] The definition of *answer sets* of an ASP program can be found in [8]. We do not report it here for space limitations. Note that, according to the definition of interpretation, we want to limit our focus only on consistent answer sets.

$r_1 : a \leftarrow (\underline{b}, c, e)\{\{c, e\}, \{\underline{b}, c\}\}[not\ d]$
$r_2 : d \leftarrow not\ a$ $\qquad$ $r_3 : \neg b \leftarrow a$
$r_4 : c \leftarrow not\ d$ $\qquad$ $r_5 : e \leftarrow$

The intended answer sets of P are: $\{a, \neg b, c, e\}, \{d, e\}$. Indeed, given $I_1 = \{a, \neg b, c, e\}$, the MC-transformation of P w.r.t. I_1 is $\bar{P}^{I_1} = \{r'_1, r_2, r_3, r_4, r_5\}$, where r'_1 is $a \leftarrow not\ d, c, e$. Observe that I_1 is an answer set of $\bar{P}^{I_1}$ thought as an ASP program. Likewise, it is easy to see that $I_2 = \{d, e\}$ is an answer set of $\bar{P}^{I_2} = \{r_2, r_3, r_4, r_5\}$ (r_1 is deleted due to the constraint $\{\underline{b}, c\}$).

Remark. Our formalism can be viewed as an extension of the ASP negation by failure. Indeed, the rule $h \leftarrow not\ b$ can be rewritten in our setting as $h \leftarrow \neg b[true]$. According to our semantics, h is derived if either b is false (i.e., $\neg b$ is true) or b is undefined (since, in this case the subordinate condition is activated and it corresponds to the constant $true$), exactly as the ASP rule $h \leftarrow not\ b$.

3 KNOWLEDGE REPRESENTATION BY MC-PROGRAMS

In this section, we show how our language can be used for naturally representing real-life situations. We show two examples, where we do not make explicit the answer sets of the programs, because they depend on the value of a number of literals, which we assume as "external variables". However, the purpose of this section is to make evident the merits of the multi-chance constructs. Therefore in the following description we just stress the aspect of the meaning of MC-rules, instead of considering the issues typically occuring in the context of answer sets (like their multiplicity), which our language completely preserves.

A Medical Example. We consider here a fragment of a diagnostic reasoning process in a medical setting. We model, by means of the following MC-program, the reasoning steps leading to both the diagnosis of *ischemic cardiopathy* (rules $r_1 - r_4$) and the consequent therapy (rules $r_5 - r_8$).

r_1 : $clin\_pict \leftarrow$ $complete\_set\_of\_symptoms$
r_2 : $\neg clin\_pict \leftarrow$ $absent\_symptoms$
r_3 : $ECG\_test \leftarrow$ $ST\_segment\_anomaly$
r_4 : $isch\_cardio \leftarrow$ $(clin\_pict, \underline{N\_test}, \underline{ECG\_test}[\underline{PPI\_test}])$ $\{\{clin\_pict, \underline{N\_test}\}\}[\underline{enz\_test}[echo]]$
r_5 : $active\_ulcer \leftarrow$ $recent\_symptoms, pos\_case\_history$
r_6 : $\neg active\_ulcer \leftarrow$ $\neg pos\_case\_history$
r_7 : $risk\_patient \leftarrow$ $smoker, diabetic, hypertensive$
r_8 : $aspirin \leftarrow$ $isch\_cardio,$ $\neg active\_ulcer[risk\_patient]$

In particular, rules r_1 and r_2 specify that the patient's clinical picture ($clin\_pict$) is either (i) true, if there exists a complete set of symptoms, or (ii) false, if all symptoms are absent. Otherwise $clin\_pict$ is considered uncertain. Rule r_3 describes that the electrocardiogram (ECG) test (ECG_test) is positive if it reveals a typical anomaly of its ST-segment. Thus, by means of rule r_4, the doctor concludes the diagnosis of ischemic cardiopathy ($isch\_cardio$) in case of both a complete clinical picture, the patient's positive reaction to a nitrate treatment (N_test), and a positive result of the ECG. In this case, rule r_4 behaves as the following standard ASP rule: $isch\_cardio \leftarrow clin\_pict, N\_test, ECG\_test$.

However, a possible uncertainty situation might arise from a negative ECG: The doctor knows that there are cases of ischemic cardiopathy not altering the ST-segment of the ECG (i.e. ECG_test is false). In such cases he applies a second-chance approach, by substituting the information given by the ECG test with that given by

another ex-adiuvantibus treatment, i.e. the administration of a protonic pump inhibitor (PPI_test). The patient's positive reaction to such a test, reveals a different kind of pathology. Thus, PPI_test is assumed true whenever symptoms persists after the administration of the inhibitor. After such a substitution, rule r_4 behaves as: $isch\_cardio \leftarrow clin\_pict, N\_test, PPI\_test$.

Unfortunately, there are other situations that may appear still unclear: (i) The clinical picture might be not complete ($clin\_pict$ may be undefined)[7]. (ii) The nitrate treatment might give either an unclear result (i.e., symptoms may weakly persist) or false negatives (N_test may be either undefined or false, respectively). (iii) Symptoms might weakly persist after the protonic-pump-inhibitor-based treatment (i.e., PPI_test may be undefined).

Observe that the admissibility constraint $\{\{clin\_pict, \underline{N\_test}\}\}$ in rule r_4 means that the simultaneous occurrence of both case (i) and (ii) is not compatible with the diagnosis of ischemic cardiopathy.

In either case (i), (ii) or (iii), the doctor decides to perform a laboratory test, which is aimed to detect specific enzymes in the blood. The result of such a test is represented by enz_test. Accordingly, rule r_4 behaves as: $isch\_cardio \leftarrow enz\_test$.

However, since this test might give false negatives, the doctor evaluates the nested second-chance ($echo$), corresponding to the result of a echocardiogram. Thus, rule r_4 behaves as: $isch\_cardio \leftarrow echo$.

In order to apply the therapy, the doctor reasons about the patient's history. In particular, he is interested in the possible presence of an active ulcer (rules r_5 and r_6). Rule r_5 states that the patient is affected by an active ulcer ($active\_ulcer$), if both the doctor recognizes the occurrence of recent symptoms and the patient's history clearly reveals previous cases of ulcer. Rule r_6 excludes the occurrence of an active ulcer in case of a clearly negative patient's history. However, it may happen that the history reported by the patient is unclear, so that the information about such a pathology remains undefined. Moreover, rule r_7 specifies under which conditions the patient has to be considered a patient with a high cardiovascular risk ($risk\_patient$).

Finally, rule r_8 describes the therapy ($aspirin$) to be administered in case of both ischemic cardiopathy and in absence of an active ulcer ($\neg active\_ulcer$). However, if the latter information is undefined, then the doctor replaces the condition $\neg active\_ulcer$ by $risk\_patient$, i.e. he decides to administer the aspirin to the patient only if both the diagnosis is certain and the patient has a high cardiovascular risk. Accordingly, rule r_8 behaves as: $aspirin \leftarrow isch\_cardio, risk\_patient$. Observe that, in this case, the doctor judges the risk of a both non-adequate and non-immediate therapy for the ischemic cardiopathy to be more relevant than the probability of side effects due to the interaction of aspirin and a (possible, but not evident) active ulcer.

The Driving Licence. A guy must go to a far town for renewing his driving license. The renewal takes a few minutes, but in order to apply for it, two documents A and B are required by the administration office. Moreover, A and B are issued by two different offices, say O_A and O_B, that are placed in that town. He is told that A and B will be available in about 15 days. Meanwhile, he can check by phone for both A and B to be ready. However, it is possible that either both O_A and O_B phone lines are busy, or the officers cannot answer. The guy wants to go to the administration office only if he knows that both offices O_A and O_B have issued the respective documents. However, if after 20 days he is sure about the availability of at least one document, and he was not informed of the contrary about the other document, then he goes. This example may be represented by the following MC-program:

$$go \leftarrow (rdy\_A, rdy\_B)\{\{rdy\_A, rdy\_B\}\}[waiting(20)]$$

where we assume that rdy_A (resp. rdy_B) is derived as a fact if the guy has been told by phone that A (resp. B) is ready, $\neg rdy\_A$ (resp. $\neg rdy\_B$) is derived as a fact if the guy has been told by phone that A (resp. B) is not ready and $waiting(20)$ is derived as a fact if the guy has waited for 20 days.

[7] Observe that if $\neg clin\_pict$ holds, then such a case is not considered as a font of uncertainty. That is, if no symptom is present, then the doctor is able to exclude the diagnosis of ischemic cardiopathy.

4 COMPARISON WITH OTHER APPROACHES

In this section first we compare by examples our language with plain ASP, in order to show that the introduction of the new modality in ASP gives real benefits. Then, we compare our programs also with *Inheritance Logic Programming* [5] and *Nested Logic Programming* [12], that appear, to our knowledge, as the best candidates among logic programming languages for representing multiple chances.

Plain ASP. Consider now the MC-program (1) presented at the end of Section 1 and encode the same real-life situation using Answer Set Programming. Since we need to make explicit all the possible combinations of uncertain elements of the conjunction (which number grows exponentially with the number of uncertain cases), the resulting ASP program is the following:

$app \leftarrow fever, pl, pf, nausea$
$app \leftarrow not\ fever, pl, pf, nausea, wbc\_test$
$app \leftarrow not\ fever, pl, pf, not\ nausea, wbc\_test$
$app \leftarrow not\ fever, pl, not\ pf, not\ \neg pf, nausea, wbc\_test$
$app \leftarrow not\ fever, not\ pl, not\ \neg pl, pf, nausea, wbc\_test$
$app \leftarrow not\ fever, not\ pl, not\ \neg pl, pf, not\ nausea, wbc\_test$
$app \leftarrow not\ fever, pl, not\ pf, not\ \neg pf, not\ nausea, wbc\_test$
$app \leftarrow fever, pl, pf, not\ nausea, wbc\_test$
$app \leftarrow fever, pl, not\ pf, not\ \neg pf, nausea, wbc\_test$
$app \leftarrow fever, not\ pl, not\ \neg pl, pf, nausea, wbc\_test$
$app \leftarrow fever, not\ pl, not\ \neg pl, pf, not\ nausea, wbc\_test$
$app \leftarrow fever, pl, not\ pf, not\ \neg pf, not\ nausea, wbc\_test$

This example shows that even though our formalism does not extend the expressive power of ASP (and does not introduce asymptotic computational cost), it has evident merits as far as its capability of representing multi-chance reasoning is concerned. Concerning the complexity and the expressiveness of logic programming, see [7].

Nested Logic Programming. *Nested Logic Programming* [12] (NLP) is a class of logic programs, where arbitrarily nested expressions – formed from literals by using negation as failure, conjunction and disjunction (;) – are allowed in both the bodies and heads of rules. Given an MC-program P, a suitable translation P' into the non-disjunctive fragment of NLP (with no NAF in heads of rules) exists, but we show that P' is extremely tedious to write and difficult to read. For instance, if we translate the MC-program (1) of Section 1, we obtain the following NLP program:

$$\begin{aligned} app \leftarrow\ & (fever; not\ fever, wbc\_test), \\ & (pl; not\ pl, not\ \neg pl, (pf; \neg pf), wbc\_test), \\ & (pf; not\ pf, not\ \neg pf, (pl; \neg pl), wbc\_test), \\ & (nausea; not\ nausea, wbc\_test) \end{aligned}$$

Disjunctive Logic Programs with Inheritance. *Disjunctive Logic Programs with Inheritance* [5] (DLP$^<$) is a knowledge representation language extending disjunctive logic programming (with strong negation) by inheritance. The addition of inheritance allows us a natural representation of default reasoning with exceptions. Thus, one could argue that DLP$^<$ may be used to represent default conditions being evaluated whenever previously required conditions are uncertain. We have compared by examples our language with DLP$^<$. For space limitations we do not report details of such a comparison here.

Synthetically, we have tested that DLP$^<$, compared to our language, is not suitable to represent multi-chance reasoning. Indeed base conditions must be thought as exceptions and (nested) chances as defaults and, further, programs are time-wasting to write and difficult to read, since all the combinations, i.e. those resulting from the possible uncertainty of the literals enclosed in an MC-formula, must be individually considered.

5 TRANSLATION TO ASP

In this section we show that for any MC-program P, an ASP program $\Gamma(P)$ exists such that there is a one-to-one correspondence between the answer sets of P and the answer sets of $\Gamma(P)$. Such a translation allows us to evaluate any MC-program by exploiting one of the existing answer set solvers (DLV [10], Smodels [14], etc).

Given an MC-program P, we define $\Gamma(P)$ as the ASP program obtained from P by executing Algorithm 2.

Algorithm 2 (The translation algorithm)
for each MC-rule $r \in P$ **do**
  for each MC-formula $\beta \in body(r) \mid \beta$ is not (the NAF of) a literal **do**
    RESOLVE_FORMULA$(r, \beta, \bar{\beta})$
  end for
end for

The core of Algorithm 2 is the recursive procedure described by Algorithm 3, where MC-formulas are of the form $(\alpha_1[\gamma_1], \cdots, \alpha_m[\gamma_m])\{T_1, \cdots, T_k\}$ $(m \geq 0, k \geq 0)$.

Algorithm 3 (The procedure RESOLVE_FORMULA**)**
procedure RESOLVE_FORMULA$(r, \beta, \bar{\beta})$
 $body(r) = body(r) \setminus \{\beta\} \cup \{\bar{\beta}\}$
 $P = P \cup \{\bar{\beta} \leftarrow \rho_1, \cdots, \rho_m, not\ \tau_1, \cdots, not\ \tau_k\} \cup$
  $\cup\{\tau_i \leftarrow \sigma_1, \cdots, \sigma_{|T_i|} \mid 1 \leq i \leq k\}$
 for $1 \leq j \leq k$ **do** let $c_1, \cdots, c_{|T_i|}$ be a permutation of T_i
  for $1 \leq l \leq |T_i|$ **do**
   if $(c_l = b) \wedge (b \in Lit^P)$ **then** $P = P \cup \{\sigma_l \leftarrow not\ b, not\ \neg b\}$
   else if $(c_l = \underline{b}) \wedge (b \in Lit^P)$ **then** $P = P \cup \{\sigma_l \leftarrow not\ b\}$
   else if $(c_l = \underline{not\ b}) \wedge (b \in Lit^P)$ **then** $P = P \cup \{\sigma_l \leftarrow b\}$
   end if
  end for
 end for
 for $1 \leq j \leq m$ **do**
  if $(\alpha_j = b \vee \alpha_j = \underline{b}) \wedge (b \in Lit^P)$ **then** $P = P \cup \{\rho_j \leftarrow b\}$
   if γ_j is not empty **then**
    if $(\alpha_j = b) \wedge (b \in Lit^P)$ **then** let $t : \rho_j \leftarrow not\ b, not\ \neg b, \gamma_j$
    else if $(\alpha_j = \underline{b}) \wedge (b \in Lit^P)$ **then** let $t : \rho_j \leftarrow not\ b, \gamma_j$
    end if
    $P = P \cup \{t\}$; RESOLVE_FORMULA$(t, \gamma_j, \bar{\gamma_j})$
   end if
  else if $(\alpha_j = not\ b \vee \alpha_j = \underline{not\ b}) \wedge b \in Lit^P$ **then**
    $P = P \cup \{\rho_j \leftarrow not\ b\}$
   if (γ_j is not empty) $\wedge$ $(\alpha_j = \underline{not\ b}) \wedge (b \in Lit^P)$ **then**
    let $t : \rho_j \leftarrow b, \gamma_j$; $P = P \cup \{t\}$; RESOLVE_FORMULA$(t, \gamma_j, \bar{\gamma_j})$
   end if
  end if
 end for
end procedure

Note that for each call of the procedure, $\rho_i, \tau_i, \sigma_i, \bar{\gamma_i}(\forall i)$ and $\bar{\beta}$ are fresh literals (not already included in Lit^P). Observe that the Algorithm 2 is exponential in the maximum number, say c, of nested chances. However, c is typically bound (informally, the number of nested chances is small). Therefore it becomes meaningful the complexity analysis w.r.t. n, that is the number of literals occurring in the program, keeping constant c. It is easy to see that such a complexity is $O(n)$. Moreover, in such a case, the computational complexity still remains linear w.r.t. both the number of program rules and the maximum number of elements in the body of an MC-rule.

In the next theorem we state the equivalence between the original MC-program P and its translation $\Gamma(P)$. The proof of the theorem is omitted for space limitations.

Theorem 1 Given an MC-program P and an interpretation I of P, then I is an answer set of P iff $I \in AS^{dep}(\Gamma(P))$, where $AS^{dep}(\Gamma(P))$ is the set of answer sets of $\Gamma(P)$ where all working literals $\bar{\beta}, \rho_i, \tau_i, \sigma_i, \bar{\gamma_i}(\forall i)$ are discarded.

6 CONCLUSIONS

The paper presents a new language relying on Answer Set Programming and including some new constructs useful for naturally representing some forms of reasoning where multiple chances of deriving a given conclusion occur. Examples described in the previous sections as well as comparisons with other languages show that the above goal is satisfactorily reached. The general KR point of view adopted in this work could represent a starting point for designing more specific solutions, once a particular research setting is chosen (for example, planning [13, 16, 11, 1]). We feel this is a very interesting direction for our future research.

ACKNOWLEDGMENTS

We are grateful to Dr. Umberto Buccafurri for the useful suggestions given us during the preparation of medical examples.

REFERENCES

[1] M. Balduccini, M. Gelfond, R. Watson, and M. Nogueira, 'The USA-Advisor: A Case Study in Answer Set Planning.', in *LPNMR*, volume 2173 of *LCNS*, pp. 439–442. Springer, (2001).
[2] G. Brewka and T. Eiter, 'Preferred Answer Sets for Extended Logic Programs', *Artif. Intell.*, **109**(1-2), 297–356, (1999).
[3] G. Brewka, I. Niemelä, and M. Truszczynski, 'Answer Set Optimization.', in *IJCAI*, pp. 867–872, (2003).
[4] G. Brewka, I. Niemelä, and M. Truszczynski, 'Prioritized Component Systems.', in *AAAI05*, pp. 596–601, (2005).
[5] F. Buccafurri, W. Faber, and N. Leone, 'Disjunctive Logic Programs with Inheritance.', *Theory and Practice of Log. Program.*, **2**(3), (2002).
[6] F. Buccafurri, N. Leone, and P. Rullo, 'Disjunctive Ordered Logic: Semantics and Expressiveness.', in *Proc. of KR'98*, pp. 418–431, (1998).
[7] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, 'Complexity and Expressive Power of Logic Programming.', *ACM Comput. Surv.*, **33**(3), 374–425, (2001).
[8] M. Gelfond and V. Lifschitz, 'Classical Negation in Logic Programs and Disjunctive Databases', *New Generation Computing*, **9**, (1991).
[9] M. Gelfond and T. C. Son, 'Reasoning with Prioritized Defaults.', in *LPKR*, pp. 164–223, (1997).
[10] 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*, 11004–+, (2002).
[11] V. Lifschitz, 'Answer Set Programming and Plan Generation.', *Artif. Intell.*, **138**(1-2), 39–54, (2002).
[12] V. Lifschitz, L.R. Tang, and H. Turner, 'Nested Expressions in Logic Programs', *Annals of Mathematics and Artif. Intell.*, **25**(3-4), (1999).
[13] A. Nareyek, E. C. Freuder, R. Fourer, E. Giunchiglia, R. P. Goldman, H. A. Kautz, J. Rintanen, and A. Tate, 'Constraints and AI Planning.', *IEEE Intelligent Systems*, **20**(2), 62–72, (2005).
[14] 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, pp. 420–429. Springer, (1997).
[15] C. Sakama and K. Inoue, 'Prioritized Logic Programming and Its Application to Commonsense Reasoning.', *Artif. Intell.*, **123**(1-2), (2000).
[16] T. C. Son, P. H. Tu, M. Gelfond, and A. R. Morales, 'Conformant Planning for Domains with Constraints-A New Approach.', in *Proc. of The XXth National Conf. on Artif. Intell. and the 17th Innovative Applications of Artif. Intell. Conf.*, pp. 1211–1216, (2005).