An Implementation of a Non-monotonic Logic in an Embedded Computer for a Motor-glider

José Luis Vilchis Medina   Pierre Siegel   Vincent Risch   Andrei Doncescu

Aix-Marseille Univ, Universit de Toulon, CNRS, LIS
Marseille, France

LAAS-CNRS
Toulouse, France

{joseluis.vilchismedina,pierre.siegel,vincent.risch}@lis-lab.fr   andrei.doncescu@laas.fr

In this article we present an implementation of nonmonotonic reasoning in an embedded system. As a part of an autonomous motor-glider, it simulates piloting decisions of an airplane. A real pilot must take care not only about the information arising from the cockpit (airspeed, altitude, variometer, compass. . .) but also from outside the cabin. Throughout a flight, a pilot is constantly in communication with the control tower to follow orders, because there is an airspace regulation to respect. In addition, if the control tower sends orders while the pilot has an emergency, he may have to violate these orders and airspace regulations to solve his problem (e.g. emergency landing). On the other hand, climate changes constantly (wind, snow, hail. . .) and can affect the sensors. All these cases easily lead to contradictions. Switching to reasoning under uncertainty, a pilot must make decisions to carry out a flight. The objective of this implementation is to validate a nonmonotonic model which allows to solve the question of incomplete and contradictory information. We formalize the problem using default logic, a nonmonotonic logic which allows to find fixed-points in the face of contradictions. For the implementation, the Prolog language is used in an embedded computer running at 1 GHz single core with 512 Mb of RAM and 0.8 watts of energy consumption.

1 Introduction

In this article we present an implementation of the calculation of extensions of a default theory in an embedded computer . The practical case is about rules of piloting an airplane. This is a complex human activity in terms of management of the rules of legislation, control tower, environment, risks. . . A pilot needs to manage these rules, plus taking into account informations from the cockpit that changes every time. A cockpit is composed of six instruments such as airspeed, altimeter, compass, variometer, turn bank and artificial horizon. All this gives the states of the airplane to the pilot. In order to tackle the possible contradictions in these changing informations, we use default logic [8], a nonmonotonic way of reasoning which is a manner to represent the way of human reasoning [2, 5, 6]. Especially, default logic gains the benefits of an interpretive semantics [1, 4, 10]. The practical interest of this paper is to implement the computation of extensions of a default theory with minimum requirements of energy consumption and effective time of computation. One of the most studied logic programming language in theoretical and practical terms is Prolog [3, 11]. Logic programming is a programming paradigm which is based on facts and rules describing a problem in a particular domain. For this implementation Prolog is used. Prolog uses a kind of default assumption when treating negation: for instance, if a negative literal cannot be proved to be true, then it is assumed to be false. Prolog is based on a fragment of first-order logic (FOL) where rules are in the form of clauses: $H : - B_1, B_2, \cdots, B_n.$ with H as the head, the symbol $: -$ as "if" and $B_1, B_2, \cdots, B_n$ as the body. Facts are clauses without body: H[1]. The motivation of this

---

[1]Prolog is based on *Horn clauses*. A Horn clause is a clause with at most one positive literal, this is a subset of FOL.

implementation is to have Prolog running in a microcomputer to be able to calculate extensions. Later, this can be incorporate in a mobile system such as a motor-glider to simulate the decisions of the pilot.

1.1   Classical Logic

Logic is a particular way of thinking, that focus on the formal principles of inference, and hence on consequences from given axioms. Formal systems, e.g., propositional, predicate, modal...logics are symbolic constructions in a particular language which allows to express different ways to deal with a conclusion [9]. The language of propositional logic is defined as the least set of expressions satisfying: $\top$ (true) and $\bot$ (false) are formulas, and if A and B are formulas, so are $\neg A$ (not A), $A \wedge B$ (A and B), $A \vee B$ (A or B) and $A \rightarrow B$ (A implies B). A *proposition* can be any sentences, e.g., "It's a sunny day", "Robert can pilot his airplane". Propositional variables are denoted by a, b, c...The sentence "It's a sunny day" could be represented by A, and the other sentence "Robert can pilot his airplane" by B. It can be composed to create complex sentences, e.g., "It's a sunny day and Robert can pilot his airplane", resulting: $A \wedge B$. First-Order Logic (FOL) or predicate logic is an extension of propositional logic that includes universal and existential quantifiers, respectively $\forall$ and $\exists$, over individuals. Predicates are used to denote properties over individuals e.g. $P(x)$, $Q(x,y)$, ...As such, FOL is very expressive, and a very convenient way to formalize sentences. For instance, we can formalize the next sentence: "all airplanes land on wheels", with the following rule:

$$\forall y, Airplane(y) \rightarrow Land\_on\_wheels(y) \tag{1}$$

But we also know that some floatplanes are airplanes that do not land on wheels and some airplanes use skis to land on ice or snow. So, we have the following rules:

$$\forall y, Ski\_airplane(y) \rightarrow Airplane(y) \tag{2}$$

$$\forall y, Ski\_airplane(y) \rightarrow \neg Land\_on\_wheels(y) \tag{3}$$

We can see that formalizations (1) and (3) are contradictory. This is because the inference in classical logic is monotonic. This property is very important in the world of mathematics, because it allows to describe lemmas previously demonstrated. But this property cannot be applied in situation where uncertain, incomplete information or exceptions have to be considered. In such situations, we would expect that by adding new information or set of formulas to a model, the set of consequences of this model might be reduced. Since the property of monotony is: $A \vdash w$ then $A \cup B \vdash w$, the problem leads directly to the general representation of common sense reasoning. By moving to a nonmonotonic framework, we can carry out the principle of explosion and nevertheless reach a conclusion.

1.2   Default Logic

Default logic is one of the best known formalization for commonsense reasoning, founded by Raymond Reiter. This kind of formalization allows to infer arguments based on partial and/or contradictory information as premises [7]. A default theory is a pair $\Delta = (D, W)$, where D is a set of defaults and W is a set of formulas in FOL. A default d is: $\frac{A(X):B(X)}{C(X)}$, where $A(X), B(X), C(X)$ are well-formed formulas. $A(X)$ are the *prerequisites*, $B(X)$ are the *justifications* and $C(X)$ are the *consequences*. Where $X = (x_1, x_2, x_3, \ldots, x_n)$ is a vector of (non-quantified) free variables. Intuitively a default means, "if $A(X)$ is true, and there is no evidence that $B(X)$ might be false, then $C(X)$ can be true". The use of defaults

implies the generation of sets containing the consequences of these defaults. Such set are called extensions. An extension can be seen as a set of beliefs of acceptable alternatives. Formally, an extension of a default theory Δ is a smallest fixed-point E for which the following property holds: If d is a default of D, whose the prerequisite is in E, and the negation of its justification is not in E, then the consequent of d is in E [7].

Definition 1.1. Let $\Delta = (D,W)$, an *extension E* of Δ is define:

- $E = \bigcup_{i=0}^{\infty} E_i$ with:
- $E_0 = W$ and,
- for $i > 0$; $E_{i+1} = Th(E_i) \cup \{C(X) \mid \frac{A(X):B(X)}{C(X)} \in D, A(X) \in E_i, \neg B(X) \notin E\}$

Here $Th(E_i)$ is the set of formulas derived from E_i. A special case concerns *normal default theories*, having only defaults of the form: $\frac{A(X):C(X)}{C(X)}$. The main characteristic of such default theories is that at least one extension is always guaranteed. The original version of the definition of an extension is difficult to compute in practice, since the condition $\neg B \notin E$ assumes that E is known, while E is not yet calculated. In the case of normal defaults, we simply check that E is an extension of Δ by replacing $\neg B(X) \notin E$ by $\neg C(X) \notin E_i$. Regarding the rules (1) and (3), we can generalize the sentence *"all airplanes land on wheels"* by "*generally, airplanes land on wheels*". Having a default theory that is composed of $D = \{\frac{Airplanes(y):Land\_on\_wheels(y)}{Land\_on\_wheels(y)}\}$, and a knowledge about airplanes: $W = \{Floatplane(y) \rightarrow Airplane(y), Floatplane(y) \rightarrow \neg Land\_on\_wheels(y)\}$. Using D, we can note that the prerequisite $Airplane(y)$ is true and the justification $Land\_on\_wheels(y)$ is inconsistent with W, because of $Floatplane(y) \rightarrow \neg Land\_on\_wheels(y)$, then we can not conclude that floatplanes land on wheels. But we know that some floatplanes have wheels, formally, $W \cup \{Floatplane\_wheels(y) \rightarrow Airplane(y)\}$. With this a new information, the prerequisite of D is true and the justification is consistent, then we can conclude that there are floatplanes that have wheels and land on wheels.

2 Embedded Computer

We use an embedded computer which is based on an ARM processor (Figure 1, embedded computer running Linux Debian). This microcomputer supports three serial protocols communications as SPI, I2C and UART, 40 digital input/output pins... Different sensors are connected to the microcomputer: a gyroscope which measures the rotations, an accelerometer that measures static and dynamic forces. Eventually, a magnetometer allows to estimate the direction by detecting the magnetic flux on earth. These three sensors can give all the information linked to a real cockpit. The embedded computer has an operating system (OS) based on Linux Debian. Running at 1 GHz single core CPU, 512 MB of RAM and an energy consumption of 0.8 Watts. Plus, `SWI-prolog version 7.7.18` (32 bits) was installed into it. An algorithm for calculating extensions of a default theory was implemented (Algorithm 1). This algorithm is coded in Prolog language and the compilation is done thanks to SWI-Prolog installed.

2.1 Routine of Computation

Facts and rules follow a particular syntax. This syntax represents a normal default $d = \frac{A:C}{C}$ with a weighting: $cl(\texttt{text}, \delta, A, C, \omega)$. Where `text` could be a comment describing the clause, δ could be a real fact "`hrd`" or default "`def`", A and C are the prerequisite and consequent respectively, and finally ω is a weighing as a priority. Real facts with no weighting are facts sampled at some moment and they are represented as following:

Figure 1: Embedded computer running Linux Debian

Algorithm 1 Calculation of extensions

Require: $E = \emptyset$
  procedure EXTENSION($E = \cup_{i=0}^{N} E_i$)
      Initialization
      Computation_of_Extension(E):
      while there is a default $\frac{A(X):C(X)}{C(X)}$ that has not yet been inspected **do**
         Select the default D
         Verify A(X) holds
         Verify C(X) is consistent with W
         Add C(X) to W
      end while
      Backtracking(deleting C(X) added to W)
      Computation_of_Extension(E)
  end procedure

```
cl(''text'',hrd,[], glider(airspeed_low),[]).
cl(''text'',hrd,[], glider(pitch_stable),[]).
cl(''text'',hrd,[], glider(roll_stable),[]).
cl(''text'',hrd,[], glider(altimeter_low),[]).
cl(''text'',hrd,[], glider(variometer_zero),[]).
```

These facts are describing the states of an airplane, in our case a glider, having no inclination nor rotation, low airspeed and low altitude, and no vertical speed. We can assume here that the glider has no motion because of the states of the airplane. Due to limited space to write, we will take the following notation; *g* for glider, *var_zero* for zero vertical speed, *pch_stb* for no inclination, *rll_stb* for no rotation, and *auth* for control tower authorization. In the same sense, defaults are represented as following:

```
cl('',def,[g(var_zero),g(pch_stb),g(rll_stb),auth],pilot(motor),[]).
```

Consider a default representing $\frac{(g(var\_zero) \wedge g(pch\_stb) \wedge g(rll\_stb) \wedge auth):pilot(motor)}{pilot(motor)}$, where the prerequisite is the information from the cockpit and authorization. In this example, if the context makes the default true and there are no contradictions with the conclusion, we jump to the conclusion: *pilot*(*motor*)[2]. This is

---

[2]Due to the lack of space, we do not describe decision making combined with the weighting in this example.

intuitively the way our Prolog program finds the solutions. Our system manages 113 rules which are as follows:

```
 1 p=0,0 hrd :  -> glider(airspeed_low)
 2 p=0,0 hrd :  -> glider(roll_stable)
 3 p=0,0 hrd :  -> glider(altimeter_low)
 4 p=0,0 hrd :  -> glider(variometer_low)
 5 p=0,0 hrd :  -> glider(land)
 6 p=0,0 def : glider(airspeed_zero), glider(altimeter_zero), -> glider(rest)
 7 p=0,0 def : glider(airspedd_zero), glider(-motor), -> glider(rest)
 ...
63 p=0,0 def : glider(landing), -> glider(rest_p)
64 p=0,0 def : glider(rest_p), -> pilot(yoke_p_n)
 ...
110 p=0,0 hrd : pilot(yoke_p_n), -> -pilot(yoke_pull)
111 p=0,0 hrd : pilot(yoke_pull), -> -pilot(yoke_p_n)
112 p=0,0 hrd : pilot(yoke_pull), -> -pilot(yoke_push)
113 p=0,0 hrd : pilot(yoke_push), -> -pilot(yoke_pull)
```

Figure 2: Our system with 113 rules of piloting.

2.2   Example and Results

Considering the facts G, we consult our Prolog program to know if in this context, we could take-off...

$$G : g(pitch\_stable), g(roll\_stable), g(motor\_off), g(low\_alt), g(low\_airspeed)$$

From Algorithm 1 programmed in Prolog, 5 different extensions are obtained. Table 1 summarizes the computed extensions with the defaults involved in each extension. The formalization of defaults are

Extensions	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}
E_0	✓	✓			✓	
E_1	✓	✓				✓
E_2	✓			✓		✓
E_3		✓	✓		✓	
E_4		✓	✓			✓

Table 1: Extensions and defaults calculated

shown in Table 2.

From G the best extension we can choose is E_3 because it has the good combinations of the actions $\{pilot(yoke\_pull), pilot(yoke\_roll\_neutral), pilot(motor)\}$ to reach the goal: **take-off**. Since the extension E_0 has: $\{pilot(yoke\_roll\_neutral), pilot(yoke\_pitch\_neutral), pilot(motor)\}$, the result is a straight flight that is not the goal, it would probably be a solution to be able to have more speed and later take-off. The extension E_1 has: $\{pilot(yoke\_roll\_neutral), pilot(yoke\_pitch\_neutral), \neg pilot(motor)\}$, there is no movement on the glider because the engine is off.

The E_2 extension has: $\{pilot(yoke\_roll\_neutral), pilot(yoke\_push), \neg pilot(motor)\}$, same result as the previous extension, no movement because the engine is off. And finally, the E_4 extension has: $\{pilot(yoke\_pull), pilot(yoke\_pitch\_neutral), \neg pilot(motor)\}$, same result as the previous extension, no

$$d_{16} = \frac{g(roll\_stable) : pilot(yoke\_roll\_neutral)}{pilot(yoke\_roll\_neutral)} \qquad d_{17} = \frac{g(pitch\_stable) : pilot(yoke\_pull)}{pilot(yoke\_pull)}$$

$$d_{18} = \frac{g(low\_alt) : pilot(yoke\_roll\_neutral)}{pilot(yoke\_roll\_neutral)} \qquad d_{19} = \frac{g(low\_alt) : pilot(yoke\_push)}{pilot(yoke\_push)}$$

$$d_{20} = \frac{g(low\_alt) : pilot(motor)}{pilot(motor)} \qquad d_{21} = \frac{g(low\_alt) : \neg pilot(motor)}{\neg pilot(motor)}$$

Table 2: Extensions calculated for this example.

movement because the engine is off. To study the case where there are the maximum number of solutions, we change the facts *G* and we obtain 13 extensions, Figure 3, with the same 113 rules. As additional information the computation of the extensions is taking 1.514 seconds. As the facts change,

```
EXTENSION : TRUE LITERALS : glider(airspeed_low), glider(altimeter_low),
    glider(descend), glider(final_approach), glider(final_approach_p),
    glider(land), glider(landing), glider(rest), glider(rest_p), glider(
    roll_stable), glider(variometer_down), pilot(yoke_p_n), pilot(yoke_r_n),
    pilot(-motor)
DEFAULTS USED : [66,65,64,57,33,31,30,27,9]
...
EXTENSION : TRUE LITERALS : glider(airspeed_low), glider(altimeter_low),
    glider(land), glider(landing), glider(roll_stable), glider(takeoff),
    glider(takeoff_p), glider(variometer_down), pilot(motor), pilot(yoke_p_n
    ), pilot(yoke_r_n)
DEFAULT USED : [70,69,68,36,31,13]
...
EXTENSION : TRUE LITERALS : glider(airspeed_low), glider(altimeter_low),
    glider(climb_p), glider(land), glider(landing), glider(landing_p),
    glider(roll_stable), glider(takeoff), glider(variometer_down), pilot(
    motor_2), pilot(yoke_pull), pilot(yoke_r_n)
DEFAULT USED : [90,72,71,62,37,31,13]
```

Figure 3: Computation of extensions: 113 instanced clauses, 5 elementary facts, and 13 extensions. Computation in 168,334 inferences, 1.478 CPU in 1.514 seconds (98% CPU, 113881 Lips).

the calculation time of the extensions, Table 3, will be variable. That is, if we have few facts, Prolog will make more logical inferences per second (Lips) to prove the consistency of the rules. However, if we have more facts, Prolog will make fewer inferences (Lips) to prove the rules.

Facts	Extensions	Instanced clauses	CPU	Lips
7	13	115	95%	114,131
5	13	113	98%	117,176
4	10	112	97%	130,098

Table 3: Comparative table on the results obtained from three different situations.

3 Conclusion

We successfully installed "`SWI-prolog version 7.7.18`" in an embedded computer. Also nonmonotonic reasoning for piloting an airplane was implemented. Our embedded system is composed of sensors such as gyroscope, accelerometer, magnetometer, pressure sensor,...Data sensors are transformed into normal defaults in order to respect a specific syntax. We tackled the problem of incomplete and uncertain information by formalizing the rules of piloting using default logic. We get good results in terms of calculation time, thanks to the use of Horn clauses and normal defaults (Considering the restriction of the embedded computer such as the low energy consumption (0.8 Watts), running at 1 GHz ARM11 (single core) and 512 Mb of RAM). Eventually, we described an example in which 5 extensions are obtained. The model implemented has 113 defaults. We are currently working on the decision making part in order to have a control system based on queries in Prolog. In addition we can take advantage of the infinite loops of Prolog which is one of his most outstanding tools. This allows to compute the extensions all the time while embedded computer is on.

References

[1] J. Delgrande & T. Schaub (2003): *On the relation between Reiter's default logic and its (major) variants. Seventh European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*, doi:10.1007%2F978-3-540-45062-7_37.

[2] J. Delgrande & T. Schaub (2005): *Expressing Default Logic Variants in Default Logic. Journal of Logic and Computation*, doi:10.1093/logcom/exi021.

[3] Michael Kifer & Yanhong Annie Liu (2018): Morgan & Claypool, doi:10.1145/3191315.

[4] Kurt Konolige (1988): *On the relation between default and autoepistemic logic.* In: *Artificial Intelligence*, 35, pp. 343–382, doi:10.1016/0004-3702(88)90021-5.

[5] J. McCarthy (1980): *Circumscription - A form of non-monotonic reasoning. Artificial intelligence* 13:1-2, doi:10.1016/0004-3702(80)90011-9.

[6] J. McCarthy (1986): *Applications of circumscription to formalizing common-sense knowledge. Artificial intelligence* 28-1, doi:10.1016/0004-3702(86)90032-9.

[7] Raymond Reiter (1980): *A logic for default reasoning. Artificial intelligence* 13:1-2, doi:10.1016/0004-3702(80)90014-4.

[8] Raymond Reiter (1981): *On closed world data bases.* In: *Readings in artificial intelligence*, Springer, pp. 119–140, doi:10.1007%2F978-1-4684-3384-5_3.

[9] Stuart J Russell & Peter Norvig (2016): Malaysia; Pearson Education Limited,, doi:10.1145/201977.201989.

[10] L. Sombé (1990): John Wiley & Sons Inc, doi:10.1002/int.4550050403.

[11] Leon Sterling (1990): MIT press, doi:10.1017/S026357470001599X.