The Dramatic True Story of the Frame Default

Vladimir Lifschitz
University of Texas at Austin

Abstract

This is an expository article about the solution to the frame problem proposed in 1980 by Raymond Reiter. For years, his "frame default" remained untested and suspect. But developments in some seemingly unrelated areas of computer science—logic programming and satisfiability solvers—eventually exonerated the frame default and turned it into a basis for important applications.

1 Introduction

This is an expository article about the dramatic story of the solution to the frame problem proposed in 1980 by Raymond Reiter [22]. For years, his "frame default" remained untested and suspect. But developments in some seemingly unrelated areas of computer science—logic programming and satisfiability solvers—eventually exonerated the frame default and turned it into a basis for important applications.

This paper grew out of the Great Moments in KR talk given at the 13th International Conference on Principles of Knowledge Representation and Reasoning. It does not attempt to provide a comprehensive history of research on the frame problem: the story of the frame default is only a small part of the big picture, and many deep and valuable ideas are not even mentioned here. The reader can learn more about work on the frame problem from the monographs [17, 21, 24, 25, 27].

2 What Is the Frame Problem?

The frame problem [16] is the problem of describing a dynamic domain without explicitly specifying which conditions are not affected by executing actions.

Here is a simple example. Initially, Alice is in the room, and Bob is not. Then Bob enters the room. Common sense tells us that both Alice and Bob will be in the room after that. But will this conclusion logically follow if we represent the given facts by formulas? The assumptions about the initial state of affairs and about the action that was executed can be written as

$$\mathit{In}_0(\mathit{Alice}),\ \neg \mathit{In}_0(\mathit{Bob}),\ \mathit{Enter}(\mathit{Bob}). \tag{1}$$

(The subscript 0 in the first two formulas indicates that we are talking about the time instant *before* the execution of the action.) The conclusions that we would like to derive are

$$\mathit{In}_1(\mathit{Alice}) \tag{2}$$

and

$$\mathit{In}_1(\mathit{Bob}). \tag{3}$$

These conclusions are obviously not entailed by formulas (1), which do not even contain the predicate symbol In_1. What is missing is the commonsense knowledge about the effect of entering a room:

$$\forall x(\mathit{Enter}(x) \rightarrow \mathit{In}_1(x)) \tag{4}$$

(any person who enters will be in the room after that).

Formula (3) logically follows from (1) and (4), but formula (2) does not. Assumptions (1) and (4) do not allow us to conclude that Alice's location has not changed. This is an instance of the frame problem.

Imagine now that we know, in addition to formulas (1), that Carol was not in the room initially: $\neg \mathit{In}_0(\mathit{Carol})$. The commonsense conclusion $\neg \mathit{In}_1(\mathit{Carol})$ is not a logical consequence of the assumptions that we made. This is an instance of the frame problem, too.

3 Default Theories

Reiter's approach to the frame problem is based on his concept of a default theory [22]. Recall that a first-order theory is characterized by the set of first-order sentences adopted as axioms. For instance, we can talk about the first-order theory with axioms (1) and (4). Default theories are more general: in addition to axioms, a default theory T can contain defaults—expressions of the form

$$\frac{F_1 \ \cdots \ F_m \ : \ \mathsf{M}\,G_1 \ \cdots \ \mathsf{M}\,G_n}{H}, \tag{5}$$

where the *premises* $F_1, \ldots, F_m$, the *justifications* $G_1, \ldots, G_n$ and the *conclusion* H are first-order formulas.[1]

Informally speaking, default (5) is a rule allowing us to derive H from $F_1, \ldots, F_m$ if the justifications $G_1, \ldots, G_n$ can be assumed without contradicting any of the

[1]The definition of a default in [22] requires $m = 1$ and $n > 0$. It is convenient for our purposes to allow m and n to be arbitrary nonnegative integers.

facts that can be derived from the axioms of T using classical logic and the defaults of T. This is a circular description, because it tells us under what conditions default (5) can be used to derive its conclusion by referring to the formulas that can be derived using the defaults of T, and (5) is one of these defaults. But Reiter found a way to make this informal idea precise. This was an important event both for logic and for artificial intelligence.

We will not reproduce Reiter's definition here in its entirety, because details are somewhat complicated. The idea is that defaults, like inference rules, allow us to extend the set of axioms by deriving new formulas, but the set of formulas that can be generated using defaults is defined by a fixpoint construction and is, generally, not unique. To determine whether a set E of sentences is an "extension" for a given default theory, we consider the set E' of sentences that can be derived from the axioms using classical logic and the inference rules

$$\frac{F_1 \ \cdots \ F_m}{H}$$

corresponding to the defaults (5) such that the justifications $G_1, \dots, G_m$ are consistent with E. This process is anti-monotone: if we make E larger then E' can only become smaller, because fewer defaults will "fire." We say that E is an extension for the given default theory if $E' = E$.

Here is an example. Consider a default theory with the axiom $p \wedge q$ and two defaults:

$$\frac{p \ : \ \mathsf{M}\, r}{r}, \qquad \frac{p \ : \ \mathsf{M}\, \neg r}{\neg r}. \tag{6}$$

Here p, q, r are propositional atoms. According to Reiter's semantics, this default theory has two extensions. One extension E_1 consists of all sentences that logically follow from $p \wedge q$ and r. Indeed, the first of the two given defaults "fires" relative to E_1 (its justification r is consistent with E_1); the second does not "fire"(its justification $\neg r$ is not consistent with E_1). The set of formulas that can be derived from the axiom $p \wedge q$ using classical logic and the inference rule

$$\frac{p}{r}$$

corresponding to the first default equals E_1. The other extension E_2 consists of all sentences that logically follow from $p \wedge q$ and $\neg r$.

A formula is *entailed* by a default theory if it belongs to each of its extensions. (This is sometimes called "skeptical" entailment.) In the example above, the formulas entailed by the theory are the logical consequences of the axiom $p \wedge q$. The two defaults "cancel" each other, as far as entailment is concerned.

4 The Frame Default

How can defaults help us solve the instances of the frame problem discussed in Section 2? We can try to make the first-order theory with axioms (1) and (4) stronger by adding the default

$$\frac{\mathit{In}_0(x)\ :\ \mathsf{M}\,\mathit{In}_1(x)}{\mathit{In}_1(x)}. \tag{7}$$

It says that if a person x is in the room before an action is executed, and it is consistent to assume that x stays in the room after the action, then that assumption is correct. It seems reasonable to expect that this default theory will entail (2).

To conclude that Carol's location was not affected when Bob entered the room, we can try to use the default dual to (7):

$$\frac{\neg\mathit{In}_0(x)\ :\ \mathsf{M}\,\neg\mathit{In}_1(x)}{\neg\mathit{In}_1(x)}. \tag{8}$$

We can expect that in application to *Carol* as x this default would allow us to derive $\neg\mathit{In}_1(\mathit{Carol})$ from $\neg\mathit{In}_0(\mathit{Carol})$. On the other hand, default (8) would not be applicable to *Bob* as x, because the justification $\neg\mathit{In}_1(\mathit{Bob})$ contradicts the consequence (3) of the axioms.

This is the idea of Reiter's approach to the frame problem. What is termed the frame default in his paper is actually the expression

$$\frac{R(x,s)\ :\ \mathsf{M}\,R(x,f(x,s))}{R(x,f(x,s))}. \tag{9}$$

It is based on the "situation calculus" model of change, more sophisticated than our use of subscripts 0 and 1 for two time instants. The premise $R(x,s)$ says that x has property R in situation s. The expression $f(x,s)$ in the justification and the conclusion represents the situation that results from applying action f to object x in situation s. In the "two time instants" notation, (9) would be written as

$$\frac{R_0(x)\ :\ \mathsf{M}\,R_1(x)}{R_1(x)},$$

which is a generalization of (7).

All examples of defaults so far were "normal": each of them has a single justification, which is identical to the default's conclusion. We will see in Section 8 that nonnormal defaults have important uses too.

To decide whether the frame default, or any other approach to the frame problem for that matter, produces satisfactory results, we need to use it in formalizations of specific dynamic domains and to investigate properties of these formalizations. This was not attempted in [22]. In 1980, when that paper was published, one could only say that the frame default looked promising.

5 Minimizing Abnormality

A few years later, John McCarthy [15] proposed an approach to the frame problem based on “abnormality theories”—first-order theories containing the special predicate Ab. A model of an abnormality theory is called minimal if the extent of Ab in it cannot be made smaller without violating the axioms.[2] Minimal models are models “with few abnormal objects.” The minimality condition can be expressed by a formula, called circumscription. This is a second-order formula: it uses a predicate variable to talk about changing the extent of Ab.

The formula

$$R_0(x) \wedge \neg Ab(x) \rightarrow R_1(x) \tag{10}$$

expresses that if x has property R at time 0 then normally x has property R at time 1 as well. McCarthy’s proposed solution to the frame problem referred to minimal models of theories containing axioms similar to (10). By minimizing abnormality, we would minimize the set of objects x for which the truth value of the property R would change with time from *true* to *false*.

This idea is used in [15] to describe the blocks world in which blocks can be moved and painted. A satisfactory solution to the frame problem would allow us to assert, first, that when a block is moved to a different location, the positions of all other blocks and the colors of all blocks remain the same; second, that when a block is painted a different color, the colors of all other blocks and the positions of all blocks remain the same.

McCarthy’s hope was that these assertions would be true in all minimal models of his abnormality theory. But this conjecture turned out to be incorrect: the theory has unintended minimal models that do not have the desired properties. The fact that some unintuitive models are not eliminated by minimizing change was proved in [9] using a counterexample that is much simpler than the blocks world domain. The example is known as the Yale Shooting Scenario, because it involves shooting, and its authors were affiliated with Yale University. In this scenario, a person named Fred is initially alive, and a gun is initially unloaded. Loading the gun, waiting for a while, and then shooting the gun at Fred can be expected to kill Fred. However, the abnormality theory describing this scenarion has an unintended minimal model, in which the gun becomes mysteriously unloaded and Fred survives.

Speaking of the frame problem in general, the authors of the counterexample expressed the view that they established more than the inadequacy of the circumscription approach: the example could be restated, they noted, in terms of default theories. Their reformulation did not use Reiter’s frame default, however.

[2] “Smaller” is understood here in the sense of set inclusion, not in the sense of comparing cardinalities. To make the concept of a minimal model precise, we need to specify whether the extents of predicates other than Ab may be changed as we try to make the extent of Ab smaller; we do not go here into discussing this issue.

Instead, they wrote axioms involving an abnormality predicate in the style of [15], but replaced the use of circumscription with a default. In the notation of [22], the minimality of Ab can be expressed by the default

$$\frac{:\ \mathsf{M}\,\neg Ab(x)}{\neg Ab(x)} \tag{11}$$

(if it is consistent to assume that x is not abnormal then that assumption is correct). So the impossibility of using the frame default (9) was not really established by their argument. Nevertheless, the general feeling after the publication of the Yale Shooting example was that the potential of default theories as a tool for solving the frame problem was very much in question.

6 Explanation Closure

The Yale Shooting paper prompted a large number of responses. Most of them proposed alternative ways to address the frame problem using some general methods of reasoning about defaults and exceptions—for instance, using circumscription, but in a different manner.

One group of authors took another path [8, 26, 23]. In many cases, axioms describing the effects of actions have a special logical form. In the "two time instants" notation, the effects of an action are often described by implications similar to (4): the consequent describes the effects of the action using predicates with the subscript 1, and the antecedent describes the action (and possibly its preconditions) using predicates with the subscript 0. Assuming this special logical form of effect axioms, we may be able to generate, by some syntactic procedure, the additional assumptions needed for solving the frame problem.

If, for instance, the only effect axiom is (4) then the additional axioms may look like this:

$$\forall x(\neg In_0(x) \land In_1(x) \to Enter(x)),$$
$$\forall x \neg(In_0(x) \land \neg In_1(x)).$$

The former expresses the idea that if the truth value of *In* has changed from *false* to *true* then there is only one possible explanation: the *Enter* action was executed. The latter says that, in the domain under consideration, the truth value of *In* cannot change from *true* to *false*: there is no action with such an effect.

Additional axioms of this kind are known as "explanation closure" axioms.

7 The Frame Default in 1987

Nineteen eighty-seven was a bad year for the frame default. No attempt had been made by that time to use the frame default for describing any specific domain. Its close relative—the idea of minimizing change—had been tested, and found inadequate. There was no software at that time that would help researchers to investigate properties of the frame default, because reasoning in default theories had not been automated. And work on explanation closure seemed to suggest that the complicated theory of defaults and extensions might not be even needed for solving the frame problem.

On the other hand, 1987 was the year when the relationship between default logic and the programming language Prolog was discovered. This event, discussed in the next section, had a profound effect on future work on the frame default.

8 Negation as Failure

A Prolog program consists of rules. For instance, each of the two lines of the program

```
p(a).
p(c) :- p(a), \+ p(b).
```

is a rule. The first rule is a "fact." The second consists of the head and the body, separated by the symbol `:-` (read "if" and resembling the shape of the left arrow). The body consists of two "subgoals" `p(a)` and `\+ p(b)`. The latter includes the "negation as failure" symbol `\+` (resembling the shape of the logical symbol $\not\vdash$).

Query evaluation in the presence of negation as failure is accomplished by the process called SLDNF resolution [13, Chapter 3]. Consider, for instance, the operation of Prolog on the program above and the query

```
?- p(X).
```

(In Prolog, capitalized identifiers are variables.) To find the ground instances of the goal `p(X)` that can be justified by the rules of the program, Prolog first matches ("unifies") `p(X)` with the fact `p(a)` and produces the answer `X = a`. Then it matches `p(X)` with the head of the second rule, `p(c)`, and attempts to evaluate the two subgoals in the body of the rule. The first subgoal `p(a)` succeeds, because it is a fact included in the program as one of its rules. To decide whether the second subgoal `\+ p(b)` succeeds, Prolog checks whether the expression following the negation symbol `\+` , taken as a goal, would fail; this is how SLDNF resolution treats subgoals containing negation as failure. This goal, `p(b)`, indeed fails, because it matches neither the fact `p(a)` nor the head `p(c)` of the second rule. Consequently the goal `p(c)` succeeds, and Prolog produces a second answer, `X = c`.

The key observation made in the groundbreaking paper [2] is that Prolog rules with negation as failure are similar to Reiter's defaults. Facts are axioms. Other rules can be rewritten as defaults in the following way. The head of a rule becomes the conclusion. The members of the body that do not contain negation as failure become the premises. Each member of the body that does contain negation as failure is turned into a negative literal by replacing `\+` with the logical negation symbol, and this literal becomes a justification.

For instance, the Prolog program shown at the beginning of this section corresponds to the default theory that has the axiom $p(a)$ and one default:

$$\frac{p(a)\ :\ \mathsf{M}\,\neg p(b)}{p(c)}. \tag{12}$$

This default theory has a single extension, which consists of all sentences that logically follow from $p(a)$ and $p(c)$. These two atoms are entailed by the default theory, and the atom $p(b)$ is not entailed. We see that Reiter's semantics matches in this case the result of evaluating the query `?- p(X).`

9 Prolog Is an Implementation of Default Logic

The correspondence between Prolog programs and default theories discussed in the previous section shows that Prolog can be viewed as a partial implementation of default logic. Let T be a default theory corresponding to a Prolog program, and let A be a ground atom. Consider what happens when we execute the program on the goal A. If A succeeds then T entails A, that is to say, A belongs to each of its extensions.[3] If A fails then T does not entail A (and, moreover, A does not belong to any of its extensions). For instance, we can determine that the default theory with the axiom $p(a)$ and the default (12) entails $p(a)$ and $p(c)$, but does not entail $p(b)$, by running the Prolog program shown at the beginning of Section 8 on these three queries. And it is possible also that the process of evaluating A will not terminate.

Our assertion that by 1987 reasoning in default logic had not been automated (Section 7) was not enirely true, as we can now see. Paradoxically, partial implementations of default logic—Prolog systems that can handle negation as failure—were already available at the time of publication of the default logic paper [22]. But it did not occur to anyone at that time that Prolog could be used in this way.

10 Answer Set Semantics

The relationship between logic programming and default logic became even closer when the answer set semantics was defined [6]. An *(extended) logic program* consists of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n, \tag{13}$$

[3]We disregard here the problems related to the occurs check and to floundering [13, Sections 4, 15].

where each L_i is a literal, positive (an atom A) or negative ($\neg A$). This syntax allows a rule to contain two kinds of negation: negation as failure *not*, familiar from traditional Prolog, and "strong" (or "classical") negation $\neg$, found within a negative literal. Rule (13) has the same meaning as the default

$$\frac{L_1 \cdots L_m \; : \; \mathsf{M}\,\overline{L_{m+1}} \cdots \mathsf{M}\,\overline{L_n}}{L_0} \tag{14}$$

(by $\overline{L}$ we denote the literal complementary to L). To be more precise, answer sets of a logic program can be described as extensions for the corresponding set of defaults (14) intersected with the set of literals. Thus the answer set semantics allows us to talk about syntactically simple defaults (such that the premises, the justifications, and the conclusion are literals) and syntactically simple elements of extensions (literals) using the syntax of logic programs with two kinds of negation.[4]

For example, the Prolog rules from Section 8 can be viewed as rules (13) in which all literals L_i are positive:

$$\begin{array}{l} p(a), \\ p(c) \leftarrow p(a), \mathit{not}\ p(b). \end{array}$$

This program has one answer set, $\{p(a), p(c)\}$. In the syntax of logic programming, defaults (6) become

$$\begin{array}{rl} r & \leftarrow p, \mathit{not}\ \neg r, \\ \neg r & \leftarrow p, \mathit{not}\ r. \end{array} \tag{15}$$

The frame defaults (7) and (8) become

$$\begin{array}{rl} \mathit{In}_1(x) & \leftarrow \mathit{In}_0(x), \mathit{not}\ \neg \mathit{In}_1(x), \\ \neg \mathit{In}_1(x) & \leftarrow \neg \mathit{In}_0(x), \mathit{not}\ \mathit{In}_1(x), \end{array} \tag{16}$$

and the "minimization default" (11) turns into

$$\neg Ab(x) \leftarrow \mathit{not}\ Ab(x). \tag{17}$$

Rules (15)–(17) contain strong negation and are different in this sense from Prolog rules.

11 The Role of Literals

Moving from Reiter's syntax of defaults (5) to the syntax of logic programs with two negations (13) has prompted a change in the methodology of representing knowledge by default theories. Syntactically simple defaults and axioms became

[4]The definition of an answer set in [6] does not refer to defaults. Instead, it adapts Reiter's fixpoint construction to the simpler framework of logic programs.

important. A default cannot be written in logic programming notation unless its premises, justifications, and conclusion are literals. If an axiom is an atom A then it is essentially identical to the default without premises and justifications that has A as its conclusion; such an axiom can be treated as a fact. If an axiom is a conjunction of atoms then it can be thought of as a set of facts. For instance, the default theory from Section 3 can be written as a logic program consisting of facts p, q and rules (15). But if an axiom is a disjunction or an implication then it is not clear how to represent it by rules of the form (13).

For instance, axiom (4), which expresses that any persons who enters the room will be in the room, contains an implication. If our intention is to restrict ourselves to default theories corresponding to logic programs then this idea has to be expressed in a different way. Axiom (4) can be replaced with an inference rule—a default without justifications [28]:

$$\frac{\mathit{Enter}(x)}{\mathit{In}_1(x)}. \tag{18}$$

Instead of the default theory described in Section 3 we will use then the logic program consisting of facts (1), rules (16), and the rule

$$\mathit{In}_1(x) \leftarrow \mathit{Enter}(x) \tag{19}$$

that represents default (18).

12 Using Prolog to Reason with the Frame Default

Once the default theory from Section 3 is replaced with the logic program (1), (16), (19), it becomes possible to use Prolog for the automation of reasoning about this action domain. Since the program contains strong negation, it cannot be processed by Prolog as is. But consider the result of replacing $\neg \mathit{In}_0$, $\neg \mathit{In}_1$ in this program with new predicate symbols Out_0, Out_1:

$$\begin{array}{rl} \mathit{In}_0(\mathit{Alice}), & \\ \mathit{Out}_0(\mathit{Bob}), & \\ \mathit{Enter}(\mathit{Bob}), & \\ \mathit{In}_1(x) & \leftarrow \mathit{In}_0(x), \mathit{not}\ \mathit{Out}_1(x), \\ \mathit{Out}_1(x) & \leftarrow \mathit{Out}_0(x), \mathit{not}\ \mathit{In}_1(x), \\ \mathit{In}_1(x) & \leftarrow \mathit{Enter}(x). \end{array} \tag{20}$$

The result of eliminating strong negation from a program P in favor of additional predicate symbols, as in this example, is a program whose answer sets are closely related to the answer sets of P [6, Proposition 2]. In particular, the answer set of program (20) is the answer set of program (1), (16), (19) with all negative literals

$$\neg \mathit{In}_0(\cdots),\ \neg \mathit{In}_1(\cdots)$$

replaced by the corresponding atoms

$$Out_0(\cdots),\ Out_1(\cdots).$$

Program (20), rewritten in accordance with the syntactic conventions of Prolog, looks like this:

```
in0(alice).  out0(bob).  enter(bob).
in1(X) :- enter(X).
in1(X) :- in0(X), \+ out1(X).
out1(X) :- out0(X), \+ in1(X).
```

Given this program, in response to the query

```
?- in1(X).
```

Prolog will calculate the set of individuals who are going to be in the room after the execution of the action:

```
X = bob ;
X = alice.
```

13 Answer Set Solvers

Since Prolog query evaluation may not terminate, the possibilities of Prolog as a mechanism for anwering queries about answer sets are limited even in the absence of strong negation. For instance, the program

$$\begin{array}{l} p \leftarrow not\ q, \\ q \leftarrow not\ p \end{array} \tag{21}$$

has two answer sets, $\{p\}$ and $\{q\}$. But we cannot get any information about its answer sets by running Prolog; an attempt to evaluate a query would produce an error message, such as

```
Out of local stack.
```

Fortunately, it turned out that the computational ideas used in the design of fast satisfiability solvers [7] can be applied also to the problem of generating answer sets of a logic program. Software systems based on this idea, such as SMODELS [19], DLV [11], and CLINGO [4], are called answer set solvers. Like Prolog systems, they provide partial implementations of default logic, and in some ways they give us more power.

Issues involved in designing answer set solvers are related to two main differnces between logic programs and sets of propositional clauses. First, logic programs are nonmonotonic: they include negation as failure, which corresponds, as we have seen, to justifications in default theories. Second, logic programs may use variables. The operation of an answer set solver begins with a series of substitutions that replace variables with judiciously chosen ground terms, intermixed with equivalent transformations that make the program smaller; this is called "intelligent instantiation."

Unlike Prolog, an answer set solver does not expect a query from the user. It can take a program as input and produce all its answer sets. For intance, given the input

```
p :- not q.
q :- not p.
```

CLINGO produces the list of all answer sets of program (21):

```
Answer: 1
p
Answer: 2
q
```

Answer set solvers can process programs with strong negation. For instance, the default theory given as an example in Section 3 can be encoded in their input language as follows:

```
p.     q.
r :- p, not -r.
-r :- p, not r.
```

Given this input, CLINGO returns two sets of literals:

```
Answer: 1
p q r
Answer: 2
p q -r
```

With an answer set solver available, we do not have to eliminate strong negation from program (1), (16), (19), as we did in Section 12. Given the input

```
in0(alice).  -in0(bob).  enter(bob).
in1(X) :- enter(X).
in1(X) :- in0(X), not -in1(X).
-in1(X) :- -in0(X), not in1(X).
```

CLINGO replies:

```
Answer: 1
in0(alice) -in0(bob) enter(bob) in1(bob) in1(alice)
```

14 Planning as Generating Answer Sets

In Sections 12 and 13 we saw that Prolog systems and answer set solvers allow us to automate reasoning about a dynamic domain described using the frame default. The example there involved a single action. Computational problems that involve a sequence of actions, such as temporal projection (deciding how the world will change after executing a given sequence of actions) and planning (generating a sequence of actions that is guaranteed to achieve a given goal), can be approached in a similar way [3]. But the simplistic "two time instants" notation introduced in Section 2 has to be replaced in this case by a more sophisticated model of time.

We can think, for instance, of a discrete sequence of instants, and write

$\mathit{Next}(t, t_1)$	for "t_1 is the time instant that follows t",
$\mathit{Holds}(\mathit{In}(x), t)$	for "person x is in the room at time t",
$\mathit{Occurs}(\mathit{Enter}(x), t)$	for "person x enters the room during the interval between t and the following time instant".

Then rule (19) will turn into

$$\mathit{Holds}(\mathit{In}(x), t_1) \leftarrow \mathit{Next}(t, t_1), \mathit{Occurs}(\mathit{Enter}(x), t),$$

and the first of the frame defaults (16) will become

$$\mathit{Holds}(\mathit{In}(x), t_1) \leftarrow \mathit{Next}(t, t_1), \mathit{Holds}(\mathit{In}(x), t), \mathit{not}\ \neg \mathit{Holds}(\mathit{In}(x), t_1). \tag{22}$$

The use of logic programs of this kind for planning is somewhat similar to the "planning as satisfiability" method [10]. That approach requires that the domain be described in propositional logic, which can be achieved, for instance, by grounding a representation formed using the process of explanation closure (Section 6). When the domain is described by a logic program, the frame default can be used instead of explanation closure, and an answer set solver takes the place of a satisfiability solver. Variables are eliminated from the program by the intelligent instantiation procedure included in the solver. We will say more about advantages of answer set planning in the next section.

15 The RCS Advisor

The planning method described above became the basis of the logic program called the RCS Advisor [20]. The RCS, or Reaction Control System, was the system aboard the Space Shuttle designed to maneuver it while it was in space. The RCS Advisor was used to verify the possibility of doing that even if several elements of the system malfunction.

Here is one of the rules from that program:

```
    h(value(W,X),T1) :-  next(T,T1), signal(X), is_wire(W),
                         h(value(W,X),T), not nh(value(W,X),T1).
```

This is a version of the frame default similar to rule (22). The main difference between the two rules is that the "toy world" condition $In(x)$ is replaced here by the "real life" condition `value(W,X)`.

Why didn't the creators of the RCS Advisor use the explanation closure solution to the frame problem and a satisfiability solver, instead of the frame default and an answer set solver?

There was a good reason for that. As discussed in Section 6, the process of explanation closure is applicable when the axioms describing the effects of actions have certain special logical form. The RCS was a complicated device, and the effects of actions, such as flipping a switch, had to be described in two steps. First, the simple direct effect was stated: when you flip the switch, the state of the switch changes. Then the other effects would logically follow using the rules describing the RCS that were included in the program. Such indirect, two-level descriptions of actions become possible when the frame default is used, but extending the explanation closure method to actions with indirect effects is not straighforward.

16 Conclusion

As discussed above, in 1987 the future of the frame default was dim. But by 2001 the frame default had been fully exonerated, efficiently implemented, and used in an important project.

The use of the answer set semantics and answer set solvers for knowledge representation and search is known as answer set programming [14, 18]. Reasoning about actions is only one of many areas to which it has been successfully applied [1, 4, 5, 12].

17 Acknowledgements

Thanks to Yuliya Lierler and to the anonymous referees for comments on a draft of this paper.

References

[1] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[2] Nicole Bidoit and Christine Froidevaux. Minimalism subsumes default logic and circumscription in stratified logic programming. In *Proceedings LICS-87*, pages 89–97, 1987.

[3] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in non-monotonic logic programs. In Sam Steel and Rachid Alami, editors, *Proceedings of European Conference on Planning*, pages 169–181. Springer, 1997.

[4] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[5] Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2008.

[6] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[7] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 89–134. Elsevier, 2008.

[8] Andrew Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence, Proceedings 1987 Workshop*, 1987.

[9] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[10] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[11] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[12] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1594–1597. MIT Press, 2008.

[13] John Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. Second, extended edition.

[14] Victor Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[15] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.

[16] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[17] Erik Mueller. *Commonsense reasoning.* Elsevier, 2006.

[18] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[19] Ilkka Niemelä and Patrik Simons. Smodels—an implementation of the stable model and well-founded semanics for normal logic programs. In *Proceedings 4th Int'l Conference on Logic Programming and Nonmonotonic Reasoning (Lecture Notes in Artificial Intelligence 1265)*, pages 420–429. Springer, 1997.

[20] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the Space Shuttle. In *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 169–183, 2001.

[21] Zenon W. Pylyshyn. *Robot's Dilemma: The Frame Problem in Artificial Intelligence.* Greenwood Publishing Group Inc., Westport, CT, USA, 1987.

[22] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[23] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[24] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[25] Erik Sandewall. *Features and Fluents*, volume 1. Oxford University Press, 1994.

[26] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyburg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, 1990.

[27] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[28] Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.