Reasoning about Plans*

Witold Łukaszewicz and **Ewa Madalińska-Bugaj**

Institute of Informatics, Warsaw University
02-097 Warszawa, ul. Banacha 2, POLAND
email: {witlu,ewama}@mimuw.edu.pl

Abstract

In classical planning we are faced with the following formal task: Given a set A of permissible actions, a description α of initial states and a description β of final states, determine a plan Π, i.e. a finite sequence of actions from A, such that execution of Π begun in any state satisfying α is guaranteed to terminate in a state satisfying β.

In this paper we extend the classical model of planning by admitting plans that are not assured to succeed. We address two basic problems connected with such plans: (1) How to determine whether a given plan is *valid* (i.e. always succeeds), *admissible* (i.e. may succeed or fail) or *inadmissible* (i.e. never succeeds). (2) Given an admissible plan, determine a minimal set of observations that are to be made in the initial state (or in some intermediate state, if the plan is in progress) to validate or falsify the plan.

1 Introduction

In classical planning we are faced with the following formal task: Given a set A of admissible actions, a formula α describing the set of initial states and a formula β representing a goal to be achieved, construct a plan Π, i.e. a finite sequence of actions from A, such that execution of Π begun in any state satisfying α is guaranteed to terminate in a state satisfying β.

This classical model of planning is oversimplified. In many practical settings it is reasonable to construct and execute plans that are not assured to succeed. There are two basic reasons for that. Firstly, it may happen that a plan that always achieves a final state does not exist. Secondly, even if such a plan exists, it may be better to choose a simpler, although uncertain plan. An example will help to illustrate this.

Suppose I want to contact John. All information I have is his phone number and the fact that he is at home. Given this, the only plan I can choose is to call him. Of course, the plan may fail, if John's phone line is busy, but there is no better possibility. To guarantee the success of the plan, I have to know that John's line is not busy at the moment. Unfortunately, this information is hardly available.

Assume now that, in addition, I know John's address. In this case, I can assure my goal by visiting him. However, it may still be much more reasonable to give up this ironclad plan and to choose the previous one.

In this paper, we extend the classical model of planning by permitting plans that are not assured to succeed. Such a plan is just a sequence of actions, considered relatively to some *specification*. A specification consists of a description of the initial states and a goal to achieve.

Plans that need not behave according to their specifications can be naturally divided into three categories: (1) Those that always achieve their goals (*valid plans*). (2) Those that may achieve their goals or not, depending on some additional information (*admissible plans*).[1] (3) Those that never achieve their goals (*inadmissible plans*).

As an example, suppose that all we know about the initial state is that a turkey is alive and the goal is to make it dead. There are two actions: *load* (loads a gun) and *shoot* (kills the turkey, provided that the gun is loaded). Consider three plans: (1) *load*; *shoot*; (2) *shoot*; (3) *load*. The first of these plans is valid, the second one is admissible, whereas the third one is inadmissible.

It is important to note that an admissible plan can be often validated (i.e. made valid) or falsified (i.e. made inadmissible) by providing new observations. Reconsider the plan (2) stated above. The observation that the gun is initially loaded (resp. unloaded) validates (resp. falsifies) the plan.

This paper addresses two problems:

(1) How to determine whether a given plan is valid, admissible or inadmissible.

(2) Given an admissible plan, determine a minimal set of observations that are to be made in the initial

*This research was partially supported by the KBN grant 8T11C 001 11.

[1] It should be emphasized that admissible plans differ from what is called *uncertain plans* in the AI literature. This latter notion corresponds to plans that may fail not because some information is missing, but rather because they involve actions that succeed with some probability. (See [Boutilier *et al.*, 1995], for a good survey concerning uncertain plans.)

state (or in some intermediate state, if the plan is in progress) to validate or falsify the plan.

To represent actions occurring in plans, we use Dijkstra's approach originally developed to deal with programs [Dijkstra, 1976; Dijkstra, Scholten, 1990]. The advantage of Dijkstra's formalism for reasoning about action and change, when compared with purely logical approaches such as Situation Calculus [McCarthy, Hayes, 1969; Lifschitz, 1988; Gelfond *et al.*, 1991] or Features and Fluents [Sandewall, 1994], is its simplicity. It has been shown in [Łukaszewicz, Madalińska, 1994; 1995; 1995a; Jabłonowski *et al.*, 1996].

The paper is organized as follows. We start with a brief summary of Dijkstra's semantics for a very simple programming language. Section 3 is devoted to the theory of prime implicants that play an important role in plan analysis. In section 4, we show how action languages are to be formalized using Dijkstra's methodology. In section 5, we provide a number of results allowing to analyse plans before their executions, whereas, in section 6, these results are generalized for plans in progress. Finally, in section 7, we provide conclude remarks and future work.

Proofs of all stated results can be found in the full version of this paper.

2 Introduction to Dijkstra's semantics

In [Dijkstra, Scholten, 1990] we are provided with a very simple programming language whose semantics is specified in terms of formula transformers. More specifically, with each command S there are associated two formula transformers, called the *weakest precondition* and the *strongest postcondition*, denoted by wp and sp, respectively. Before providing the meaning of these transformers we introduce some terminology.[2]

First of all, we assume here that the programming language under consideration contains one type of variables only, namely Boolean variables. This assumption may seem overly restrictive, but as a matter of fact no other variables will be needed for our purpose. In the rest of this paper Boolean variables will be referred to as *fluents*.

Let $\mathcal{F}$ be a set of fluents. A *state over* $\mathcal{F}$ is any function σ from the members of $\mathcal{F}$ into the truth-values $\{0,1\}$. A state σ is said to be a *model* of a formula α iff α is true in σ.

An *assertion language* over a set $\mathcal{F}$ of fluents is the set of all formulae constructable in the usual way from members of $\mathcal{F}$, sentential connectives $(\neg, \Rightarrow, \wedge, \vee, \equiv)$ and truth-constants T (true) and F (false). In what follows, the term 'formula' refers always to a formula of some fixed assertion language. If β and α are formulae and x is a fluent, then we write $\beta[x \leftarrow \alpha]$ to denote the formula which obtains from β by replacing all occurrences of x by α. If x is a fluent and α is a formula, then we write $\exists x.\alpha$ as an abbreviation for $\alpha[x \leftarrow T] \vee \alpha[x \leftarrow F]$.

[2]We ignore the *weakest liberal precondition* transformer, considered in [Dijkstra, Scholten, 1990], because it will not be used in the sequel.

The formula transformers mentioned above are to be understood as follows. For each command S and each formula α:

- $wp(S,\alpha)$ is the formula whose models are precisely all states such that execution of S begun in any one of them is guaranteed to terminate in a state satisfying α.
- $sp(S,\alpha)$ is the formula whose models are precisely all states such that each of them can be reached by starting execution of S in some state satisfying α.

For a detailed discussion of Dijkstra's methodology the reader should consult [Apt, Olderog, 1991].

2.1 List of commands

The considered language consists of *skip* command, *assignment* to simple variables, *alternative* command and *sequential composition* of commands[3]. Semantics of these commands is specified in terms of formula transformers explained above.

1. **The *skip* command.** This is the "empty" command in that its execution does not change the computation state. The semantics of *skip* is thus given by

   $$wp(skip,\alpha) = sp(skip,\alpha) = \alpha.$$

2. **The *assignment* command.** This command is of the form $x := e$, where x is a fluent and e is a (propositional) formula. The effect of the command is to replace the value of x by the value of e. Its semantics is given by

   $$wp(x := e,\ \alpha) = \alpha[x \leftarrow e].$$

   $$sp(x := e, \alpha) = \exists y.((x \equiv e[x \leftarrow y]) \wedge \alpha[x \leftarrow y]). \quad (1)$$

   If the fluent x does not occur in the expression e, the equation (1) can be simplified. In this case

   $$sp(x := e, \alpha) = (x \equiv e) \wedge \exists x.\alpha. \quad (2)$$

   In the sequel we shall often deal with assignment commands, $x := e$, where e is T or F. In this case the equation (2) can be replaced by

   $$sp(x := e, \alpha) = \begin{cases} x \wedge \exists x.\alpha & \text{if } e \text{ is } T \\ \neg x \wedge \exists x.\alpha & \text{if } e \text{ is } F \end{cases} \quad (3)$$

3. **The *sequential composition* command.** This command is of the form $S_1; S_2$, where S_1 and S_2 are any commands. It is executed by first executing S_1 and then executing S_2. Its semantics is given by

   $$\begin{aligned} wp(S_1; S_2, \alpha) &= wp(S_1, wp(S_2, \alpha)). \\ sp(S_1; S_2, \alpha) &= sp(S_2, sp(S_1, \alpha)). \end{aligned}$$

4. **The *alternative* command.** This command is of the form

   $$\textbf{if}\quad B_1 \to S_1 \quad [\!] \quad \cdots \quad [\!] \quad B_n \to S_n \quad \textbf{fi} \quad (4)$$

   where $B_1, \ldots, B_n$ are formulae and $S_1, \ldots, S_n$ are commands. $B_1, \ldots, B_n$ are called *guards* and expressions of the form $B_i \to S_i$ are called *guarded*

[3]The original Dijkstra's language contains *abort* command and *iterative* commands as well, but they are not needed for our purpose.

commands. In the sequel, we refer to the general command (4) as IF. The command is executed as follows. If none of the guards is true, then the execution aborts. Otherwise, one guarded command $B_i \rightarrow S_i$ with true B_i is *randomly* selected and S_i is executed.[4] The semantics of IF is given by

$$wp(\text{IF}, \alpha) = \bigvee_{i=1}^{n} B_i \wedge \bigwedge_{i=1}^{n} (B_i \Rightarrow wp(S_i, \alpha)).$$

$$sp(\text{IF}, \alpha) = \bigvee_{i=1}^{n} (sp(S_i, B_i \wedge \alpha)).$$

3 Prime implicants

In this section we provide a brief introduction to the theory of prime implicants that will play the crucial role in the rest of this paper. Our presentation is partially based on [Brown, 1990].

We start with some preliminary terminology.

A *literal* is a fluent or its negation. A *term* is either T or F or a conjunction of literals in which no fluent appears more than once. A formula is said to be in *disjunctive normal form* (*DNF*, for short) if it is a disjunction of different terms.[5] It is well-known that each formula can be constructively transformed into its equivalent in DNF. We say that a term t_1 *absorbs* a term t_2 if either t_1 is T or t_2 is F or t_1 is a subterm of t_2. For instance, the term a absorbs the term $a \wedge l$. Let α be a formula in DNF. We write $ABS(\alpha)$ to denote the formula obtained from α by deleting all absorbed terms. Clearly, α and $ABS(\alpha)$ are equivalent.

Let t be a term different from F and suppose that α is any formula. We say that t is an *implicant of* α iff the formula $t \Rightarrow \alpha$ is a tautology. An implicant t of α is said to be *prime* iff no proper subterm of t is an implicant of α.

The problem of finding all prime implicants of a given (propositional) formula has been extensively studied in the Switching Circuits Theory.[6] Actually, there exist a number of algorithms solving this task. One of them, usually referred to as *iterated consensus*, is given below.

Two terms are said to have an *opposition* if one of them contains the fluent f and the other the fluent $\neg f$. For instance, the terms $\neg a \wedge l$ and $a \wedge d$ have a single opposition, in the fluent a.

Suppose that two terms, t_1 and t_2, have exactly one opposition. Then the *consensus* of t_1 and t_2, written $c(t_1, t_2)$, is the term obtained from the conjunction $t_1 \wedge t_2$ by deleting the opposed fluents as well as any repeated fluents. For example, $c(\neg a \wedge l, a \wedge d)$ is $l \wedge d$.

Let α be a formula. The *Blake canonical form of* α, written $BCF(\alpha)$, is the formula obtained from α by the following construction.

(1) Replace α by its disjunctive normal form. Denote the resulting formula by β.

(2) Repeat as long as possible:
if β contains a pair t_1 and t_2 of terms whose consensus exists and no term of β is a subformula of $c(t_1, t_2)$, then $\beta := \beta \vee c(t_1, t_2)$.

(3) Take $ABS(\beta)$. This is $BCF(\alpha)$.

Theorem 1 ([Brown, 1990]) Formulae α and $BCF(\alpha)$ are equivalent. ■

Theorem 2 ([Brown, 1990]) $BCF(\alpha)$ is the disjunction of all prime implicants of α. ■

Example 1 Let α be $(\neg a \wedge d) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg c \wedge \neg d)$. Since α is in disjunctive normal form, $\beta = \alpha$. After performing step (2), we get

$$(\neg a \wedge d) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg c \wedge \neg d) \vee (b \wedge d \wedge \neg c). \quad (5)$$

Since $ABS((5)) = (5)$, the formula (5) is the Blake canonical form of α. ■

Classically, a prime implicant of a formula α is a minimal satisfiable term logically implying α. In what follows, we shall be interested in finding minimal satisfiable terms logically implying a given formula α, under the assumption that a given formula β holds. This motivates the following definition.

Definition 1 Let α and β be formulae. A term t is an implicant of α wrt β iff (1) $\beta \wedge t$ is satisfiable; (2) $\beta \wedge t \Rightarrow \alpha$ is a tautology. A term t is said to be a prime implicant of α wrt β iff t is an implicant of α wrt β and no proper subterm of t is an implicant of α wrt β. ■

The following theorem holds.

Theorem 3 A term t is a prime implicant of α wrt β iff t is a prime implicant of $\beta \Rightarrow \alpha$ such that $\beta \wedge t$ is satisfiable. ■

The application of Theorem 3 requires satifiability check. This check can be effectively done if β is in DNF. The details follow.

Let t be a term different from F and let $\beta = t_1 \vee \cdots \vee t_n$ be a formula in DNF. The *quotient of* β *wrt* t, written β/t, is the formula (also in DNF) obtained from β by the following construction: (1) Replace each t_i containing a literal whose negation occurs in t by F. (2) In the remaining terms delete all fluents occurring in t (if all fluents are deleted, replace the term by T). For instance, if t is $\neg p \wedge q$ and β is $\neg q \wedge r \vee \neg p \wedge s$, then β/t is $F \vee s$.

The following result is straightforward.

Theorem 4 Let t be a term different from F and suppose that β is any formula in DNF. The formula $t \wedge \beta$ is satisfiable iff β/t contains at least one term different from F. ■

4 Action languages and plans

Specification of an action language is a three-step process.

(1) The first step is to choose an underlying assertion language L serving to represent the considered world. In this paper, L is always the classical propositional logic based on a finite set of fluents.

[4] Note that when more than one guard is true, the selection of a guarded command is nondeterministic.

[5] In the logical literature DNF is often defined as a disjunction of terms where a term is understood as either T or F or any conjunction of literals. Note, however, that we can always restrict ourselves to terms in which no fluent appears more than once: each repeated occurence of a fluent l can be removed from a term, whereas any term including l and $\neg l$ can be replaced by F.

[6] It should be stressed that this problem is NP-complete.

(2) The next step is to provide action symbols representing actions. For instance, we can have the action symbol *load*, representing the action that makes a gun loaded.

(3) The final step is to provide action symbols with Dijkstra-style semantics. This is done by first translating these symbols into commands of programming language described in section 2, and then calculating the weakest precondition and the strongest postcondition for the chosen actions. The action *load* can be naturally translated into the assignment command $l := T$, where l is the fluent standing for *loaded*. The semantics of *load* is given by

$$wp(load, \alpha) = \alpha[l \leftarrow T]; \quad sp(load, \alpha) = l \wedge \exists l.\alpha.$$

The objects we are primarily interested in are *plans*. These are expressions of the form $A_1; \ldots; A_n$, where $A_1, \ldots, A_n$ are action symbols. Each plan is always considered wrt a specification, i.e. a pair $\langle \alpha, \beta \rangle$, where α and β are satifiable formulae[7]. The plan $A_1; \ldots; A_n$, considered wrt a specification $\langle \alpha, \beta \rangle$, has the following interpretation: the actions $A_1, \ldots, A_n$ are to be sequentially performed to achieve a goal β, provided that the initial state satisfies α. As remarked earlier, we never assume that the execution of the plan activated in a state satisfying α guarantees β.

5 Analysis of plans

Let $\Pi = A_1; \ldots; A_n$ be a plan. We say that Π is *valid* wrt a specification $\langle \alpha, \beta \rangle$ iff Π terminates in a state satisfying β whenever it is activated in a state satisfying α. Π is said to be *inadmissible* wrt $\langle \alpha, \beta \rangle$ iff Π terminates in a state satisfying $\neg\beta$ whenever it is activated in a state satisfying α. Finally, Π is called *admissible* wrt $\langle \alpha, \beta \rangle$ iff it is neither valid wrt $\langle \alpha, \beta \rangle$ nor inadmissible wrt $\langle \alpha, \beta \rangle$.

Formally, a plan $\Pi = A_1; \ldots; A_n$ is a program in Dijkstra's language. Accordingly, $wp(A_1; \ldots; A_n, \beta)$ is the formula whose models are precisely all states σ such that whenever Π is activated in σ, it is guaranted to terminate in a state satisfying the goal β. On the other hand, if $wp(A_1; \ldots; A_n, \neg\beta)$ holds in the initial state, then the plan Π never achieves the goal β. Thus we have:

Theorem 5 A plan $\Pi = A_1; \ldots; A_n$ is valid (resp. inadmissible) wrt $\langle \alpha, \beta \rangle$ iff the formula $\alpha \Rightarrow wp(A_1; \ldots; A_n, \beta)$ (resp. $\alpha \Rightarrow wp(A_1; \ldots; A_n, \neg\beta)$) is a tautology. Otherwise, i.e. if neither $\alpha \Rightarrow wp(A_1; \ldots; A_n, \beta)$ nor $\alpha \Rightarrow wp(A_1; \ldots; A_n, \neg\beta)$ is a tautology, the plan Π is admissible wrt $\langle \alpha, \beta \rangle$. ■

As we remarked earlier, admissible plans can be often validated or falsified by providing new information. The details are these.

Let $\Pi = A_1; \ldots; A_n$ be a plan admissible wrt $\langle \alpha, \beta \rangle$. A term $t = l_1 \wedge \ldots \wedge l_k$ $(k > 1)$ such that $\alpha \wedge t$ is satisfiable is called a *support* (resp. *counter-support*) *for* Π wrt $\langle \alpha, \beta \rangle$ iff Π is valid (resp. inadmissible) wrt $\langle \alpha \wedge t, \beta \rangle$. A support (resp. counter-support) t for Π is said to be *minimal* iff no proper subterm of t is a support (resp. counter-support) for Π. Intuitively, a minimal support t for a plan can be viewed as a minimal set of additional observations (literals of t) whose truth in the initial state validates the plan. Similarly, a minimal counter-support for the plan can be viewed as a minimal set of additional observations whose truth in the initial state falsifies the plan.

The next theorem provides a method of finding all minimal supports and counter-supports for a given plan.

Theorem 6 Let $\Pi = A_1; \ldots; A_n$ be a plan admissible wrt $\langle \alpha, \beta \rangle$. A term t is a minimal support (resp. counter-support) for Π iff t is a prime implicant of $wp(A_1; \ldots; A_n, \beta)$ (resp. $wp(A_1; \ldots; A_n, \neg\beta)$ wrt α.

■

Example 2 We have four fluents a, l, h, d standing for *alive* (a turkey), *loaded* (a gun), *hidden* (a turkey) and *deaf* (a turkey), respectively. There are two actions, *load* and *shoot*, specified as follows.

load: **if** $d \rightarrow l := T \;[\!]\; \neg d \rightarrow l := T; h := T$ **fi**

shoot: **if** $l \wedge \neg h \rightarrow l := F; a := F \;[\!]\; \neg l \vee h \rightarrow l := F$ **fi**.

Consider the plan $\Pi = load; shoot$, regarded wrt the specification $\langle a \wedge \neg h, \neg a \rangle$. It is easily verified that

$$wp(load; shoot, \neg a) \equiv \neg h \wedge d \vee \neg a$$

$$wp(load; shoot, a) \equiv a \wedge (\neg d \vee h).$$

Since neither $a \wedge \neg h \Rightarrow wp(load; shoot, \neg a)$ nor $a \wedge \neg h \Rightarrow wp(load; shoot, a)$ is a tautology, we infer that the plan Π is admissible wrt $\langle a \wedge \neg h, \neg a \rangle$.

To find all minimal supports for Π wrt $\langle a \wedge \neg h, \neg a \rangle$, we calculate all prime implicants of $wp(load; shoot, \neg a)$ wrt $a \wedge \neg h$. By Theorem 3, these are the prime implicants of $a \wedge \neg h \Rightarrow wp(load; shoot, \neg a)$ which are consistent with $a \wedge \neg h$. Applying the algorithm stated in section 3, it is readily verified that the BCF of $a \wedge \neg h \Rightarrow wp(load; shoot, \neg a)$ is $\neg a \vee h \vee d$. Since $\neg a$ and h are both inconsistent with $a \wedge \neg h$, we conclude that there is one prime implicant of $wp(load; shoot, \neg a)$ wrt $a \wedge \neg h$, namely d. Accordingly, d is the only minimal support for Π wrt $\langle a \wedge \neg h, \neg a \rangle$.

We leave it to the reader to check that the term $\neg d$ is the only minimal counter-support for Π wrt $\langle a \wedge \neg h, \neg a \rangle$.

■

The following theorem provides a partial relationship between supports and counter-supports.

Theorem 7 Let $\Pi = A_1; \ldots; A_n$ be a plan admissible wrt $\langle \alpha, \beta \rangle$. Suppose further that the commands corresponding to $A_1, \ldots, A_n$ are all deterministic. If all the minimal supports for Π wrt $\langle \alpha, \beta \rangle$ are false in the initial state, then at least one counter-support for Π wrt $\langle \alpha, \beta \rangle$ is true in this state. ■

Theorem 7 allows us to falsify a deterministic plan when all its supports are false. However, the theorem does not generally hold for non-deterministic plans.

6 Plans in progress

In the previous section we have provided a number of results allowing to analyse a plan before its execution.

[7] The assumption that α and β are satisfiable is not necessary. However, specifications violating this assumption are of no practical interest.

Sometimes, however, it is reasonable to provide a similar analysis when a plan is in progress. To illustrate this, reconsider the plan from Example 2. To validate or falsify this plan, we have to establish whether the turkey is deaf or not in the initial state. Unfortunately, this observation is hardly available. On the other hand, we can start the plan and try to figure out whether the turkey is hidden after performing the action *load*. If so, we know that the plan will fail and should be given up. Otherwise, the plan will succeed. Clearly, it is much easier to determine whether the turkey is hidden or not than to determine whether it is deaf or not.

A *plan in progress* is any expression of the form

$$A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n, \tag{6}$$

where $A_1, \ldots, A_k, A_{k+1}, \ldots, A_n$ are action symbols and $1 \leq k < n$. Such a plan, considered wrt a specification $\langle \alpha, \beta \rangle$, has the following intuitive interpretation: the actions $A_1, \ldots, A_k$ have been already performed, starting in a state satisfying α, and now the actions $A_{k+1}, \ldots, A_n$ are to be performed to achieve a state satisfying β.

The only difference between a plan in progress of the form (6) and the ordinary plan $A_1; \ldots; A_k; A_{k+1}; \ldots; A_n$ is that the former has been partially executed. This motivates the following definition.

Let $\Pi = A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n$ be a plan in progress. Π is said to be *valid* (resp. *admissible, inadmissible*) wrt a specification $\langle \alpha, \beta \rangle$ iff the (ordinary) plan $A_1; \ldots; A_k; A_{k+1}; \ldots; A_n$ is valid (resp. admissible, inadmissible) wrt $\langle \alpha, \beta \rangle$.

Let $\Pi = A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n$ be a plan in progress, considered wrt $\langle \alpha, \beta \rangle$. Obviously, Π can be reduced to the plan $A_{k+1}; \ldots; A_n$, considered wrt $\langle \gamma, \beta \rangle$, where γ is a formula characterizing all states reachable after performing $A_1; \ldots; A_k$ begun in a state satisfying α. By the definition of *sp* (the strongest postcondition) transformer, γ is just $sp(A_1; \ldots; A_k, \alpha)$. Thus, we immediately have:

Theorem 8 Let $\Pi = A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n$ be a plan in progress. Π is valid (resp. admissible, inadmissible) wrt $\langle \alpha, \beta \rangle$ iff the plan $A_{k+1}; \ldots; A_n$ is valid (resp. admissible, inadmissible) wrt $\langle sp(A_1; \ldots; A_k, \alpha), \beta \rangle$. ■

Let $\Pi = A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n$ be a plan in progress, admissible wrt $\langle sp(A_1; \ldots; A_k, \alpha), \beta \rangle$. A *minimal support* (resp. *counter-support*) for Π is any minimal support (resp. counter-support) for $A_{k+1}; \ldots; A_n$ wrt $\langle sp(A_1; \ldots; A_k, \alpha), \beta \rangle$. Intuitively, a minimal support (resp. counter-support) for a plan $A_1; \ldots; A_k \parallel A_{k+1}; \ldots; A_n$ is a minimal set of observations whose truth in the state being the result of performing the actions $A_1; \ldots; A_k$ validates (resp. falsifies) the plan.

Example 2 (continued) To validate or falsify the plan Π, we have to establish the value of the fluent d in the initial state. Suppose that this information is unavailable. All we can do in this case is to start the plan by performing the action *load*, i.e. to replace Π by a plan in progress $\Pi_1 = load \parallel shoot$. Since Π is admissible wrt $\langle a \wedge \neg h, \neg a \rangle$, Π_1 is also admissible wrt $\langle a \wedge \neg h, \neg a \rangle$. To find minimal supports for Π_1 wrt $\langle a \wedge \neg h, \neg a \rangle$, we calculate all minimal supports for the plan *shoot* wrt $\langle sp(load, a \wedge \neg h), \neg a \rangle$. In view of Theorem 6, these are prime implicants of $wp(shoot, \neg a)$ wrt $sp(load, a \wedge \neg h)$. It is readily verified that

$$sp(load, a \wedge \neg h) = d \wedge l \wedge a \wedge \neg h \vee \neg d \wedge l \wedge h \wedge a. \tag{7}$$

$$wp(shoot, \neg a) = l \wedge \neg h \vee \neg a. \tag{8}$$

Performing the algorithm from section 3, we get $BCF((7) \Rightarrow (8)) \equiv \neg a \vee \neg l \vee \neg h \vee d$. It is easy to check that implicants $\neg h$ and d are consistent with (7), whereas $\neg a$ and $\neg l$ are not. Thus, Π_1 has two minimal supports wrt $\langle a \wedge \neg h, \neg a \rangle$, namely d and $\neg h$. In other words, if we observe that the turkey is either deaf or not hidden, after performing the action *load*, we can safetely continue the plan to achieve its goal.

To find all minimal counter-supports for Π_1 wrt $\langle a \wedge \neg h, \neg a \rangle$, we calculate prime implicants of $wp(shoot, a)$ wrt $sp(load, a \wedge \neg h)$. It is easily checked that these are h and $\neg d$. Accordingly, if we observe that the turkey is hidden or not deaf, after executing the action *load*, we know that our plan will fail and hence should be given up. ■

The next example illustrates an interesting phenomenon: there are plans that can be validated or falsified only when they are in progress.

Example 3 There are two fluents, a (alive) and l (loaded), and two actions *spin* and *shoot*1, defined by the commands

spin: **if** $T \rightarrow l := T \,[\!]\, T \rightarrow l := F$ **fi**

*shoot*1: **if** $l \rightarrow a := F; l := F \,[\!]\, \neg l \rightarrow skip$ **fi**.

Consider the plan $\Pi = spin; shoot1$, regarded wrt the specification $\langle a, \neg a \rangle$.

$$wp(spin; shoot1, \neg a) \equiv \neg a. \tag{9}$$

$$wp(spin; shoot1, a) \equiv F. \tag{10}$$

Since neither $a \Rightarrow (9)$ nor $a \Rightarrow (10)$ is a tautology, we conclude that Π is admissible wrt $\langle a, \neg a \rangle$.

$$BCF(a \Rightarrow (9)) \equiv \neg a.$$

$$BCF(a \Rightarrow (10)) \equiv \neg a.$$

Since $\neg a$ is inconsistent with a, there are neither minimal supports nor minimal counter-supports for Π wrt $\langle a, \neg a \rangle$. This intuitively means that no additional observation in the initial state can validate or falsify the plan Π.

To validate/falsify the plan Π, we have to consider it in progress. Assume therefore, that we started the plan by performing the action *spin*. Denote the resulting plan, $spin \parallel shoot1$, by Π_1. Since Π is admissible wrt $\langle a, \neg a \rangle$, Π_1 is also admissible wrt $\langle a, \neg a \rangle$.

$$sp(spin, a) = a. \tag{11}$$

$$wp(shoot1, \neg a) = l \vee \neg a. \tag{12}$$

$$BCF((11) \Rightarrow (12)) \equiv \neg a \vee l.$$

Since $\neg a$ is inconsistent with a, whereas l is consistent with a, we conclude that there is one minimal support for Π_1 wrt $\langle a, \neg a \rangle$, namely l. Accordingly, if we observe that the gun is loaded after performing the action *spin*, we know that the plan will succeed.

$$wp(shoot1, a) \equiv \neg l \wedge a. \tag{13}$$

$$BCF((11) \Rightarrow (13)) \equiv \neg a \vee \neg l.$$

Since $\neg a$ is inconsistent with a, whereas $\neg l$ is consistent with a, we infer that there is one minimal counter-support for Π_1 wrt $\langle a, \neg a \rangle$, namely $\neg l$. Thus, if we observe that the gun is unloaded after performing the action *spin*, we know that the plan will fail and should be given up. ■

7 Conclusions

In this paper, we have argued that it makes sense to consider and execute plans that are not guaranteed to succeed. We have adressed two fundamental problems related to such plans:

(1) How to determine whether a given plan is valid (i.e. always succeeds), admissible (i.e. succeeds or fails depending on some additional information) or inadmissible (i.e. always fails).

(2) Given an admissible plan, determine a minimal set of observations that are to be made in the initial state (or in some intermediate state, if the plan is in progress) to validate or falsify the plan.

To formalize actions occurring in plans, we have employed Dijkstra's semantics for programming languages. This allows us to represent a broad class of plans, in particular those including actions with non-deterministic effects. In addition, we do not require that initial or final states are to be completely specified.

We believe that technical results stated in sections 5 and 6 can be used while constructing plans. We would like to pursue this topic in the future.

Ackowledgements

We would like to thank Władysław M. Turski for his comments on the earlier draft of this paper.

References

[Apt, Olderog, 1991] K. Apt, E.Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, 1991.

[Boutilier *et al.*, 1995] C. Boutilier, T. Dean, S. Hanks. Planning under Uncertainty: Structural Assumptions and Computational Leverage. In *Proc. 3rd European Workshop on Planning (EWSP-95)*, 1995.

[Brown, 1990] F. M. Brown. *Boolean Reasoning.* Kluwer Academic Publishers, 1990.

[Dijkstra, 1976] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[Dijkstra, Scholten, 1990] E. W. Dijkstra, C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.

[Gelfond *et al.*, 1991] M. Gelfond, V. Lifschitz, A. Rabinov. What Are the Limitations of Situation Calculus? In *Proc. AAAI Symposium of Logical Formalization of Commonsense Reasoning*, Stanford, 1991, 55-69.

[Jabłonowski *et al.*, 1996] J. Jabłonowski, W. Łukaszewicz, E. Madalińska-Bugaj. Reasoning about Action and Change: Defeasible Observations and Actions with Abnormal Effects. In *Proc. of 20th German Conference on Artificial Intelligence*, Springer-Verlag, Lecture Notes on Artificial Intelligence, **1137**, p.135-148.

[Lifschitz, 1988] V. Lifschitz. Formal Theories of Action. In *Readings in Nonmonotonic Reasoning*, M. Ginsberg (ed.), Morgan Kaufmann Publishers, Palo Alto, 1988, 35-57.

[Łukaszewicz, Madalińska, 1994] W. Łukaszewicz, E. Madalińska-Bugaj. Program Verification Techniques as a Tool for Reasoning about Action and Change. In *Proc. of 18th German Conference on Artificial Intelligence*, Springer-Verlag, Lecture Notes in Artificial Intelligence, **861**, 226-236, 1994.

[Łukaszewicz, Madalińska, 1995] W. Łukaszewicz, E. Madalińska-Bugaj. Reasoning about Action and Change Using Dijkstra's Semantics for Programming Languages: Preliminary Report. In *Proc. IJCAI-95*, Montreal, Canada, 1950-1955, 1995.

[Łukaszewicz, Madalińska, 1995a] W. Łukaszewicz, E. Madalińska-Bugaj. Reasoning about Action and Change: Actions with Abnormal Effects. In *Proc. of 19th German Conference on Artificial Intelligence*, Springer-Verlag, Lecture Notes in Artificial Intelligence, **981**, 209-220, 1995.

[McCarthy, Hayes, 1969] J. McCarthy, P.J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie (eds.), *Machine Intelligence* **4**, 1969, 463-502.

[Sandewall, 1994] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems.* Oxford Logic Guides, **30**, Oxford Science Publications, 1994.