Fluent-free Action Representation with IK-STRIPS Planning Formalism

Michal Čertický*

Supervisor: Ján Šefránek†

Department of Applied Informatics, Comenius University, Mlynská Dolina 842 48 Bratislava

Abstract: Widespread approach to representation of planning problems is to use the concept of domain-specific fluents to describe all of its time and action dependant components. By introducing novel fluent-free planning formalism IK-STRIPS we have solved potential problems resulting from occasional need to separate the definition of actions from other parts of the planning problem. Despite being inspired by obsolete STRIPS formalism from 1970s, it can be effectively used in complex domains with incomplete knowledge using stable model semantics of answer set programming. After the formal definition of IK-STRIPS planning language and its semantics, we have not only proven that its expressive power is greater than that of its predecessors, but also presented an experimental implementation of a planner using it. Comparison of our IK-STRIPS planner to its commonly used alternative DLV^K has shown that the average computation time of increasingly complex planning tasks is significantly lower using the IK-STRIPS planner, making it more suitable for use in complex real-world domains.

Keywords: planning language, knowledge representation, answer set programming, planner

1 Introduction

In this paper, we will introduce a novel logic-based planning and action representation formalism called IK-STRIPS and present the implementation of a fast planner using it. We will emphasize its differences to classical STRIPS representation (which is unusable for planning in real-world domains with incomplete knowledge) and newer, commonly used alternatives, such as $\mathcal{K}$ language.

1.1 Background

IK-STRIPS formalism and planner aim to be used by autonomous intelligent agents existing in complex real-world domains. Most real world domains differ from the micro-worlds traditionally used in A.I. in that they have an incomplete factual database which changes over time [16]. To deal with such incomplete knowledge about our domain, we have chosen the stable model semantics of Answer Set Programming (ASP) [7][12]. Subrahmanian and Zaniolo [20] came with this idea of reducing a planning problem to the problem of satisfying an answer set (also called "stable model") of a logic program already in 1995. According to Vladimir Lifschitz [11], "the advantage of this *answer set programming* approach to planning is, that the representation of properties of actions is easier when logic programs are used instead of axiomatizations in classical logic, in view of nonmonotonic negation as failure". This choice of answer set semantics over classical logic allows us to use both explicit negation and nonmonotonic negation as failure, thus enhancing the expressive power available for dealing with incompleteness of knowledge at our disposal.

There are currently several alternative planning formalisms and planners, such as commonly used $\mathcal{K}$ language and its planner DLV^K [3], that are capable of successful reasoning with incomplete knowledge. In their case, all the actions are defined using the concept of so called *"fluents"*.

1.2 Problem

Fluents in typical planning languages describe all the time and action dependant predicates [14]. It is important to understand, that those fluents are inseparable part of the definition of one certain planning problem. In other words, domain-specific fluents are used to describe all the dynamics in a given planning domain, including preconditions and effects of actions, thus making action representation dependant on current planning problem (figure 1). This fact can in some cases be considered a slight drawback.

There are situations, where we would like our actions to be separated from other kinds of knowledge about the planning domain. We have encountered

*certicky@fmph.uniba.sk

†sefranek@ii.fmph.uniba.sk

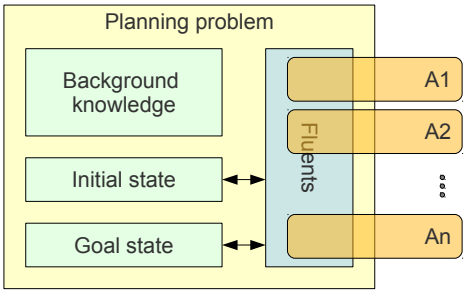

Figure 1: Representation of actions using the concept of fluents in case of $\mathcal{K}$ language.

such situations during our experiments with learning agents, when we were trying to create new actions by composing sequences of already known ones without modifying the rest of agents knowledge base.

Another problematic situation arises, when we want to add completely new actions on the run, which are supposed to affect the parts of the world represented by non-fluent predicates. In typical case, we don't have the possibility and/or time to revise the whole knowledge base each time an agent learns a new action.

1.3 Solution

To solve the problems mentioned above, we needed some kind of planning formalism with representation of actions separated from the rest of the problem without using the concept of domain-specific fluents. Example of such formalism was STRIPS [5][1] introduced in 1971, and its later modifications. In STRIPS-based formalisms, action operators didn't operate on states of some transition system [6] (like in case of fluent-based formalisms), but directly affected world models. What is more important is, that action representation in STRIPS-based formalisms (figure 2) was completely separated, domain-independent, and encapsulated by default.

As its name suggests, our IK-STRIPS formalism drew inspiration from classical STRIPS. Naturally, numerous modifications and improvements were needed in order for it to be usable with incomplete knowledge and answer set semantics.

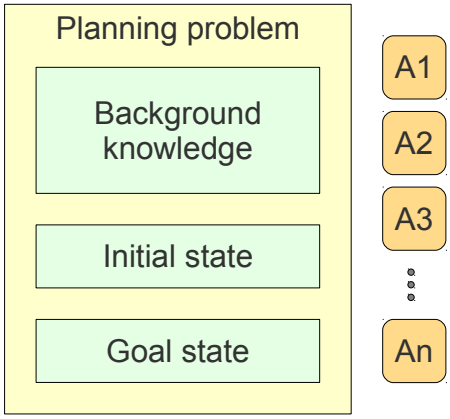

Figure 2: Representation of actions in case of IK-STRIPS and other STRIPS-based formalisms.

1.4 Additional motivation

First piece of additional motivation for introducing a novel formalism (and for reading the rest of this paper) is of practical nature. Implementation of most commonly used planner DLV^K is well fit for theoretical experiments, or even practical planning on limited simplistic domains, but its usability with agents existing in complex environments is questionable for following two reasons. DLV^K always needs to have expected plan length submitted by user and can only produce plans of that given length. Naturally, artificial agents have no way of knowing the length of the plan leading to their goal. IK-STRIPS planner (section 5) always tries to find the shortest possible plan, using optimized A^* algorithm. Second problem of DLV^K planner is its poor performance when computing longer plans, which is often a necessity. In this article we will show, that IK-STRIPS planner is significantly faster than DLV^K when computing plans of length $l > 10$.

Naturally, improving classical STRIPS with answer set semantics is not enough for truly intuitive and easy to understand action representation. Our goal was to get closer to a way humans understand actions. After all, action languages are, according to Gelfond and Lifschitz [6], supposed to be a formal models of parts of the natural language that are used for talking about the effects of actions. In order to achieve intuitiveness and similarity to natural language, each IK-STRIPS action has a set of effects, which are applied (or not applied) depending on cur-

rent world model. This can also lower the number of actions an agent needs, thus also lowering the computational complexity of planning tasks. As an example let's consider a hunter agent with action $shoot(D)$, where variable D denotes direction. Such action is commonly understood to be applicable if an agent has some ammunition, and it is meant to have two effects: a) ammunition is decreased by 1, and b) if there is something in direction D, it will be hit. However, such conditional model-dependant effect cannot be represented in classical STRIPS. We would need to have two actions $shoot\_if\_direction\_free(D)$ and $shoot\_if\_direction\_obstructed(D)$.

1.5 Outline

In the rest of the article, after some preliminaries concerning answer set semantics and classical STRIPS, we will give formal specification of IK-STRIPS formalism by defining *actions* with their *effect descriptors* and *effect instances*, *goals* and *plans*, and describing the process of world model modification. After that we will show that expressive power of IK-STRIPS is greater than that of classical STRIPS representation, and present the performance results of our IK-STRIPS planner implementation in comparison with DLV^K. We will also provide short description of used heuristics and ideas for alternative approaches, and finally outline a direction for possible further improvement and development.

2 Preliminaries

2.1 Answer Set Semantics

IK-STRIPS formalism is based on a paradigm of logic programming, and all the knowledge used in the planning process is represented by *Extended Logic Programs* (*ELP*) without function symbols. ELP is basically any finite set of rules of a form

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

where a_i, b_j and c are literals, i.e., either propositional atoms, or such atoms preceded by explicit negation sign "¬". The "not" symbol denotes negation by failure (default negation) [2]. Part of the rule before "←" sign is called *head* and part after it is *body*. Rule with empty body ($n = m = 0$) is called a *fact* and rule without head is an *integrity constraint*. Facts can be written in abbreviated form without the implication symbol ("←"). If r is a rule of the form described above, then c is denoted by $head(r)$ and $\{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_n\}$ by $body(r)$. We will also denote the default negation free part of $body(r)$ consisting of a set $\{a_1 \ldots a_n\}$ by $body^+(r)$.

As we can see, extended logic programs provide two kinds of negation operators. Explicit negation of an atom a ($\neg a$) is used, when we have explicit information (e.g. observation) of given atom a being false. Default negation of an atom a (*not a*) means that we are unable to resolve a from our current knowledge. In other words it means, that we don't know about a being true.

Representing knowledge with two kinds of negation (explicit and default negation) is useful for reasoning with incomplete knowledge, since it creates more expressive truth value system - see figure 3.

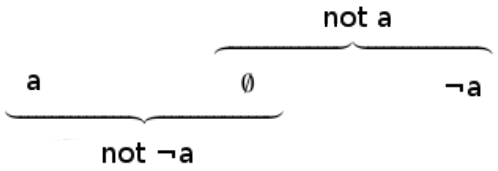

Figure 3: Truth values in ELP.

Remark 1. For easier understanding of the meaning of truth values in ELP, imagine yourself waiting at the railway crossing, and take a look at figure 3. Let's say, that an atom a means that there is a train approaching. Then a means, that we see the train, *not a* means that we don't know about it being there (perhaps we didn't even look), and $\neg a$ means that we are sure that there is no train (we looked and saw that there isn't one). Now, that we understand the difference between those two negation operators, we can see that if we want to safely cross the railway, we need to have an information $\neg a$ saying, that we are sure that there is no train coming.

Understanding of several ASP-related terms is required before we can introduce the concept of answer sets and explain the semantics of ASP. In the rest of this subsection, we will explain those terms, finishing with the concept of inference through *answer set following* ($\models_{AS}$).

First of all, *Extended herbrand base* is a set of all literals of a form $p(t_1, \ldots, t_n)$ or $\neg p(t_1, \ldots, t_n)$, where p is n-ary predicate symbol and each t_i is a constant. Such variable-free literals are called *ground literals*.

Extended herbrand base of a program P is denoted $H(P)$.

A *herbrand interpretation* of a logic program P is any coherent (not containing literals a and $\neg a$ together) subset $I \subseteq H(P)$ of its extended herbrand base. We say an interpretation $I \in S$ is *minimal* in the set of interpretations S if there does not exist an interpretation $J \in S$ such that $J \subset I$.

A *herbrand model* of a logic program P is a herbrand interpretation of P such that it satisfies all rules in P. We say that interpretation I *satisfies* a rule r iff:

$$body^{+}(r) \subseteq I \Rightarrow head(r) \in I$$

Interpretation *satisfies* an extended logic program P, iff it satisfies each rule $r \in P$. We can also say, that rule r or extended logic program P is satisfied in interpretation I.

For any interpretation I there exists a *program reduct* P^I which is obtained from P by deleting

- each rule r such that $body^{not}(r) \cap I \neq \emptyset$
- each default literal *not* a such that $a \notin I$,

where $body^{not}(r)$ denotes a set of all literals b, such that *not* $b \in body(r)$.

Each interpretation I of extended logic program P is then an *answer set* if and only if I is a minimal model of program reduct P^I. We will denote a set of all answer sets of program P as $AS(P)$.

Now, that we have a definition of answer sets, we can finally explain what AS-following means. Actually, there are two possible types of semantics for AS-following: *cautious* and *brave* following.

In case of cautious following, we say that a literal l *AS-follows* from a program P ($P \models_{AS} l$) iff $\forall S \in AS(P), l \in S$. A set of literals X then cautiously AS-follows from a program P, iff each literal $l \in X$ cautiously AS-follows from P. A rule r cautiously AS-follows from P iff $\forall S \in AS(P) : r\ is\ satisfied\ in\ S$. If U is a set of rules then $P \models_{AS} U$ iff $\forall r \in U\ P \models_{AS} r$.

The difference between brave and cautious following is, that for a literal l to bravely AS-follow from program P, it is sufficient to be included only in one of programs answer sets: $\exists S \in AS(P), l \in S$. For AS-following of sets of literals, rules, and sets of rules, the difference is similar. A set of literals X bravely AS-follows from a program P, iff each literal $l \in X$ bravely AS-follows from P. A rule r bravely AS-follows from P iff $\exists S \in AS(P) : r\ is\ satisfied\ in\ S$. If U is a set of rules then $P \models_{AS} U$ iff $\forall r \in U\ P \models_{AS} r$.

Choice between cautious and brave reasoning is optional and should be made in accordance with certain application. Potential advantage of brave reasoning is, that when deciding the AS-following of a certain literal we may (in some cases) not need to generate all the answer sets of a given ELP. Knowledge obtained by cautious reasoning however tends to model real situation with higher precision. Implementation of planner introduced in this article uses cautious reasoning by default.

2.2 Classical STRIPS

STRIPS originally stands for Stanford Research Institute Problem Solver [5], which was a pioneer planning program developed by Fikes and Nilsson around 1970. Even though authors called it a *problem solver*, in terminology of modern AI, a term *planner* is more precise [17][1] (since the planning is not the only problem-solving technique). For the purposes of this article, the program itself is however not important. Term STRIPS is nowadays used to denote an *action representation language* developed originally for the program of the same name. Due to its power of representation, the language developed for describing STRIPS' actions was since then used, with minor modifications, in a large number of traditional (but more advanced planner implementations) [1].

First of all, it is important to understand, that STRIPS-like planning is an algorithmic process manipulating so called *world models*. World models are simple sets of literals of a given logical language, describing current state of the world.

Each action in STRIPS is represented on two levels: *action operator* and *action routine*. Abstract *operators* have their corresponding *routines* - procedures whose execution causes actual changes in the world. Significant difference is, that operators are applied to world models during planning, while execution of routines actually affects the world, for example by effectors of some robotic agent. Planning can therefore be understood as an attempt to find a sequence of operators whose associated routines will lead to a desired state of the world. Just like routines have their procedural arguments, operators have their own corresponding parameters.

Language of routines used for representation of those actions can be any procedural programming language. Choice of this language is not important and can be different for each practical application.

Abstract and environment/hardware-independent operators however, are significant and basic part of STRIPS formalism.

Representation of operators in classical STRIPS language consists of three major components:

1. *Name* of operator and its *parameters*,
2. *preconditions*, and
3. *effects*.

First component consists merely of name of the operator and its parameters (arity ≥ 0). Those parameters are variables which are in the process of planning substituted by certain constants. Therefore when deciding the applicability of an operator, or computing the world model after its application, all the variables from preconditions or effects that correspond to variables in parameters are replaced by the same constants as parameters before doing anything else. We can see, that one operator can be applicable on world model with one set of constants as parameters, and inapplicable with different one. Also one action can modify the same world model differently, depending on its parameters.

Second component represents preconditions of operators applicability on given world model. In original 1971 STRIPS [5] they were expressed by a set of well-formed logical formulas. For example: $(\exists x,u)[AT(u,x) \wedge ATR(X)]$.
In later variants of STRIPS language, they were however expressed more simply by a conjunction or set of first-order logic atoms, which say what must be true (included in a world model) before the operator can be applied [17].

Third component of STRIPS operator represents its effects on the world model. They were also originally expressed by a set of well-formed formulas, but later planner variants use only first-order logic atoms. Those atoms are either added to a given world model, or they are removed from it. Distinction between those two groups is achieved by dividing effect-describing atoms into two sets: *add list* and *delete list*.

In the following section, we will introduce and define an alternative planning and action representation formalism IK-STRIPS, which is based on STRIPS language.

3 IK-STRIPS Formalism

Now, that we have defined all the required terminology related to extended logic programs and answer-set semantics, and a description of classical STRIPS planning formalism, we may proceed to definition of actions, goals and plans in our IK-STRIPS formalism.

3.1 Actions

Representation of actions will be, as mentioned before, based on the classical representation language STRIPS. Some modifications are needed to ensure usability along with ELP-based knowledge representation, and for reasoning with incomplete knowledge.

IK-STRIPS and classical STRIPS both define its actions as a double (R,O) consisting of a routine R and abstract operator O. However, since the form of actions routine depends on a certain practical application and is separated from the rest of the planning problem, we will ignore the routines and focus solely on the operators.

Operators of IK-STRIPS have, similarly to their predecessors, their set of parameters. Those parameters are variables that are substituted by constants during the planning process, along with all their occurences in the rest of operator description. We can call all those variables, that occur in the set of parameters, a *parametric variables*. From now on, we will talk about applicability and effects of an action/operator, but we should bear in mind, that we are always talking about an operator with all its parametric variables substituted by certain constants in advance. The difference here is, as we will see, that unlike in classical STRIPS, there can also be some non-parametric variables in operator description.

Formal definition of IK-STRIPS operator (definition 5.) will be given later in this article. For now, it is only important to understand its three-component structure:

1. *Name* of operator and its *parameters*,
2. two sets of *positive and negative applicability preconditions*,
3. two sets of so-called *effect descriptors*.

First component of IK-STRIPS operator is identical to that of its predecessor - name with $k \geq 0$ parameters. There are no modifications here.

As for the second component, we will (since we are working with an incomplete knowledge) use two sets of ELP facts instead of first-order-logic atoms: let's call them $preconditions^+$ and $preconditions^-$. Slightly similar solution of the problem with representation of actions preconditions with incomplete knowledge was proposed in [18], where $preconditions^-$ were called *constraints*. Operator/action is then applicable on a world model M, iff each fact from $preconditions^+$ and no fact from $preconditions^-$ AS-follows from $M \cup B$, where B is our background knowledge (which typically consists of ELP rules describing other than directly observable domain-specific knowledge).

Definition 1 (Applicability of operator / executability of action)**.** *Let B be a set of ELP rules representing background knowledge. Let P^+ and P^- be two sets of ELP facts representing $preconditions^+$ and $preconditions^-$ of operator O (action A). Operator O (action A) is then* applicable *on a world model M iff holds:*

$$\forall p \in P^+ : M \cup B \models_{AS} p \ \wedge \ \nexists n \in P^- : M \cup B \models_{AS} n$$

Third component of an action description defines its effects represented by so-called *effect descriptors* (definition 2) which are stored in two sets called (like in classical STRIPS [5]) *add list* and *delete list*. Effects in classical STRIPS were represented simply by single atoms and each of them were applied to any world model (if the whole action was executable). However, we want each of our effects to have some kind of applicability conditions saying when (on which world model) they can be applied. Therefore we introduced *effect descriptors* consisting, besides the actual description of effect (called *modifier*), also of their own conditions of applicability (called *positive and negative requirements*). Another difference difference between our effect descriptors and effects of classical STRIPS is, that STRIPS-like effects could only contain parametric variables, which were all substituted in advance. That means that there was no relation between current world model and the way they modified it (since those variables were substituted independently on given world model). Our effect descriptors can contain also non-parametric variables in both modifier and requirements. They are substituted by a constants from current world model, what allows effect descriptors to add/remove literals with different constants in accordance with current world model.

Definition 2 (Effect descriptor)**.** *Let e be an ELP literal and R^+, R^- two coherent sets of ELP literals. Triple (e, R^+, R^-) is called an* effect descriptor *of operator (action). We will call e a* modifier *and R^+, R^- sets of* positive and negative requirements *of effect descriptor (e, R^+, R^-). All literals in effect descriptor can contain variables, but if there is a variable in modifier, there has to be the same variable also in R^+ or R^-.*

Effect descriptor (e, R^+, R^-) intuitively says, that if R^+ is true and R^- is false, we modify our current world model by adding or deleting literal e. Classical STRIPS doesn't provide such conditions for applicability of a single effect. Effect descriptors allow us to represent actions more intuitively: In the introduction of this article we mentioned as an example the hunter agent with action $shoot(D)$. One of two effects of such action was conditional and said, that if something was in direction D, it was hit. Effect descriptor $(hit(Obj), \{direction\_to(Obj, D)\}, \emptyset)$ representing this could be a part of actions add list. As we have mentioned before, since classical STRIPS cannot handle conditional effects, those conditions would have to be moved to preconditions of an action itself, which would result in a need for more actions (in this case $shoot\_if\_direction\_free(D)$ and $shoot\_if\_direction\_obstructed(D)$).

Those effect descriptors specify how to create a new world model, using their modifier literals. However, since there can possibly be some non-parametric variables in effect descriptors, first thing to do is to get rid of them by substitution. This can produce several instances of one effect descriptor for each world model, because there can be more than one such possible substitution.

Definition 3 (Effect instance)**.** *Let M and B be two sets of ELP rules representing a world model and a background knowledge. Let (e, R^+, R^-) be an effect descriptor, possibly containing variables as parameters of some of its literals. Let $\theta_1 \dots \theta_k$ be all possible (substitution) functions $\theta_i((e, R^+, R^-)) = (e_i, R_i^+, R_i^-)$ such that each variable of (e, R^+, R^-) is substituted by a constant in (e_i, R_i^+, R_i^-) and holds:*

- $\forall l \in R_i^+ : M \cup B \models_{AS} l$
- $\nexists l \in R_i^- : M \cup B \models_{AS} l$

Then for each $i \in \{1 \dots k\}$ we call (e_i, R_i^+, R_i^-) an instance of effect descriptor (e, R^+, R^-). *We say that*

an effect descriptor is applicable *on a world model M w.r.t. background knowledge B iff it has at least one instance w.r.t. B.*

In other words, we have some effect descriptors for each action. Each of those descriptors has a set (possibly an empty set) of instances for current world model (w.r.t. BK). We produce those instances by first substituting its variables for constants and then checking positive and negative requirements. In each effect instance those requirements are met, and all the parameters are constants. Thus we consider them applicable and we are able to use their modifiers to compute new world models.

Computation of a world model by one effect descriptor depends on whether it is in add list or in delete list of an action.

Definition 4 (Computing the world models)**.** *Let O be an action operator of action A. Let M_1 be a set of ELP facts representing a world model. Let L_A be a set of all the modifier literals from all the instances (for model M_1, w.r.t. BK) of effect descriptors contained in* add list *of operator O (action A). Similarly, let L_D be a set of all the modifier literals from all the instances (for model M_1, w.r.t. BK) of effect descriptors contained in* delete list *of operator O (action A). M_2 is a world model* computed by applying operator O (or action A) *on the world model M_1 iff holds:*

$$M_2 = (M_1 \setminus L_D) \cup L_A$$

Notation: $M_1 \rhd_A M_2$ or $M_2 \lhd_A M_1$

Now, that we have defined the concepts of effect descriptors, their instances, and applicability of operator, we can finally give a formal definition of an "action" in IK-STRIPS.

Definition 5. *An* action *represented by IK-STRIPS is a double (R,O), where:*

1. *R is actions* routine - *procedure whose execution causes actual changes in the world,*
2. *and $O = (a, P^+, P^-, L_A, L_D)$ is corresponding actions* operator *consisting of:*
   (a) *actions name and parameters represented by an atom a,*
   (b) *preconditions represented by by two sets of ELP literals (P^+ and P^-) which define a conditions of applicability and inapplicability of the operator,*
   (c) *and two sets of effect descriptors* - add list L_A *and* delete list L_D.

In the following example we will see an action $move(B,L)$ of an artificial planning domain called Blocks World. The simplicity and clarity of this problem is the reason, why it became by far most frequently used example in planning-related literature and lessons. John Slaney and Sylvie Thiébaux describe this domain in [19]:

The Blocks World consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The positioning of the towers on the table is irrelevant. The Blocks World planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower or to the table (fig. 4).

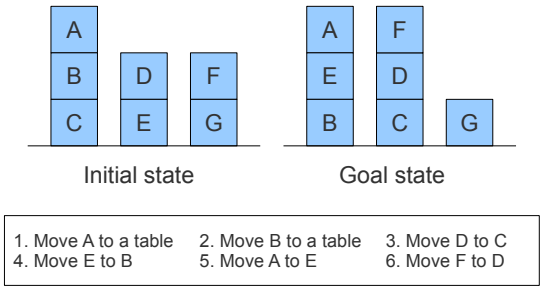

Figure 4: Blocks World planning problem and optimal plan according to [19].

In case of Blocks World domain, action $move(B,L)$ is the only action we need to have, in order to achieve any valid goal state. Following self-explanatory code depicts a definition of this actions operator in simple syntax processed by our IK-STRIPS planner:

```
action move(B,L) {

    def preconditions+ {
        block(B).
        location(L).
    }

    def preconditions- {
        occupied(L).
        blocked(B).
        equal(B,L).
        on(B,L).
    }

    def addlist {
        effectdescriptor on(B,L) {
            req+ {}
            req- {}
        }

    }

    def deletelist {
        effectdescriptor on(B,Fl) {
            req+ {
                on(B,Fl).
            }
            req- {
```

```
                    equal(L,Fl).
                }
            }
        }
}
```

Notice, that we have variables B and L as parameters of our operator (B stands for Block and L for Location). However, we will only try to apply an operator with B and L variables substituted by constants. Let's say that we want to move block a to the table. We will replace each B in P^+, P^-, *add list*, and *delete list* by constant "a". The same goes for L and constant "table". Then we will check the applicability by making sure that both facts $block(a)$ and $location(table)$, and none of the facts $occupied(table)$, $blocked(a)$, $equal(a,table)$, and $on(a,table)$ AS-follows from union of current world model and our background knowledge. After we found out, that our action is applicable on current world model, we can proceed to computation of a new world model by finding instances of effect descriptors and applying them.

The effect descriptor in delete list with binary literal $on(B,Fl)$ as its modifier is interesting because it contains a variable Fl (stands for Former location), which isn't in actions parameters. Therefore we need to substitute it during the process of generating effect instances by such a constant, that all the requirements of this descriptor would be satisfied. In other words, we need to find such a constant x, that $on(a,x)$ AS-follows and $equal(table,x)$ doesn't AS-follow from the union of current world model and BK. The number of constants that satisfy those two requirements equals to number of our effect instances. For example, if we had a world model $\{on(a,b),\ on(b,c),\ on(c,table)\}$ and a background knowledge $\{equal(X,Y) \leftarrow X = Y.\}$, our effect descriptor would have exactly one instance with Fl substituted by b, and the fact $on(a,b)$ would be removed from current world model. This simple example illustrates how effect instances are generated and how they affect the world model the operator is applied to. We can see, that effect descriptors from the delete list are capable of removing facts from models (when we move block a to the table, it is no longer on block b). Similarly, descriptors from add list can add some facts to world models (in this case, we would add a fact $on(a,table)$, since we have just moved a on the table).

Next concept that needs to be formally defined for IK-STRIPS formalism is a *goal*. Unlike in classical STRIPS, each goal in IK-STRIPS is represented by two sets of ELP facts - D^+ and D^- - together describing desired state of the world.

We can understand each literal in D^+ as one condition that an agent wants to be met and each literal in D^- as a condition that it wants not to be met. A goal is reached in any world state modelled by M, where $\forall p \in D^+ : M \models_{AS} p$, and $\nexists n \in D^- : M \models_{AS} n$.

Definition 6. *A* goal *in IK-STRIPS is a double* (D^+, D^-)*, where* D^+ *and* D^- *are two sets of ELP literals describing desired state of the world by defining positive and negative conditions respectively.*

Goal is an incomplete description of a world state. In some cases one of the sets D^+ and D^- contains only one literal, while other set is empty. A typical example of such simple goal is a double ($D^+ = \emptyset$, $D^- = \{energy(low)\}$). Set D^- contains a single fact, saying that an agent has low energy. Since this fact is in D^-, the goal is reached in any world state, from whose model this literal doesn't (w.r.t. BK) AS-follow. Naturally, a sequence of actions leading an agent to this goal state would include getting to some kind of power source and recharging its energy supply.

3.2 Plans

In classical STRIPS, plan is a sequence of actions with given constant parameters leading from initial state of the world to desired goal state. In deterministic domains where we can be sure about each actions effects, this approach is of course sufficient. However in real-world nondeterministic domains, we can never be sure that our actions will be executed successfully. Failure in execution of one action can cause the rest of our plan to be invalid. This is why we usually need some kind strategy for dealing with discrepancies in plan execution.

In this section we propose slightly more complex concept of plans for IK-STRIPS formalism, which could make dealing with discrepancies easier. However if reader is only interested in using our formalism in deterministic environments, he can skip this part of the text and understand plans as simple sequences of actions.

Instead of simple actions, IK-STRIPS plan can be understood as a sequence of several *stages*, each of which consists of a description of one single action and intended state of the world after its execution,

which allows us to check whether an action was executed successfully after each such stage. There are various heuristic-based approaches to solve the situation when the state of the world is inconsistent with intended state, but those are not the topic of this article.

Definition 7 (Plan)**.** *Let A be an IK-STRIPS action denoted by an atom of a form $a(p_1,\ldots,p_n)$, where a is actions name, and $p_1 \ldots p_n$ are ground terms. Let $I^+ = \{a \mid a$ is a modifier of any instance of any effect descriptor $e \in add\ list\ of\ A\}$ and $I^- = \{d \mid d$ is a modifier of any instance of any effect descriptor $e \in delete\ list\ of\ A\}$ be a pair of sets describing intended state of the world after execution of action A.*
Triple (A, I^+, I^-) is then called a plan stage*. A* plan *is any finite sequence of plan stages.*

Definition 8 (Leading)**.** *Let M_i be a model of initial state of the world and M_g a model of desired (goal) state. Let $A_1, A_2, \ldots, A_n$ be actions corresponding to individual stages of a plan P. We say a plan P* leads *from initial world state (modelled by M_i) to goal world state (modelled by M_g) iff:*

$$M_i \rhd_{A_1} M_1,\ M_1 \rhd_{A_2} M_2,\ \ldots,\ M_n \rhd_{A_n} M_g$$

Definition 9 (Correct plan)**.** *We say, that a plan P consisting of stages $S_1, \ldots, S_n$ leading from a world state modelled by M_1 to a world state modelled by M_n is* correct*, iff for each subsequent stages S_i, S_{i+1} with corresponding world models M_i, M_{i+1} holds:*

$$M_i \rhd_{A_i} M_{i+1} \text{ and } A_{i+1} \text{ is applicable on } M_{i+1}$$

where A_i and A_{i+1} are actions of stages S_i and S_{i+1} respectively.

Remark 2. IK-STRIPS planner (which will be introduced in section 5) produces *correct* plans *leading* (definitions 9 and 8) from initial state of the world, to a goal state. Intuitively it means, that from the model of initial world state, we can compute (definition 4) a model of goal state by subsequently applying all the actions of our produced plan.

4 Expressive Power

To show that expressive power of IK-STRIPS is greater than that of classical STRIPS formalism, we are going to prove that it can be used to define higher number of actions. In other words $|A| > |B|$, where A and B denote the sets of all actions that can be described by IK-STRIPS and STRIPS formalism respectively.

1. $B \subseteq A$

   Each action that can be described in classical STRIPS can also be easily expressed in IK-STRIPS. We can convert it easily in three steps:

   (a) copy name of action and its parameters,

   (b) put all literals from STRIPS-like preconditions into pre^+ and leave pre^- empty,

   (c) and finally for each literal l in STRIPS-like add list and delete list create an effect descriptor without any requirements, with l as its modifier.

   Since we can see that each STRIPS-like action can be converted into IK-STRIPS formalism, we can say that B is a subset of A. Next thing we need to show is, that there are some actions that can be described by IK-STRIPS, but not classical STRIPS.

2. $A \setminus B \neq \emptyset$

   Existence of IK-STRIPS actions which cannot be expressed by classical STRIPS can very easily be proven by an example of such action. The $shoot(D)$ action, that we mentioned in the first chapter of this article (named *Motivation*) is an example good enough for us.

   As another example let's imagine a trivial real-world action of drinking a glass of water - *drink(G)*. To make things simple, it can be understood as an action with one negative executability precondition *empty(G)* and two effects: 1.) glass G becomes empty after we drink it, and 2.) if there was a poison in it, actor will get sick. First effect descriptor with modifier *empty(G)* is obviously without any additional applicability conditions and can still be expressed in classical STRIPS. However, second effect with modifier *i_am_sick* has applicability condition *contains_poison(G)*, and can only be directly expressed in IK-STRIPS. With classical STRIPS, we would need to have two separate actions *drink_safe_glass(G)* and *drink_poisoned_glass(G)* to express such situation.

3. $B \subseteq A \wedge A \setminus B \neq \emptyset \implies |B| \leq |A|$ □

The reason why IK-STRIPS representation can be used to express greater set of actions is the fact, that it provides individual requirements for applicability of each effect of one action and allows effects to modify different world models differently by using variables as parameters. This not only makes representation of actions richer, but also more intuitive and similar to natural languages.

5 Implementation and Performance

Another result of our research is an experimental implementation of planner using IK-STRIPS as its formalism for representation of actions and planning problems.

We have chosen the Python as our programming language because of its usability on multiple platforms and ease of coding itself. Drawback of this choice in comparison with other alternatives is, that Python scripts tend to run slower than programs written in lower-level languages, such as C or C++. However, as we will show later in this section (fig. 7), IK-STRIPS planner runs faster than its common alternative DLV^K in typical cases of planning problems with increasing length of shortest possible plan.

Planning itself was reduced to the problem of pathfinding on an oriented graph, where each node represents one world model and edges are actions executable in that world model. Planning problem is then understood as a problem of finding the shortest possible path from a node representing an initial world state to one of the nodes representing goal states.

From the variety of available pathfinding algorithms, we have chosen an informed algorithm A^* [9][17], which allows us to optimize the search by choosing to visit always the most promising node from currently known part of search-space by evaluation using distance-plus-cost heuristic function. There are numerous possibilities for heuristic determination of the cost of given node. For our implementation we have chosen to simply take into account how many goal conditions are met in a world model represented by current node. This method works well in typical cases with goals described by numerous literals. Alternative heuristics could for example favour actions with some of goal conditions among the modifiers of their effect descriptors, favour world models with greater set of executable actions, or simply try to compute the sketch of plan in advance by using monotonic classical STRIPS-like planning and try actions contained in it first.

Since the most time consuming part of planning was computation of answer sets of individual world models (which we need to have in order to be able to determine which actions are executable and which effects are applicable), our next optimization was on-the-run reorganization of our graph, so that we never compute answer sets of two identical nodes.

For the generation of answer sets itself, our planner can use one of the following external ASP solvers: Smodels [13], Cmodels [8], or DLV [14]. Those three are the most commonly used tools in answer set programming. Even though each of them has its own advantages and disadvantages, in case of planning problems we have tested, their performance was very similar. When comparing our planner against DLV^K we used DLV solver, since it is also used by DLV^K planner.

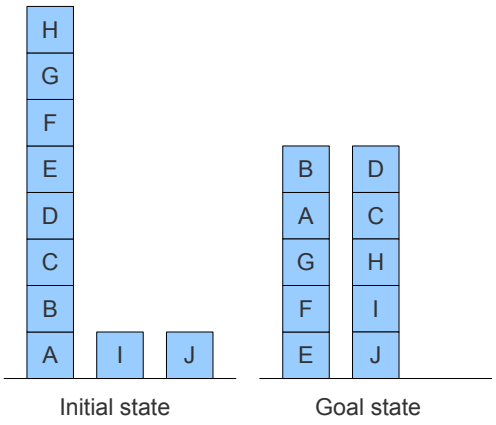

Figure 5: Example of Blocks World instance used in our comparison.

For the comparison, we generated six different instances of Blocks World problem with varying length of shortest possible plan. Figure 5 depicts one of Blocks World instances we used. There are ten blocks in this example, and the length of shortest possible plan leading from initial to goal state is 13.

Each instance was computed five times and resulting times were averaged. In order to achieve as exact measurement as possible, we used Unix program *time*. Exact times in seconds can be found in figure 6.

Naturally, more difficult planning problems have longer computation times. We can however see, that

DLV^K

Min. Plan Length	1st	2nd	3rd	4th	5th	Average
9	1.704	1.720	1.724	2.371	1.717	1.847
10	2.176	2.227	2.244	2.250	2.219	2.223
11	3.468	3.550	3.585	3.585	3.579	3.553
12	15.327	15.496	15.395	15.351	15.457	15.405
13	20.138	20.116	20.177	20.149	20.139	20.144
14	518.56	510.34	510.75	515.05	521.08	515.160

IK-STRIPS

Min. Plan Length	1st	2nd	3rd	4th	5th	Average
9	0.874	0.892	0.943	0.896	0.886	0.898
10	1.409	1.503	1.446	1.474	1.495	1.465
11	2.265	2.272	2.227	2.378	2.285	2.285
12	2.528	2.576	2.521	2.619	2.598	2.568
13	4.646	4.564	4.273	4.287	4.152	4.384
14	5.393	5.817	5.970	6.014	6.025	5.844

Figure 6: Computation times of individual instances of Blocks World problem by both DLV^K and IK-STRIPS planner.

the time needed for our planning task grows much faster when using DLV^K planner, and that it becomes practically unusable for computation of plans longer than 13 (since computation time exceeds 500 seconds). Better performance of IK-STRIPS planner is perhaps more obviously depicted in figure 7. It depicts the average computation times of both planners in relation to minimum plan length.

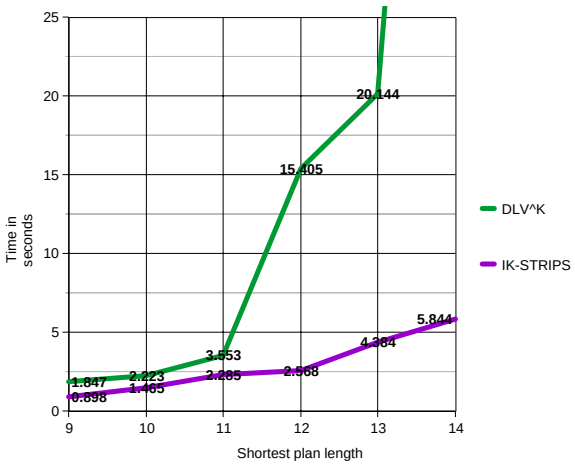

Figure 7: Average computation times of Blocks World problem instances using either DLV^K or IK-STRIPS planner.

Taking into consideration heuristic function used in IK-STRIPS planner, it is naturally also possible to design a special cases of planning problems, that would cause it to perform worse than other alternative planners. This is not surprising, since it is a common drawback of all heuristic algorithms (including the A^* algorithm, which was chosen as the basis for implementation of our IK-STRIPS planner).

6 Conclusion

Widespread approach to representation of planning problems is to use the concept of domain-specific fluents to describe all of its action and time dependant parts. The introduction of IK-STRIPS planning formalism solves potential problems resulting from occasional need to separate action definitions from the rest of the planning problem. Despite being inspired by obsolete STRIPS formalism from 1970s, it can be effectively used in complex domains with incomplete knowledge using answer set semantics.

After the formal definition of individual concepts of IK-STRIPS formalism, we have shown that its expressive power is greater than that of its predecessor.

Implementation of an experimental planner using this formalism is included in the section 5 of this paper along with the comparison to commonly used planning engine DLV^K, which uses alternative planning language $\mathcal{K}$. We have shown, that the average computation time of increasingly complex planning tasks is significantly lower using the IK-STRIPS planner.

The goal of further development concerning this topic will be aimed to speeding up the process of planning in complex domains even more by finding more effective heuristics, paralelization of used algorithms, and studying the possibilities of partial generation of answer sets.

References

[1] Cocosco, C. A. 1998. *A Review of "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving by R.E. Fikes, N.J. Nillson, 1971"*.

[2] Dix, J. - Brewka, G. 1996. *Knowledge Representation with Logic Programs*. Dept. of CS of the University of Koblenz-Landau: 1996.

[3] Eiter, T. - Faber, W. - Leone, N. - Pfeifer, G. - Polleres, A. 2000. *Planning under Incomplete Knowledge*. *Lecture Notes in Computer Science 2000*. Springer Berlin / Heidelberg: 2000.

[4] Eiter, T. - Erdem, E. - Faber, W. 2004. *Diagnosing Plan Execution Discrepancies in a Logic-based Action Framework*. *INFSYS Research Report 2004*.

[5] Fikes, R. E. - Nilsson, N. J. *STRIPS: A new approach to the application of theorem proving to problem solving*. *Artificial Intelligence 2*: 1971.

[6] Gelfond, M. - Lifschitz, V. 1998. *Action Languages*. *Electron. Trans. Artif. Intell. 2*. 1998.

[7] Gelfond, M. - Lifschitz, V. *Classical Negation in Logic Programs and Disjunctive Databases*. *New Generation Computing*: 1991.

[8] Giunchiglia, E. - Lierler, Y. - Maratea, M. 2004. *Cmodels-2: SAT-Based Answer Set Programming*, *Proceedings of AAAI*. 2004.

[9] Hart, P. E. - Nilsson, N. J. - Raphael, B. 1972. *Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"*. *SIGART Newsletter 37: 28–29*. 1972.

[10] Hendler, J. - Kitano, H. - Nebel, B. 2008. *Handbook of Knowledge Representation*. Elsevier: 2008.

[11] Lifschitz, V. 2002. *Answer Set Programming and Plan Generation*. *Artificial Intelligence vol. 138*: 2002.

[12] Minker, J. - Ruiz, C. *Semantics for disjunctive logic programs with explicit and default negation*. *Fundamenta Informaticae*: 1994.

[13] Niemela, I. - Simons, P. - Syrjanen, T. *Smodels: a system for answer set programming*. *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*. 2000.

[14] Polleres, A. 2001. *The DLVK System for Planning with Incomplete Knowledge*. Masters thesis, Vienna University of Technology, Austria. 2001.

[15] Rao, A. S. - George, M. P. 1992. *An Abstract Architecture for Rational Agents*. *Knowledge Representation and Reasoning*. 1992.

[16] Rosenberg, S. 1978. *Understanding in Incomplete Worlds*. MIT Artificiel intelligence Laboratory: 1978.

[17] Russell, J. - Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs: 1995.

[18] Simari, G. R. - Garcia, A. J. - Capobianco, M. 2004. *Actions, Planning and Defeasible Reasoning*. 10th International Workshop on Non-Monotonic Reasoning. 2004.

[19] Slaney, J. - Thiébaux, S. 2001. *Blocks World revisited*. *Artificial Intelligence 125: 119-153*. 2001.

[20] Subrahmanian, V. S. - Zaniolo, C. 1995. *Relating Stable Models and AI Planning Domains*. Proceedings of ICLP-95.