Constraint Based Reasoning over Mutex Relations in Graphplan Algorithm

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
surynek@ktiml.mff.cuni.cz

Abstract. This paper describes a work in progress. We address a problem of reachability of states in planning from the constraint programming perspective. We propose an alternative way how planning graphs known from the Graphplan algorithm can be constructed. We propose to do an additional consistency check every time when our variant of the planning graph is extended. This leads into a more accurate approximation of reachable states than it is done by the standard Graphplan algorithm. It is theoretically shown in this paper that our variant of planning graph forbids strictly more situations than it is forbidden by the standard planning graphs.

1 Introduction

Planning is widely studied area of artificial intelligence. The importance of studying planning arises from the needs of real-life applications such as industrial automation, logistics, robotics and other braches [22]. The research in planning is also motivated by the needs of researches in other areas. The most prominent example in this sense is the space exploration where autonomous spacecrafts [5] and vehicles [1] are successfully used. But there are many other situations both in science and real-life where autonomous devices are used. The autonomous behavior is controlled by planning techniques and algorithms in many of these cases.

From the traditional view of planning, the planning problem is posed as finding of a sequence of actions which transform a specified initial state of the planning world into a desired goal state of the world [2,11]. The limitation is that only actions from a set of allowed actions can be used. The action typically makes a small local change of the state of the world.

Among the most successful techniques for solving planning problems belong algorithms based on state reachability analysis. The first such algorithm was Graphplan [7]. The algorithm introduced a notion of so called *planning graphs*. The planning graph is a structure from which it is possible to answer the question whether a certain state of the planning world can be reached by using a certain set of actions. In this paper we concentrate on planning graphs from the constraint programming perspective.

The paper is organized as follows. First we discuss how constraint programming techniques are exploited within planning. We concentrate especially on works that are based on state reachability analysis. Next we introduce some basic definitions and facts from constraint programming and planning methodology. The main part of the paper presents a new method for construction of planning graphs.

2 Related works

Several techniques for solving planning problems are trying to translate the problem into another formalism. Then they solve the problem in this formalism. Many of these approaches use Boolean formula (SAT) or constraint satisfaction as the target formalism. SAT based planners are described in [15,17,18]. The drawback of these methods is that the information induced by the original formulation is often lost during translation into the target formalism. Some planners are trying to overcome this drawback by hand tailored encoding of a planning problem into the target formalism [23].

The significant breakthrough in planning was done when reachability analysis using planning graphs was incorporated into the planners. Many of the successful existing planners use some Boolean formula or constraint satisfaction algorithms to solve the sub-problems arising in planning graph analysis [3,10,13,14,16,19]. Constraint programming represents a technique which is intensively used in this way [21]. This kind utilization of constraint programming in planning is more typical than the direct translation of the problem from one formalism into another. In this paper we will use constraint programming just in this way.

3 CSP and Planning Basics

We provide some basic definitions in this section. First we briefly introduce what the constraint satisfaction is. Then we introduce planning problems in more details.

A *constraint satisfaction problem* (CSP) is a triple (X,D,C) [8], where X is a finite set of variables, D is a finite domain of values for the variables from X and C is a finite set of constraints over the variables from X. The constraint can be an arbitrary relation over the elements of the domains of its variables.

Having a constraint satisfaction problem the task is to find an assignment of values from D to all the variables from X such that all the constraints from C are satisfied. The problem of finding a solution of the constraint satisfaction problem is NP-hard in general [8].

Now, we will formally define the problem of finding a plan. We are using the similar notation and definitions as it is used in [11]. Consider that we have a first-order language L with finitely many predicate and constant symbols. We do not have any function symbols and variable symbols. Thus all constructs over the language L are ground.

Let l be a set of literals, then l^+ denotes a set of all atoms occurring in positive literals of the set l and l^- denotes a set of all atoms occurring in negative literals of the set l.

Definition 1 (State). A *state s* is a finite set of atoms. Semantically it is a set of propositions that are true in a certain state of the planning world.

Definition 2 (Action). An action *a* is a pair *(precond(a), effects(a))*, where *precond(a)* is a finite set of atoms and *effects(a)* is a finite set of literals for which a condition $effects^{+}(a) \cap effects^{-}(a)=\emptyset$ holds. Semantically the action determines how the state can be changed (the change is specified by *effects(a)*) provided that the state allow the use of the action (the allowance is specified by *precond(a)*).

Definition 3 (Applicability). An action *a* is *applicable* to the state *s* if and only if $precond(a) \subseteq s$. The *result* of the application of the action *a* to the state *s*, where *a* is applicable to *s*, is a new state $\gamma(s,a)$, where $\gamma(s,a)=(s-effects^{-}(a)) \cup effects^{+}(a)$.

A no-operation action (*noop*) is also considered to be a valid action. It does not change anything when it is applied on the state. The no-operation *noop(p)* action is associated with every possible atom *p*. Formally we have an action $noop(p)=(\{p\}, \{p\})$ for every atom *p*.

Definition 4 (Goal). A *goal g* is a finite set of literals. The goal *g* is satisfied in a state *s* if and only if $g^{+} \subseteq s$ and $g^{-} \cap s=\emptyset$. Semantically the goal is a set of propositions we want to be true in a certain state.

Given a set of actions and a goal the task is to find out how to reach a state satisfying the given goal by using the allowed actions only. The whole process of finding of how to satisfy the goal starts in a specified initial state of the planning world. This notion is described in the following definitions more formally.

Definition 5 (Problem). A *planning problem P* is a triple (s_0,g,A), where s_0 is an initial state, *g* is a goal and *A* is a finite set of allowed actions. Semantically the initial state describes the planning world state at the beginning and *g* represents a condition which a state we want to reach must satisfy. The required goal can be satisfied by using the allowed actions from the set *A* only.

Definition 6 (Solution). We inductively define the *application of a sequence of actions* $\theta=(a_1,a_2,...,a_n)$ to a state s_0 in the following way: a_1 must be applicable to the state s_0, let us denote $s_i=\gamma(s_{i-1},a_i)$, then a_i must be applicable to s_i for all $i=2,...,n$. The result of the application of the sequence of actions θ to the state s_0 is the state s_n. We denote $s_n=\gamma(s_0,\theta)$. The sequence $sol=(a_1,a_2,...,a_n)$ is a *solution* of the planning problem $P=(s_0,g,A)$ if and only if the sequence *sol* is applicable to the initial state s_0 and *g* is satisfied in the result of the application of the sequence *sol* and $a_i \in A$ for $i=1,2,...,n$.

4 Graphplan Algorithm

The Graphplan algorithm [7] relies on the idea of state reachability analysis. The standard formulation of the Graphplan algorithm puts an additional restriction on goals. Negative literals are not allowed in a goal. The goal has to be a finite set of atoms. Since the preconditions of actions are also a finite set of atoms (definition 2), the preconditions of actions are goals as well in the standard Graphplan formulation.

The state reachability analysis is done by constructing a data structure called *planning graph* in the Graphplan algorithm. The algorithm works in two interleaved phases. In the first phase planning graph is incrementally expanded. The second phase consists of an extraction of a valid plan from the extended planning graph. If the second phase is unsuccessful the process continues with the first phase - the planning graph is extended again.

The planning graph for a planning problem $P=(s_0,g,A)$ is defined as follows. It consists of two alternating structures called *proposition layer* and *action layer*. The initial state s_0 represents the *0*th proposition layer P_0. The layer P_0 is just a list of atoms occuring in s_0. The rest of the planning graph is defined inductively. Consider that the planning graph with layers P_0, A_1, P_1, A_2, P_2,..., A_k, P_k has been already constructed (A_i denotes the *i*th action layer, P_i denotes the *i*th proposition layer). The next action layer A_{k+1} consists of actions whose preconditions are included in the *k*th proposition layer P_k and which satisfy the additional condition. This additional condition requires that no two propositions of the action are *mutually excluded* (we briefly say that they are *mutex*). The mutual exclusion relation will be defined inductively using the following definitions.

Definition 7 (Independence). A pair of actions $\{a,b\}$ is *independent* if and only if:
(i) $effects^-(a)\cap(precond(b)\cup effects^+(b))=\emptyset$ and
(ii) $effects^-(b)\cap(precond(a)\cup effects^+(a))=\emptyset$.
Otherwise $\{a,b\}$ is a pair of *dependent* actions. A set of actions π is *independent* if and only if every pair of actions $\{a,b\}$ from π is independent.

Definition 8 (Action mutex / mutex propagation). We call the two actions a and b within the action layer A_i a *mutex* if and only if either the pair $\{a,b\}$ is dependent or an atom of the precondition of a is *mutex* with an atom of the precondition of b (defined in the following definition).

Definition 9 (Proposition mutex / mutex propagation). We call the two atoms p and q within the proposition layer P_i a *mutex* if and only if every action a within the layer A_i where $p\in effects^+(a)$ is mutex with every action b within the layer A_i for which $q\in effects^+(b)$ and layer A_i does not contain any action c for which $\{p,q\}\subseteq effects^+(c)$.

Action and proposition mutexes are represented in the planning graph as special links between nodes representing actions and propositions. We have just defined the $(k+1)$th action layer A_{k+1}. The $(k+1)$th proposition layer contains all the propositions that appear as the effect of some action in the $(k+1)$th proposition layer.

Theorem 1 (Necessary condition on state reachability). Consider a state s containing atoms p and q that are mutex in layer P_i. Then the state s cannot be reached from the initial state s_0 by any sequence of actions determined by the action layers A_1, A_2,..., A_i.

We omit the proof of the theorem since it is given in details in [7]. The theorem gives the necessary condition for the existence of a solution of the planning problem. In other words, a valid plan can be extracted from the planning graph only if atoms of g are contained in some proposition layer P_j and there is no mutex between any two atoms from g at layer P_j. The Graphplan algorithm utilizes the result of the proposition for reduction of the search space that is necessary to be explored during the search for a solution.

If we look closer on the structure of the planning graph from the perspective of the constraint programming methodology we can observe that the mutex network or even the whole planning graph resembles some kind of a constraint satisfaction problem. This observation was developed by a number of authors into several elaborate methods for solving planning graphs using constraint satisfaction or related techniques [3,10,13,14,19]. The problem of extraction of a plan from the planning graph can be viewed as a *dynamic constraint satisfaction problem* [9,10,13]. Another method was developed in [19] where the authors propose a translation of the planning graph into a standard CSP. All the mentioned approaches are designed to provide an effective extraction of a valid plan from the model. Contrary to these works we concentrate on the phase of planning graph expansion.

5 Planning with State Variables

It is possible to use another approach for the representation of planning domain which is more suitable with respect to constraints. Such approach is for example a so called *state variable representation*. Instead of saying that some proposition holds in a certain situation we say that a certain property takes a certain value in that situation. The planning domain in state variable representation consists of a finite set of *state variable functions* $f_1, f_2, \ldots, f_m$, where $f_i: S \rightarrow D(f_i)$ for $i \in \{1,2,\ldots,m\}$. S denotes the set of possible states of the planning world and $D(f_i)$ is the domain of the state variable function f_i. An individual state variable function represents a single property of the object residing in the planning world (for example we can have a state variable function representing the location of a robot, the domain of such function would be all the possible locations where the robot can go). A state variable F_i is associated with every state variable function f_i for $i \in \{1,2,\ldots,m\}$.

Definition 10 (State in state variable representation). A *state s* in state variable representation is a finite set of assignments of the form $F_j = d_k$, where $d_k \in D(f_j)$.

Definition 11 (Action in state variable representation). An action a in state variable representation is a pair *(precond(a), effects(a))*, where *precond(a)* and *effects(a)* are finite set of assignments of the form $F_j = d_k$, where $d_k \in D(f_j)$.

Definition 12 (Goal in state variable representation). A *goal g* in state variable representation is a finite set of propositions of the form $F_j \ R \ d_k$, where $d_k \in D(f_j)$ and $R \in \{=, \neq\}$.

We omit the translation of *applicability*, *planning problem* and *solution* into the state variable representation since it is almost obvious. The planning graphs can be easily defined over the state variable representation. The relation of independence of the actions as well as the relation of mutual exclusion and its propagation can be defined within the state variable formulation of the planning problem.

As in the standard Graphplan algorithm we restrict the definition of goals. A goal is a finite set of assignments (negative assignments are not allowed in the goal). Thus the goal is a state within our Graphplan formulation as well.

6 Mutex Relations and Constraints in the Graphplan Algorithm

We want to propose a technique for reasoning over mutex relations in order to rule out more incompatibilities than it is done by existing Graphplan based algorithms. Such a technique would provide a more accurate approximation of reachable states within the planning graph. The necessary condition for plan existence would be tighter. In the following paragraphs we will describe such technique in several steps. We start with a simple method and we will refine the method step by step.

Consider all the state variable functions $f_1, f_2, ..., f_m$ describing attributes of all the objects appearing in the given planning world. Next consider all the actions applicable to a certain state s. Some of these actions are dependent. Each action changes the value of certain state variables corresponding to state variable functions. For each such a change of state variables let us assign a triple (a_i,F_j,d_k), where a_i is an action, F_j is a state variable corresponding to the state variable function f_j and d_k is a value from the domain $D(f_j)$ of the state variable function f_j. The triple (a_i,F_j,d_k) says that the action a_i has an assignment $F_j=d_k$ as its effect when it is applied to the state s.

Now we will be constructing a special graph $G_s=(V_s,E_s)$ corresponding to application of actions to the state s. We want the graph G_s to represent information determining what states can be reached form the state s and what states cannot by using the actions applicable to the state s only. This is exactly what is done within the Graphplan algorithm. In other words we are asking what states can be reached in the proposition layer following the layer representing the state s.

The nodes V_s of the graph G_s are all the possible triples (a_i,F_j,d_k) as described above (each vertex corresponds to an applicable action and its effect). The set of edges E_s of the graph contains a pair $\{(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2})\}$ if and only if the actions a_{i1} and a_{i2} are dependent with respect to the state s (using the terminology of mutual exclusion we can say a_{i1} and a_{i2} to be mutex).

Proposition 1 (Condition on reachable states from a state). It is possible to reach a state $t=[F_{j1}=d_{k1}, F_{j2}=d_{k2}, ..., F_{jl}=d_{kl}]$, where $j_1,j_2,...,j_l \in \{1,2,...,m\}$; $k_1,k_2,...,k_l \in \{1,2,...,m\}$ and $d_{k1} \in D(f_{j1}), d_{k2} \in D(f_{j2}), ..., d_{kl} \in D(f_{jl})$ from the state s by using the actions applicable to the state s if and only if the following conditions hold for the graph $G_s=(V_s,E_s)$:

(i) $G_s=(V_s,E_s)$ contains vertexes $(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2}); ...; (a_{il},F_{jl},d_{kl})$ for some actions $a_{i1}, a_{i2}, ..., a_{il}$, let us denote this set of vertexes $V(t)$, and

(ii) there is no edge in E_s connecting any two vertexes from $V(t)=\{(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2}); ...; (a_{il},F_{jl},d_{kl})\}$, in other words V(t) is a stable set of G_s [12].

Notice that the condition is equivalence. If the condition is not satisfied then the specified state cannot be reached.

Proof. The proof of the proposition is almost obvious. If (i) and (ii) hold, then it is possible to select a sequence of actions from an independent set of actions transforming the given state s into the given state t. On the other hand, if there does not exist any set $V(t)=\{(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2});...; (a_{il},F_{jl},d_{kl})\}\subseteq V_s$ for any actions $a_{i1}, a_{i2},..., a_{il}$ then the state t is unreachable obviously. If for any $V(t)=\{(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2});...; (a_{il},F_{jl},d_{kl})\}\subseteq V_s$ for any actions $a_{i1}, a_{i2},..., a_{il}$ there is an edge from E_s between some two vertexes in $V(t)$, it is not possible to reach the state t because of the properties of the independence relation. ■

Let us consider the following constraint model for the situation from the proposition 1. To model the situation using constraints we can post a Boolean variable for each vertex (a_i,F_j,d_k) of G_s, let us denote the variable as $v(a_i,F_j,d_k)\in\{true, false\}$. The semantics is straightforward, the action a_i is selected into the sequence of actions for satisfying the new state t if the variable $v(a_i,F_j,d_k)$ takes the value *true*. Edges of E_s are represented as (binary) difference constraints between corresponding variables. Since it is possible to have more than one Boolean variables corresponding to a single action it is necessary to ensure consistency in selection of the actions. It can be done by adding constraints $v(b,F_{j1},d_{k1})= v(b,F_{j2},d_{k2})$ for all the possible combinations of action b and F_{j1},d_{k1},F_{j2} and d_{k2}.

This model is simple but it is more difficult to solve it. The solution is an assignment of Boolean values to the variables. The variables that have assigned the *true* value form the set of actions satisfying (i) from the proposition 1 and none of the constraints is violated. Notice that this is not a pure constraint satisfaction problem according to the definition described in section 3. To solve the problem it is necessary to detect a stable set [19] satisfying additional conditions. Without a proof we can conjecture it is a hard problem. Detection of stable sets is discussed in details in [12].

Let us consider another more sophisticated model. We introduce a variable $a(F_j,d_k)$ into the model if there is a vertex (a_i,F_j,d_k) in the graph G_s. The domain of the variable $a(F_j,d_k)$ will contain an element for every action that has the assignment $F_j=d_k$ as its effect. A symbol a_i is in the domain of $a(F_j,d_k)$ if there is a vertex (a_i,F_j,d_k) in the graph G_s. An additional value $\perp$ is introduced into each variable domain. The semantics is expectable. If a variable $a(F_j,d_k)$ takes the value a_i it means that it was selected to gain the effect $F_j=d_k$ by application of the action a_i. If a variable $a(F_j,d_k)$ takes the value $\perp$ it is interpreted as if the assignment $F_j=d_k$ remains unsatisfied.

Now we are ready to introduce constraints which would describe conditions that must hold in the model to be a proper model of our situation. First we can imagine that there are empty constraints between every pair of variables in the model at the beginning. In the following steps we will refine these constraints by forbidding pairs of values.

If there is an edge $\{(a_{i1},F_{j1},d_{k1}); (a_{i2},F_{j2},d_{k2})\}$ in the graph G_s, then the pair of values (a_{i1},a_{i2}) in the domains of $a(F_{j1},d_{k1})$ and $a(F_{j2},d_{k2})$ respectively becomes forbidden. Informally said we forbid any pair of dependent actions with respect to the state s.

Proposition 2 (Correctness of the constraint model). It is possible to reach a state $t=[F_{j1}=d_{k1}, F_{j2}=d_{k2},..., F_{jl}=d_{kl}]$, where $j_1,j_2,...,j_l \in \{1,2,...,m\}$; $k_1,k_2,...,k_l \in \{1,2,...,m\}$ and $d_{k1} \in D(f_{j1})$, $d_{k2} \in D(f_{j2}),...,$ $d_{kl} \in D(f_{jl})$ from the state s by using the actions applicable to the state s if and only if the described constraint model with an additional constraint $(a(F_{j1},d_{k1}) \neq \perp) \& (a(F_{j2},d_{k2}) \neq \perp) \& ... \& (a(F_{jl},d_{kl}) \neq \perp)$ has a solution.

Proof. If the constraint with the additional constraint $(a(F_{j1},d_{k1}) \neq \perp) \& (a(F_{j2},d_{k2}) \neq \perp) \& ... \& (a(F_{jl},d_{kl}) \neq \perp)$ has a solution, the solution determines the set of independent actions. These independent actions applied in an arbitrary order to the state s results into the state t. On the other hand if the state t can be reached from state s it must be done by independent actions only (a dependent action would destroy the effects of previously applied actions). ■

The above constraint model represents only the initial proposition layer of the Graphplan algorithm. We will now extend the model to capture the whole planning graph. We will proceed by induction according to the number of layers of the planning graph. Consider we have already constructed the constraint model for $m>1$ first layers. In the following we will call such model a model of size m. The variables representing the constraint model of the ith layer will be denoted as $a[i]$ (for example a variable $a[i](F_{jl},d_{kl})$ stands for the variable $a(F_{jl},d_{kl})$ within the ith layer).

Definition 13 (Strong reachability). Let us have a constraint model M of size m. A state s is *strong-reachable* with respect to the constraint model M if and only if the constraint model contains a variable $a[m](F_j,d_k)$ for every $(F_j=d_k) \in s$ and the constraint model M with the additional constraints $a[m](F_j,d_k) \neq \perp$ for every $(F_j=d_k) \in s$ has a solution.

Definition 14 (Strong applicability). Let us have a constraint model M of size m. An action $a=(precond(a),effect(a))$ is *strong-applicable* with respect to the constraint model M if and only if the constraint model contains a variable $a[m](F_j,d_k)$ for every $(F_j=d_k) \in precond(a)$ and the constraint model M with the additional constraints $a[m](F_j,d_k) \neq \perp$ for every $(F_j=d_k) \in precond(a)$ has a solution.

To check the applicability of an action according to the above definition is as hard as to extract a valid plan leading to the satisfaction of the precondition of the action. This conjecture will be proved in the following sections. The applicability of an action has to be checked easily in order to be used within an algorithm. Therefore this definition cannot be utilized for building of a new algorithm.

The following definitions are tying to resolve this drawback. It is possible to require something weaker than the existence of a solution. Such a weaker requirement is represented by a *consistency technique* from the constraint programming methodology. It is much easier to check a consistency of the constraint satisfaction problem than to solve it. It depends on the type of the consistency technique but most of the standard consistency techniques require polynomial time. Contrary to this the problem of finding of a solution of the CSP is NP-hard.

Definition 15 (Weak reachability). Let us have a constraint model M of size m and consistency technique C. A state s is *weak-reachable* with respect to the constraint model M and consistency technique C if and only if the constraint model contains a variable $a[m](F_j,d_k)$ for every $(F_j=d_k)\in s$ and the constraint model M with the additional constraints $a[m](F_j,d_k)\neq\perp$ for every $(F_j=d_k)\in s$ is C-consistent.

Definition 16 (Weak applicability). Let us have a constraint model M of size m and consistency technique C. An action $a=(precond(a),effect(a))$ is *weak-applicable* with respect to the constraint model M and consistency technique C if and only if the constraint model contains a variable $a[m](F_j,d_k)$ for every $(F_j=d_k)\in precond(a)$ and the constraint model M with the additional constraints $a[m](F_j,d_k)\neq\perp$ for every $(F_j=d_k)\in precond(a)$ is C-consistent.

This weaker definition of applicability of an action is more suitable to be used within an algorithm. We will use the weak applicability to finish the definition of the constraint model. Consider that the constraint model M of size m is already constructed.

Let us again construct a special graph $G_m=(V_m,E_m)$ corresponding to the application of actions with respect to the constraint model M and consistency technique C. The nodes V_m of the graph G_m are all the possible triples (a_i,F_j,d_k) where a_i is weak-applicable with respect to the constraint model M and consistency technique C with $F_j=d_k$ as its effect.

The set of edges E_m of the graph contains a pair $\{(a_{i1},F_{j1},d_{k1});\ (a_{i2},F_{j2},d_{k2})\}$ if and only if the actions a_{i1} and a_{i2} are dependent with respect to the states represented by the mth layer of the constraint model.

Proposition 3 (Necessary condition on strong-reachable state). If it is possible to obtain a state $t=[F_{j1}=d_{k1},\ F_{j2}=d_{k2},\ldots,\ F_{jl}=d_{kl}]$, where $j_1,j_2,\ldots,j_l\in\{1,2,\ldots,m\};\ k_1,k_2,\ldots,k_l\in\{1,2,\ldots,m\}$ and $d_{k1}\in D(f_{j1}),\ d_{k2}\in D(f_{j2}),\ldots,\ d_{kl}\in D(f_{jl})$ from a strong-reachable state s with respect to the constraint model M by using the strong-applicable actions with respect to M. Then it is possible to reach the state t from the state s using the weak-applicable actions with respect to M and C.

Notice that the condition is an implication. If a state is not weak-reachable then it cannot be reached (with respect to the model). But if a state is weak-reachable we cannot conclude that it can be reached.

Proof. The proof directly flows from the correctness of the consistency technique C. And of course, the correctness of consistency technique C is supposed. ■

The constraint M of size m will be extended to the constraint model of size $m+1$ in the following paragraphs. A variable $a[m+1](F_j,d_k)$ is added to the model M if there exists a vertex (a_i,F_j,d_k) in the graph G_m. The domains of the variables will be following. A value $\perp$ is added into the domain of every new variable. A value a_i is added into the domain of the variable $a[m+1](F_j,d_k)$ if and only if there exists a vertex (a_i,F_j,d_k) in the graph G_m.

We add the constraints into the model in a similar way as it was done for the single layer model. Again we can imagine that every pair of variables in the $(m+1)$th layer of the model are constrained with an empty constraint. The empty constraint will be refined by forbidding new pairs of values.

A pair of values (a_{i1},a_{i2}) is forbidden in the domains of the variables $a[m+1](F_{j1},d_{k1})$ and $a[m+1](F_{j2},d_{k2})$ respectively if and only if there exists an edge $\{(a_{i1},F_{j1},d_{k1});\ (a_{i2},F_{j2},d_{k2})\}$ in the graph G_m. Contrary to the single layer constraint model another type of constraints is introduced. If it is found out that an action must be selected at a certain layer of the model then it is necessary to ensure that preconditions of the action are reachable. This is done by the conditional constraints of the following form: $(a[m+1](F_{j1},d_{k1})=b) \Rightarrow ((a[m](F_{j1},d_{k1}) \neq \perp) \& (a[m](F_{j2},d_{k2}) \neq \perp) \& \ldots \& (a[m](F_{jl},d_{kl}) \neq \perp))$, where $precond(b)=\{F_{j1}=d_{k1},\ F_{j2}=d_{k2}, \ldots,\ F_{jl}=d_{kl}\}$. Such constraint is added for every action that can took into account at the $(m+1)$th layer of the model.

Recall the definition of action and proposition mutex propagation. We simulate it in the constraint model as well. The two assignments $F_{j1}=d_{k1}$ and $F_{j2}=d_{k2}$ are *mutex* (with respect to the constraint model) at the m*th* layer of the model if and only if every action that produces $F_{j1}=d_{k1}$ is mutex (according to the definition 9) with every action producing $F_{j2}=d_{k2}$ as its effect. It is if and only if the constraint over the variables $a[m](F_{j1},d_{k1})$ and $a[m](F_{j2},d_{k2})$ respectively has $(\perp,\perp)$ as its only valid assignment of values. Two actions a_{i1} and a_{i2} are *mutex* (with respect to the constraint model) at the $(m+1)$th level of the model if either there is an edge $\{(a_{i1},F_{j1},d_{k1});\ (a_{i2},F_{j2},d_{k2})\}$ in graph G_m or if a precondition of the action a_{i1} is mutex with a precondition of the action a_{i2} (which we already know how to check). For the second condition we forbid a pair of values $(a_{i1},\ a_{i2})$ for every two variables $a[m+1](F_{j1},d_{k1})$ and $a[m+1](F_{j2},d_{k2})$ respectively that has a_{i1} and a_{i2} in their domains.

The constraint model for the planning problem $P=(s_0,g,A)$ is build by induction according to the described process. The process is initialized by constructing a single layer constraint model originating in the state s_0.

Proposition 4 (Correctness of the extended constraint model). If a goal $g=[F_{j1}=d_{k1},\ F_{j2}=d_{k2}, \ldots,\ F_{jl}=d_{kl}]$, where $j_1,j_2,\ldots,j_l \in \{1,2,\ldots,m\};\ k_1,k_2,\ldots,k_l \in \{1,2,\ldots,m\}$ and $d_{k1} \in D(f_{j1}),\ d_{k2} \in D(f_{j2}), \ldots,\ d_{kl} \in D(f_{jl})$ is strong-reachable with respect to the constraint model M constructed for a planning problem $P=(s_0,g,A)$. Then a plan satisfying the goal g exists. The plan can extracted from the solution of the constraint model M.

Proof. To prove the proposition we will proceed by induction according the number of layers of the constraint model. The initial induction step is ensured by the proposition 2. That is, a proposition 4 holds for the constraint model of size 1. Suppose that the proposition holds for the constraint model of size m. If a goal g is strong-reachable with respect to the constraint model M of size $m+1$ then the model M with the additional constraint $(a[m+1](F_{j1},d_{k1}) \neq \perp) \& (a[m+1](F_{j2},d_{k2}) \neq \perp) \& \ldots \& (a[m+1](F_{jl},d_{kl}) \neq \perp)$ has a solution. The values of the variables $a[m+1](F_{j1},d_{k1});\ a[m+1](F_{j2},d_{k2}) \neq \perp); \ldots;\ (a[m+1](F_{jl},d_{kl}) \neq \perp)$ in the solution determine the set of independent actions I. Each action from I has a set of preconditions that are strong-reachable with respect to the constraint model M without the last $(m+1)$th layer. By induction hypothesis a valid plan *sol* for reaching the set of precondition of actions from I exists. This plan can be extracted from the constraint model M without the last layer. A plan obtained by a concatenation of the plan *sol* with the actions form the set *I* in an arbitrary order is a valid plan for reaching the goal *t*. ■

Corollary 1 (Necessary condition on existence of a plan). If a goal $g=[F_{j1}=d_{k1}, F_{j2}=d_{k2},..., F_{jl}=d_{kl}]$, where $j_1,j_2,...,j_l \in \{1,2,...,m\}$; $k_1,k_2,...,k_l \in \{1,2,...,m\}$ and $d_{k1} \in D(f_{j1}), d_{k2} \in D(f_{j2}),..., d_{kl} \in D(f_{jl})$ is not weak-reachable with respect to the constraint model *M* for the planning problem $P=(s_0,g,A)$. Then it is not possible to extract any plan satisfying the goal *g* from the constraint model *M*.

The corollary provides the necessary condition on the existence of a plan. More specifically, if a goal is not weak-reachable then any plan of length at most *l* can satisfy the goal. The length *l* is the maximum length of the sequence of strong-applicable actions starting from the initial state s_0.

Proof. The corollary can be obtained directly by the combination of proposition 3 and proposition 4. ■

7 Singleton Arc Consistency and the Constraint Model

We propose to use *singleton arc-consistency (SAC)* [4,6] as the consistency technique in the definition of the weak-applicability of the actions. The singleton arc-consistency is suitable for our model since the filtering algorithms for SAC rule out values from the domains of variables. Singleton arc-consistency strengthens *arc-consistency* [20].

Definition 17 (Arc-consistency). The value *d* of the variable *x* is *arc-consistent* if and only if for every variable *y* connected to *x* by the constraint *c* there exists a value *e* in the domain of *y* such that the assignment $x=d$ & $y=e$ is allowed by the constraint *c*. The constraint satisfaction problem *(X,C,D)* is arc consistent if and only if every value of every variable is arc-consistent.

Definition 18 (Singleton arc-consistency). The value *d* of the variable *x* is *singleton arc-consistent* if and only if the constraint satisfaction problem restricted to $x=d$ is arc-consistent. The constraint satisfaction problem *(X,C,D)* is singleton arc-consistent if and only if every value of every variable is singleton arc-consistent.

The suitability of singleton arc-consistency becomes evident when some domains of variables of the model become unit due to the removals of values. Then it is possible to propagate though conditional constraints that makes connection between the layers of the constraint model. The conditional constraints rule out the value $\perp$ from the domains of variables in the previous layer of the model. This mechanism leads to a further propagation.

The utility of maintaining singleton arc-consistency is summarized in the following proposition.

Proposition 5 (Constraint model provides at least the same propagation). The mutex propagation (definitions 3, 4) is covered by the relation of weak-applicability of actions with singleton arc-consistency and by the properties of the constraint model.

Proof. It is sufficient to observe that the notion of action and proposition mutexes and their propagation (according to the definitions 3 and 4) is precisely translated into the proposed constraint model. The fact that a set of actions or propositions contains a mutex pair of actions or propositions is discovered by the singleton arc-consistency. In other words, it is done at least the same in the constraint model as it is done in the standard construction of the planning graph. ■

Observation 1 (Constraint model provides stronger propagation). A stronger propagation can be achieved in the proposed constraint model than it is achieved in the standard planning graph framework using the mutex propagation.

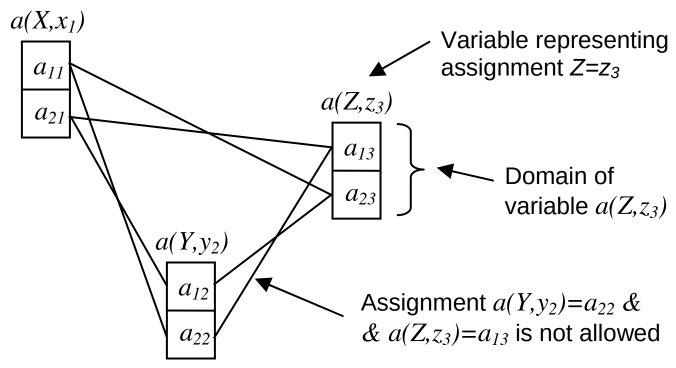
$a(X,x_1)$
a_{11}
a_{21}
Variable representing assignment $Z=z_3$
$a(Z,z_3)$
a_{13}
a_{23}
Domain of variable $a(Z,z_3)$
$a(Y,y_2)$
a_{12}
a_{22}
Assignment $a(Y,y_2)=a_{22}$ & & $a(Z,z_3)=a_{13}$ is not allowed

Fig. 1. A part of the constraint model

Demonstration. Consider the following situation. Let us have state variable functions x,y and z and corresponding state variables $X\in\{x_1,...,x_{nx}\}$; $Y\in\{y_1,...,y_{ny}\}$ and $Z\in\{z_1,...,z_{nz}\}$. Next we have several actions. Since the particular preconditions of these actions are not interesting in this demonstration we omit them. The actions are $a_{11}=(\_,X=x_1)$; $a_{12}=(\_,Y=y_2)$; $a_{13}=(\_,Z=z_3)$; $a_{21}=(\_,X=x_1)$; $a_{22}=(\_,Y=y_2)$; $a_{23}=(\_,Z=z_3)$. Consider now that actions a_{11}, a_{12} and a_{13} are pair wise mutex. And the same holds for actions a_{21}, a_{22} and a_{23}. A part of the constraint model of the described situation is depicted in figure 1. Now consider an action b that requires the assignment $[X=x_1, Y=y_2, Z=z_3]$ as its precondition. The action b is supported by actions a_{11}, a_{12}, a_{13}, a_{21}, a_{22} and a_{23} with respect to the standard mutex propagation. Hence the action b can be added to the next action layer of the planning graph. But the action b is not supported with respect to our model since the part of the constraint model represented by the variables $a(X,x_1)$; $a(Y,y_2)$ and $a(Z,z_3)$ is not singleton arc-consistent. Hence the action b cannot be added into the next layer of the constraint model. ■

All the ingredients for building of a new algorithm are ready at this moment. It should be clear now that the proposed constraint model and methods for its expansion can be directly utilized for proposing a variant of the Graphplan algorithm. Figure 2 shows the revised algorithm for planning graph expansion.

```
ExpandModel(M,m) {M is the constraint model of size m}
1   let $G_m=(V_m,E_m)$
2   $V_m \leftarrow \emptyset$, $E_m \leftarrow \emptyset$
3   for each action b=(precond(b),effect(b)) do
4     let precond(b)=($(F_{j1}=d_{k1})$;...;$(F_{j1}=d_{k1})$)
5     c=$(a(F_{j1},d_{k1})\neq\bot)$ &...& $a(F_{j1},d_{k1})\neq\bot)$
6     if $M\cup\{c\}$ is singleton arc-consistent then
7       add a vertex $(b,F_j,d_k)$ to $V_m$ for
8       every $(F_j,d_k)\in effect(b)$
9   for each pair of vertexes $(a_{i1},F_{j1},d_{k1});(a_{i2},F_{j2},d_{k2})\in V_m$
10     where $a_{i1}$ and $a_{i2}$ are dependent do
11       add an edge $\{(a_{i1},F_{j1},d_{k1});(a_{i2},F_{j2},d_{k2})\}$ to $E_m$
12  extend the model M according to the graph $G_m=(V_m,E_m)$
13  perform the mutex propagation in the model
```

Fig. 2. Revised planning graph expansion

The algorithm is described by using the high level steps. This concerns mainly the last two lines of the algorithm. The auxiliary graph G_m is translated into the variables and constraints of the model at line 12. The mutex propagation with respect to the model is performed at line 13. Both steps are described in details in the previous sections.

Checking of the singleton arc-consistency in the model is stronger than the mutex propagation of the standard planning graph expansion. Since that our model represents a more accurate approximation of the relation of reachability of states (the intuitive notion of reachability of states can be replaced by the strong-reachability).

For simplicity reasons the singleton arc-consistency *is checked* only in the algorithm. This simple checking may be time consuming. The more effective way how to extend the model is to *maintain* singleton arc-consistency in the model along the whole resolution process.

8 Conclusions

The paper describes the work in progress which the goal is to provide more accurate approximation of the state reachability in planning problems. Such state reachability analysis is done within one of the most successful planning algorithm Graphplan and its variants. Therefore we consider the techniques for determining reachability of states worth further studying. The Graphplan algorithm builds the structure of planning graph which is then used to answer the question of state reachability. We propose a constraint programming approach to the reachability analysis. More specifically we refine the building of the planning graph by the techniques of constraint programming. Contrary to other works about this topic we concentrate on expansion of the planning graph. We propose more accurate conditions for expansions of the planning graphs than the rules used in the standard planning graph expansion. This leads into a more accurate approximation of state reachability property and to tighter necessary condition on plan existence.

We propose a new method for planning graphs expansion in this paper. The method is based on the theory developed in this paper. We theoretically proved that the proposed method for planning graph expansion covers the standard expansion. More specifically mutex propagation used in standard planning graph expansion is preserved in our approach. Moreover we theoretically proved that our planning graph expansion is stronger - forbids some situations that are allowed by standard method.

As we said the work is in progress. The important step which is now missing is empirical analysis of the proposed method. We expect that the standard Graphplan and our method have different costs of operations. Therefore an empirical analysis is necessary for conclusion that the method represents some improvement (or not).

Finally let us note that our method has an algorithm for checking/maintaining singleton arc-consistency as a module. We intend to compare arc-consistency algorithms as well as the singleton arc-consistency algorithms. The open question is whether it would have some benefit to use algorithms for enforcing higher level consistencies (such algorithms generate new constraints forbidding tuples of values).

References

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Hsu, A. Jónson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. Chafin, W. Dias and P. Maldague. *MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission.* IEEE Intelligent Systems 19(1), 8-12, IEEE Press, 2004.

[2] J. Allen, J. Hendler and A. Tate (editors). *Readings in Planning.* Morgan Kaufmann Publishers, 1990.

[3] M. Baioletti, S. Marcugini and A. Milani: *An Extension of SATPLAN for Planning with Constraints.* In Proceedings of 8th International Conference AIMSA (AIMSA-98), 39-49, LNCS 1480, Springer-Verlag, 1998.

[4] R. Barták and R. Erben. *A New Algorithm for Singleton Arc Consistency.* In Proceedings of the 17th Florida Artificial Intelligence Research Society Conference (FLAIRS-2004), AAAI Press, 2004.

[5] D. Bernard, E. Gamble, N. Rouquette, B. Smith, Y. Tung, N. Muscettola, G. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayak and K. Rajan. *Remote Agent Experiment. Deep Space 1 echnology Validation Report.* NASA Ames and JPL report, 1998.

[6] C. Bessière and R. Debruyne. *Optimal and Suboptimal Singleton Arc Consistency Algorithms.* In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005), 54-59, Professional Book Center, 2005.

[7] A. Blum and M. L. Furst. *Fast Planning through Planning Graph Analysis.* Artificial Intelligence 90(1-2), 281-300, AAAI Press, 1997.

[8] R. Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, 2003.

[9] R. Dechter and A. Dechter. *Belief Maintenance in Dynamic Constraint Networks.* In Proceedings the 7th National Conference on Artificial Intelligence (AAAI-88), 37-42, AAAI Press, 1988.

[10] M. B. Do and S. Kambhampati. Sol*ving Planning-graph by Compiling it into CSP.* In Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-2000), 82-91, AAAI Press, 2000.

[11] M. Ghallab, D. S. Nau and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers, 2004.

[12] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[13] S. Kambhampati. *Planning Graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP Search Techniques in Graphplan.* Journal of Artificial Intelligence Research 12 (JAIR 12), 1-34, AAAI Press, 2000.

[14] S. Kambhampati, E. Parker and E. Lambrecht. *Understanding and Extending Graphplan.* In Proceedings of 4th European Conference on Planning (ECP-97), 260-272, LNCS 1348, Springer-Verlag, 1997.

[15] H. A. Kautz, D. A. McAllester and B. Selman. *Encoding Plans in Propositional Logic.* In Proceedings of the 5th Conference on Principles of Knowledge Representation and Reasoning (KR-96), 374-384, Morgan Kaufmann Publishers, 1996.

[16] H. A. Kautz and B. Selman. *Unifying SAT-based and Graph-based Planning.* In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 318-325, Morgan Kaufmann Publishers, 1999.

[17] H. A. Kautz and B. Selman. *Planning as Satisfiability.* In Proceeding of 10th European Conference on Artificial Intelligence (ECAI-92), 359-363, John Wiley and Sons, 1992.

[18] H. A. Kautz and B. Selman. *Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search.* In Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), 1194-1201, AAAI Press, 1996.

[19] A. Lopez and F. Bacchus. *Generalizing Graphplan by Formulating Planning as a CSP.* In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), 954-960, Morgan Kaufmann Publishers, 2003.

[20] A. K. Mackworth. *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118, AAAI Press, 1977.

[21] A. Nareyek, E. C. Freuder, R. Fourer, E. Giunchiglia, R. P. Goldman, H. A. Kautz, J. Rintanen and A. Tate. *Constraints and AI Planning.* IEEE Intelligent Systems 20(2), 62-72, IEEE Press, 2005.

[22] D. S. Nau , W. C. Regli and K. S. Gupta. *AI Planning versus Manufacturing Operation Planning: A Case Study.* In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), 1670-1676, Morgan Kaufmann Publishers, 1995.

[23] P. Van Beek, X. Chen. *CPlan: A Constraint Programming Approach to Planning.* In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 585-590, AAAI Press, 1999.