

Refinement-based *Planning As Satisfiability**

Amol D. Mali, Subbarao Kambhampati

Dept. of computer science & Engg.
Arizona state university, Tempe, AZ 85287-5406.
{amol.mali, rao}@asu.edu

Abstract

It has been shown recently that planning problems are easier to solve when they are cast as model finding problems. Some schemes for automated generation of the encodings of the planning problems in propositional logic have been designed. However these schemes lack several of the refinements that traditional split & prune type planners do. We show that it is possible to transfer these refinements into the encodings. Since no single encoding has been shown to have the smallest size and the best performance, it is necessary to know what the space of the encodings is. Knowing this space makes a more flexible and efficient design of encodings possible. We show how refinements can be used to generate one such space of encodings. We examine this space and provide the asymptotic sizes of the encodings. This inclusion of refinements into the encodings results in more flexible encodings, some of which are smaller. Our work bridges the gap between the previous research in planning (planning as search) and the recent exciting developments (planning as model finding) and shows how satisfiability-based planning can be integrated with refinement planning.

1 Introduction

A classical planning problem is the problem of computing a sequence of actions that transforms one state of the world into the desired. A classical planning problem is specified as a 3-tuple $\langle I, G, A \rangle$, where I is completely specified initial state of the world, G is partly specified goal state, and A is a set of actions. Each action has preconditions and effects. An action can be applied in a world state only if its preconditions are satisfied by the world state. The effects of the action are guaranteed to be true in the new world state resulting from application of the action, as per classical planning assumption. Classical planning research has made a number of assumptions like static and perfectly observable environment, deterministic actions and perfect perception. However, this type of planning is use-

ful in many real world domains in spite of the simplifying assumptions.

Traditional planners cast planning as a search problem. Starting with the initial state of the world, they apply the actions whose preconditions are true in I and compute new state of the world. This procedure can be repeated till a state of the world in which the goal is true is reached. This process can be carried out starting from the goal as well. This style of planning is termed as state space planning. Since a total order is imposed on the plan steps (each step is mapped to an action), state space algorithms are said to have high commitment to the position of an action in the plan. There are planning algorithms that search in the space of plans rather than states. They impose partial order on the plan steps. These planners are also called least commitment or plan space or causal link planners, e.g. SNLP [McAllester & Rosenblitt 91]. They lack the notion of state of a world. Any step sequence that leads from I to G and is consistent with the precedence orderings and some other constraints is a valid plan. The performance of both state space and plan space planners can be improved with the use of heuristics. There are planners that interleave the state space and plan space refinements like the universal classical planner [Kambhampati & Srivastava 95]. These planners create different branches for different extensions of an incomplete plan, and hence are also termed as “split & prune” planners. These planners are also called refinement planners since they start with a null plan and refine it by adding constraints to it.

Since solving a planning problem means finding a sequence of actions that is executable from an initial state and leads to goal state, one can cast this problem as a deductive theorem proving problem. One has to prove that there exists a sequence of steps $[a_1 a_2 a_3 \dots a_n]$ such that $(I \Rightarrow prec(a_1))$ and $Result(a_1 a_2 a_3 \dots a_n, I) \Rightarrow G$, where $prec(a_1)$ denotes the preconditions of step a_1 , ..., $Result(a_1 a_2 \dots a_{n-1}, I) \Rightarrow prec(a_n)$. $Result(a_1 a_2 a_3 \dots a_i, I)$, denotes the situation prevailing

To appear in the working notes of the AIPS-98 workshop “Planning As Combinatorial Search.”

after the execution of the step sequence $[a_1 a_2 a_3 \dots a_i]$ from I . These constraints can be converted into a set of clauses and a formula can be generated, for a given planning problem $\langle I, G, A \rangle$. There will be a solution of n or less steps for the given planning problem if and only if there exists a model of this formula. We refer this formula in conjunctive normal form (CNF), as a *plan encoding*. [Kautz & Selman 96] reported very impressive results by casting planning as satisfiability.

Various types of planning algorithms lead to various types of encodings. State space encodings of planning problems have been tried in previous literature [Kautz & Selman 96]. Plan space (causal) encodings have been proposed in [Kautz, McAllester & Selman 96]. Different encodings generally have different number of clauses and variables. It is believed that these factors affect the hardness of solving an encoding. Though it is desirable to have an encoding that is as small as possible (has fewer clauses and fewer variables), the smallest encodings in [Kautz, McAllester & Selman 96] are not the easiest ones to solve. This points out the importance of generating and exploring a space of the encodings. This motivates our work. There are a number of properties of split & prune refinement planners that the current plan encodings lack. The plan refinements can proceed either in the forward direction (from I) or in the backward direction (from G) or both directions. Also, state space and plan space refinements can both be carried out. One can also do pre-processing to prune information irrelevant to the planning problems and then generate smaller plan encodings (however we do not consider pre-processing to be a type of plan refinement). These dimensions of refinement planning show that many more plan encodings exist in the rich space in Figure 1. Encodings E1, E2 there use both forward and backward refinements that are causal or state space, and also use the outcomes of pre-processing like operator graphs, macro-operators.

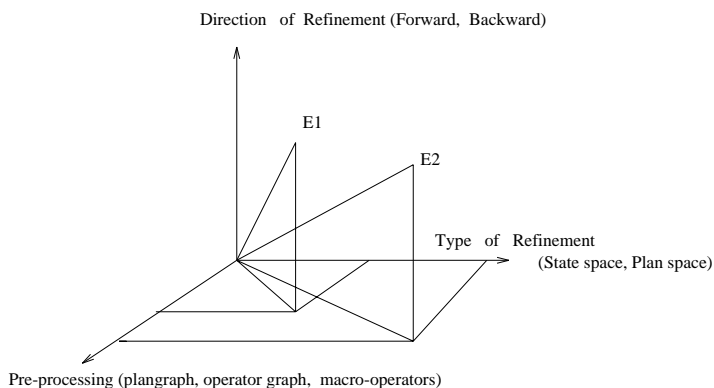


Figure 1: A Space of Plan Encodings

We provide procedures for the generation of these encodings and examine their sizes. In section 2, we explain the notation used in the encodings. We discuss variations of the state space encodings (sec. 3). We discuss a hybrid encoding that combines both state space and plan space refinements (sec. 4). We discuss various pre-processing strategies and evaluate their impact on the sizes of the encodings (sec. 5) and present conclusions in section 6.

2 Notation

We explain the notation here. A primitive action is denoted by o_i . $o_i(j)$ denotes that the action o_i occurs at time j . For this to happen, the preconditions of o_i must be true at time j . p_i denotes a step in the plan space encoding. k denotes the total number of plan steps in an encoding. We assume that there are $|O|$ ground primitive actions in the domain, O being the set of these actions. We assume that a primitive action $o_i, 1 \leq i \leq |O|$, has A_i add effects, D_i delete effects and R_i preconditions. We call our steps and actions primitive since we do not deal with hierarchical task networks.

The add effects of an action o_i are denoted by $a_{i1}, a_{i2}, \dots, a_{iA_i}$, the delete effects are denoted by $d_{i1}, d_{i2}, \dots, d_{iD_i}$ and the preconditions are denoted by $n_{i1}, n_{i2}, \dots, n_{iR_i}$. These are all propositions. The encoding schemes generate an encoding that should be solved to get a k step plan. We require no-ops here, since the plan may have fewer than k actions. No-ops (denoted by ϕ) are null actions with null preconditions and null effects. $\phi(i)$ denotes that the null action ϕ occurs at time i . “ \prec ” denotes temporal precedence relation and “ $p_i \xrightarrow{f} p_j$ ” denotes a causal link from step p_i to step p_j . This means that step p_i has the add effect f , p_j has f in the list of its preconditions and $p_i \prec p_j$. p_i, p_j are also called contributor and consumer respectively. This link is said to be threatened when there is a step p_s which can come between p_i and p_j and has f in the list of its *delete* effects. The initial state I is assumed to be $(l_1 \wedge l_2 \wedge l_3 \dots \wedge l_{|I|})$. The goal state is assumed to be $(a_1 \wedge a_2 \wedge \dots \wedge a_h)$. Γ denotes the set of propositions in the domain. γ_j denotes an element of the set Γ . $\gamma_j(t) \rightarrow p_i$ denotes a causal link where γ_j is true at time t and γ_j is also a precondition of the step p_i . The step p_i occurs after time t . Ground encodings use instantiated actions (e.g. $move(A, B, C)$), as opposed to variablized actions (e.g. $move(?x, ?y, ?z)$).

3 State Space Encodings

[Kautz, McAllester & Selman 96] report encodings based on the traditional state space planning. There are two categories of these encodings - linear and parallel. In linear encodings, only one action occurs at a time step. In parallel encodings, multiple actions can

3.

$$\wedge_{i=0}^{k-1} ((\vee_{j_1=1}^{|\mathcal{O}|} ((\wedge_{j_2=1}^{R_{j_1}} n_{j_1 j_2}(i) \Rightarrow o_{j_1}(i))) \vee \phi(i)),$$

$$\wedge_{i=0}^{k-1} \wedge_{j_1=1}^{|\mathcal{O}|} \wedge_{j_2=1}^{R_{j_1}} (\neg n_{j_1 j_2}(i) \Rightarrow \neg o_{j_1}(i))$$

Figure 2: Schemas for Linear Forward Encoding

occur at the same time step, if they do not interfere. These encodings however, are not sensitive to the direction of refinement. Our encodings in this section are based on the direction of refinement.

3.1 Linear Forward Encoding

This encoding is based on forward state space refinement. Formal versions of the important schemas for generating this encoding have been shown in Figure 2. Schema 1 says that some action has to occur at every time step. Schema 2 says that 2 or more actions cannot occur at the same time step. Schema 3 says that if the preconditions of an action are true at a time, that action or null action can occur at that time (note that we do not say that the action will occur at that time). It also says that if some precondition of an action is false at a time, the action cannot occur at that time. This captures the key idea in forward state space refinement. The corresponding linear encoding schema of [Kautz, McAllester & Selman 96] states that if an action occurs at time t , the preconditions must be true at t . Schema 4 states that if an action occurs at time t , the effects of the action hold at time $(t+1)$. Schema 5 represents the classical frame axioms - if a proposition is true at time t and a no-op or an action not deleting that proposition occurs at time t , the proposition holds at time $t+1$. Schema 6 says that every proposition not implied by I is false at time 0. Schema 7 states that the goal must be true at time k .

Linear forward encoding contains $O(k* | \mathcal{O} |^2 + k* | \mathcal{O} | * | \Gamma |)$ clauses (from schemas 2, 5) and $O(k* | \Gamma | + k* | \mathcal{O} |)$ variables (from schemas 1, 5).

3.2 Linear Backward Encoding

This encoding is based on the backward state space refinement. The formal versions of the important schemas of this encoding are shown in Figure 3. Schema 5 says that each literal in goal is either true at time $(k-1)$ or added by an action occurring at time $(k-1)$. Schema 6 says that if an action occurs at time t , its preconditions are true at time t . Schema 7 says that if an action occurs at time t , its effects hold at time $(t+1)$. Schema 8 says that a proposition true at time t was either true at time $(t-1)$ or an action adding it occurred at $(t-1)$. This encoding contains $O(k* | \mathcal{O} |^2 + k* | \Gamma |)$ clauses and $O(k* | \Gamma | + k* | \mathcal{O} |)$

5.

$$\wedge_{i=1}^k (a_i(k) \Rightarrow ((\vee_{j=1, Adds(o_j, a_i)}^{|\mathcal{O}|} o_j(k-1)) \vee a_i(k-1)))$$

6.

$$\wedge_{i=k-1}^0 \wedge_{j_1=1}^{|\mathcal{O}|} (o_{j_1}(i) \Rightarrow (\wedge_{j_2=1}^{R_{j_1}} n_{j_1 j_2}(i)))$$

7.

$$\wedge_{i=k-1}^0 \wedge_{j=1}^{|\mathcal{O}|} (o_j(i) \Rightarrow ((\wedge_{j_1=1}^{A_j} a_{j j_1}(i+1)) \wedge (\wedge_{j_2=1}^{D_j} \neg d_{j j_2}(i+1))))$$

8.

$$\wedge_{i=k-1}^1 \wedge_{j=1}^{|\Gamma|} (\gamma_j(i) \Rightarrow (\gamma_j(i-1) \vee (\vee_{q=1, Adds(o_q, f_j)}^{|\mathcal{O}|} o_q(i-1))))$$

Figure 3: Schemas for Linear Backward Encoding

variables. The classical frame axioms are not needed because of schema 8. Hence linear backward encoding has fewer clauses than linear forward encoding. We do not need to say that a proposition that is true at time steps t and $(t+1)$ remains so because no action deleting it occurs at t . The reason is - if a proposition is true at time $(t+1)$ and an action deleting the proposition occurs at t , the encoding will not have a model.

3.3 Linear Bidirectional Encoding

This encoding is based on state space refinements that can be backward or forward. This encoding, like other encodings, is formed for k step plans, where $k = (k_1 + k_2)$. The part of the encoding for the first k_1 steps is based on the linear forward encoding schemas, and the remaining k_2 step part is based on the linear backward encoding schemas. This encoding contains $O(k* | \mathcal{O} |^2 + k_1* | \mathcal{O} | * | \Gamma | + k_2* | \Gamma |)$ clauses and $O(k* | \Gamma | + k* | \mathcal{O} |)$ variables. The initial state in the problem specification is the initial state for the forward encoding and the goal state in the problem specification is the goal state in the backward encoding. The state of the world after k_1 steps serves as the “intermediate” goal state for the forward encoding and also as an “intermediate” initial state for the backward encoding. Hence schemas for forward and backward encoding need to be combined with care, to derive the schemas for generating the bidirectional encoding. The state of the world after k_1 steps depends upon the actions that occur at the first k_1 steps. This state is not known a priori and it is not necessary to know it, since the bidirectional encoding will not have a model if that intermediate state cannot lead to goal state after applying a sequence of k_2 actions.

3.4 Parallel Encodings

In parallel encodings, the restriction that only one action occurs at each time step (linear encodings have this restriction) is removed. If two actions interfere (if

one action adds a proposition and other action deletes it, or one action deletes the precondition of another action), they cannot occur at the same time step. If 2 actions do not interfere, and their preconditions are true at a time t , they can occur at t . If actions o_i and o_j occur at the same time and o_j deletes some *add* effect of o_i , the encoding will not have a model, since a proposition cannot both be true and false. Hence no explicit consideration for this type of interference is required. However, we need to consider the interference where one action deletes precondition of other action. The parallel encoding can have 3 versions, like the linear encodings - forward, backward and bidirectional. These versions can be derived from the schemas for linear forward, linear backward and linear bidirectional encodings respectively by replacing the restriction

$$\bigwedge_{i=0}^{k-1} \bigwedge_{j_1=1}^{|O|} \bigwedge_{j_2=1, j_1 \neq j_2}^{|O|} \neg(o_{j_1}(i) \wedge o_{j_2}(i))$$

by

$$\bigwedge_{i=0}^{k-1} \bigwedge_{j_1=1}^{|O|} \bigwedge_{j_3=1}^{R_{j_1}} \bigwedge_{j_2=1, j_2 \neq j_1, \text{Dels}(o_{j_2}, n_{j_1 j_3})}^{|O|} \neg(o_{j_1}(i) \wedge o_{j_2}(i))$$

The asymptotic number of clauses and variables in the three versions of the parallel encoding are same as those in the respective versions of linear encodings.

The encodings are a purely declarative representation. To enforce directionality, we need to assign weights to different clauses to bias the solver to follow a particular order while satisfying clauses responsible for adding an action or a causal link. Choosing weights that strictly monotonically decrease in the direction of refinement fulfills the constraint above.

4 Hybrid Encoding

This encoding is inspired by the idea that a planner can do both state space and plan space refinements. A node in the search space of traditional planners is an incomplete plan that can be refined further in multiple ways. Many times, it is desirable to choose different types of refinements for different nodes, based on the branching factor (which shows the number of possible refinements).

The hybrid encoding has two parts. The first part for k_1 steps is generated using linear encoding schemas and the remaining k_2 step part is generated using ground causal encoding schemas of [Kautz, McAllester & Selman 96]. The encoding has $k = (k_1 + k_2)$ steps. The formal schemas are shown in Figure 4. We explain them here. Each step in causal part succeeds each step in the linear part. The world state after the first k_1 steps serves as the initial state for the next k_2 steps. As argued in the discussion on linear bidirectional encoding, an absence of this knowledge does not affect the soundness and completeness of the encodings.

Schemas 1,...,6 are similar to the ones in the linear encodings. Schemas 7, 8, ..., 18 apply to the causal part of the encoding with k_2 steps. These schemas can be used to generate a purely causal encoding for an arbitrary number of steps by making some minor variations to them. The schemas for causal part here are a variation of the schemas for purely causal encoding of [Kautz, McAllester & Selman 96] and we do this variation to make the causal component compatible with the linear component of the encoding. Schema 7 says that each plan step in the causal part of the encoding can be mapped to any of the available actions, including the no-ops. Schema 8 states that a plan step cannot be mapped to more than one action. Schema 9 says that if a step needs a precondition, there should be a causal link that supports the precondition. The goal state G can be viewed as a step F that has preconditions equal to G and no effects. Every other step has to come before F . Schema 10 says this. A step cannot precede itself. This is captured in schema 11. The precedence relation on steps is transitive. This is stated in schema 12. It also eliminates cycles of length two. Schemas 13,14,15 say that if a step adds, needs or deletes a proposition, then the step is mapped to an action which adds, needs or deletes that proposition respectively. If a step in the causal part threatens a causal link, then the step should be ordered to come before the contributor or after the consumer (demotion or promotion). If a literal in goal is established by the linear part, then no step in causal part should delete it. This is captured in schema 16. Schema 17 says that for each precondition of goal step F , there should be a causal link that supports the precondition. Schema 18 says that in a causal link, the contributor adds literal that the consumer needs and the contributor precedes the consumer. This encoding contains $O(k * |O|^2 + k_1 * |O| * |\Gamma| + k_2^3 * |\Gamma|)$ clauses and $O(k_1 * |\Gamma| + k * |O| + k_2^2 * |\Gamma|)$ variables.

Note that some weighting scheme will be needed to enforce directionality in the encoding. We consider the refinements based on the nature of step ordering (total, partial) and the direction to be of orthogonal nature. The direction of refinement affects the branching factor in the search space and the nature of step ordering affects the knowledge of the world state (complete knowledge of the world state is available in forward state space planning. Causal planning is devoid of this knowledge). Let us consider the variation forward causal planning. In this type of planning, causal links are added in the forward direction, beginning from the initial state. These link the initial state to steps that are mapped to actions whose preconditions are satisfied in the initial state. Then causal links are created

between these steps and the new steps.

3.

$$\bigwedge_{i=0}^{k_1-1} \bigwedge_{j_1=1}^{|\mathcal{O}|} (o_{j_1}(i) \Rightarrow (\bigwedge_{j_2=1}^{R_i} n_{j_1 j_2}(i)))$$

7.

$$\bigwedge_{i=k_1}^{k-1} ((\bigvee_{j=1}^{|\mathcal{O}|} (p_i = o_j)) \vee (p_i = \phi))$$

9.

$$\bigwedge_{i=k_1}^{k-1} \bigwedge_{j=1}^{|\Gamma|} (Needs(p_i, \gamma_j) \Rightarrow (\bigvee_{s=k_1, s \neq i}^{k-1} (p_s \xrightarrow{\gamma_j} p_i) \vee (\gamma_j(k_1) \rightarrow p_i)))$$

16.

$$\begin{aligned} \bigwedge_{i=k_1}^{k-1} \bigwedge_{j=k_1}^{k-1} \bigwedge_{i \neq j} \bigwedge_{q=k_1, q \neq i, q \neq j} \bigwedge_{s=1}^{|\Gamma|} ((p_i \xrightarrow{\gamma_s} p_j \wedge DelS(p_q, \gamma_s)) \Rightarrow \\ ((p_q \prec p_i) \vee (p_j \prec p_q))), \end{aligned}$$

$$\bigwedge_{i=k_1}^{k-1} \bigwedge_{j=k_1}^{k-1} \bigwedge_{i \neq j} \bigwedge_{s=1}^{|\Gamma|} (((\gamma_s(k_1) \rightarrow p_i) \wedge DelS(p_j, \gamma_s)) \Rightarrow (p_i \prec p_j)),$$

$$\bigwedge_{i=1}^h ((a_i(k_1) \rightarrow F) \Rightarrow (\bigwedge_{j=k_1}^{k-1} \neg DelS(p_j, a_i)))$$

$$\bigwedge_{i=k_1}^{k-1} \bigwedge_{j=k_1}^{k-1} \bigwedge_{i \neq j} \bigwedge_{s=1}^h ((p_i \xrightarrow{a_s} F \wedge DelS(p_j, a_s)) \Rightarrow (p_j \prec p_i))$$

$$17. \quad \bigwedge_{j=1}^h ((\bigvee_{s=k_1}^{k-1} (p_s \xrightarrow{a_j} F)) \vee (a_j(k_1) \rightarrow F)),$$

$$\bigwedge_{i=k_1}^{k-1} \bigwedge_{j=1}^h (p_i \xrightarrow{a_j} F \Rightarrow (AddS(p_i, a_j) \wedge (p_i \prec F)))$$

$$18. \quad \bigwedge_{i=k_1}^{k-1} \bigwedge_{j=k_1}^{k-1} \bigwedge_{i \neq j} \bigwedge_{s=1}^{|\Gamma|} ((p_i \xrightarrow{\gamma_s} p_j) \Rightarrow$$

$$(AddS(p_i, \gamma_s) \wedge Needs(p_j, \gamma_s) \wedge (p_i \prec p_j))),$$

$$\bigwedge_{i=k_1}^{k-1} \bigwedge_{j=1}^{|\Gamma|} ((\gamma_j(k_1) \rightarrow p_i) \Rightarrow Needs(p_i, \gamma_j))$$

Figure 4. Schemas for Hybrid Encoding.

5 Pre-processing Strategies

Not all actions in a domain are always relevant to the given planning problem. For example, if packets are to be transported using *fly*, *load*, *unload* actions, then *fly* actions that fly the planes between cities that are neither start nor destination locations of any packets are irrelevant to the problem. Knowing this a priori does reduce the search space and allows the planner to focus on more relevant actions. Also, if a planning problem is unsolvable, pre-processing can detect that. We discuss the popular pre-processing strategies below.

5.1 Operator Graphs

[Smith & Peot 96] propose a representation called *operator graphs*, which contain all actions relevant to a planning problem (the term operator is used to refer to a variablized action, however, the operator graphs can be formed for ground actions as well). These are constructed in the following manner - all actions

that add some literal in goal are considered to be relevant. Then all actions that support the preconditions of the already relevant actions are considered to be relevant. This process is repeated until no more relevant actions are found. Then the relevant actions are processed to remove the actions whose preconditions are unsupported. The effects of this removal are propagated. Let us say that the number of actions appearing in an operator graph is $|O'|$, where O' is the set of actions in the operator graph. Then $|O'| \leq |O|$ always holds. Since fewer actions are relevant, we have fewer preconditions relevant, hence $|\Gamma'| \leq |\Gamma|$, where Γ' is the set of propositions from the preconditions and effects of actions in O' . The use of operator graphs then reduces the number of clauses and variables in linear forward encoding by $O(k * (|O|^2 - |O'|^2) + k * (|O| * |\Gamma| - |O'| * |\Gamma'|))$ and $O(k * (|O| - |O'|) + k * (|\Gamma| - |\Gamma'|))$ respectively.

5.2 Plan Graph

[Blum & Furst 95] reported a planner that applied actions in parallel and propagated constraints to find a valid plan. This planner did not introduce the traditional planners' branching in its search space. The resulting structure that contains actions, propositions and causal links is called *plan graph*. This structure can be used to know which operators are irrelevant to the planning problem. Let the set of actions in a *plan graph* be denoted by O' , $|O'| \leq |O|$. A parallel state space encoding can be generated, based on this structure (one such encoding is proposed in [Kautz, McAllester & Selman 96]). This encoding refers to fewer actions and fewer propositions and has a smaller size.

5.3 Occurrence Bounds

In many cases, it is possible to tell that certain types of actions will not be required more than a certain number of times. For example, if packets are to be transported with trucks and trucks run out of fuel after each one way trip and there is no action to re-fuel a truck, one knows that the action of a truck's travel can occur at the most once in a plan. In all the encodings, we have said that any action can occur at any time instant. This leads to $k * |O|$ variables capturing step-action mapping and $O(k * |O|^2)$ clauses (when we say that a step p_i cannot be mapped to more than one action). If we know that an action o_i will be needed at the most m_i times, then we know that the total number of steps in the encoding should be no more than

$$\sum_{i=1}^{|\mathcal{O}|} m_i$$

since we can map the steps to actions a priori, leaving

for the planner the job of ordering them. This changes the the number of variables from $k * |O|$ to

$$\sum_{i=1}^{|O|} m_i$$

If

$$\sum_{i=1}^{|O|} m_i < k * |O|,$$

the encoding size can be proved to be lower. The number of clauses may be reduced, even if this inequality does not hold. We will still need the no-ops, but we will not be required to say that a step cannot be mapped to two or more non null actions, since we make the step-action mappings a priori, to eliminate this need. Hence we do not need the $O(k * |O|^2)$ clauses. These improvements are applicable to the causal encodings.

5.4 Macro Operators

Many times, planners are used to solve certain types of problems. In these problems, it is possible to sequence and group primitive actions into macro-actions. This reduces the search space of the planner significantly, since the combinations of macro-actions are fewer than the combinations of the primitive actions. For example, the actions of loading a packet, flying a plane and unloading the packet can be chunked in this order to derive the macro-action *transport*. If we have $\alpha * |O|$ macro-operators, $0 < \alpha < 1$, then the number of clauses in linear forward encoding reduces by $O(k * |O|^2 * (1 - \alpha^2) + k * |O| * (|\Gamma| - \alpha * |\Gamma'|))$ and the number of variables reduces by $O(k * |O| * (1 - \alpha) + k * (|\Gamma| - |\Gamma'|))$ where Γ' is the set of propositions in preconditions and effects of macro-operators. This is based on the assumption that only the macro-operators are used in the encoding and the actual solution containing only primitive actions is derived by replacing the macro-operators in the model of the encoding by the sequences of primitive actions.

6 Conclusion

We have shown that integrating the refinements into the encodings of planning leads to a new space of plan encodings. This space shows that many kinds of unexplored encodings exist. Looking at the space of encodings leads to another question - what kinds of refinements can co-exist? Does applying more refinements yield smaller encodings? A critical look shows that one can optimize an encoding by applying the two types of refinements (order based (total order and partial order), direction-based) and pre-processing simultaneously. Though combining order-based and direction-based refinements preserves soundness and completeness, the use of macro-operators may not

maintain these properties (when the problem cannot be solved using macro-operators and one needs sequences of primitive actions that the use of macro-operators prohibits). Though those combinations of the refinements that improve the asymptotic size of an encoding are more promising, given the fact that smallest encodings are not the easiest to solve, it becomes necessary to evaluate other combinations as well. The schemas for generating encodings anywhere in the space in Figure 1, can be derived from the schemas for the base cases. We have shown how this can be achieved. It is necessary to generate and tune the encodings with greater flexibility. Our research has provided a structured framework for this exploration. This also cross-fertilizes the ideas from traditional planning (split & prune type refinements) and recently developed planners (that propagate constraints on a more compact representation).

References

- [Blum & Furst 95] Avrim Blum and Merrick Furst, Fast planning via planning graph analysis, Proc. of IJCAI-95.
- [Kambhampati & Srivastava 95] Subbarao Kambhampati and Biplav Srivastava, Universal classical planner: An algorithm for unifying state space and plan space planning, Preprints of the EWSP, 81-94.
- [Kautz, McAllester & Selman 96] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, Proc. of KRR-96.
- [Kautz & Selman 96] Henry Kautz and Bart Selman, Pushing the envelope: Planning, Propositional logic and Stochastic search, Proc. of AAAI-96.
- [McAllester & Rosenblitt 91] D. McAllester and D. Rosenblitt, Systematic non-linear planning, Proc. of AAAI-91, 634-639.
- [Smith & Peot 96] David Smith and Mark Peot, Suspending recursion in causal-link planning, Proc. of AIPS-96, 182-190.