# Logic Programs and Classical Planning with LTL Constraints

**Ravi Palla**

Advisor: J Benton

## Abstract

The paper deals with encoding planning problems with **LTL** constraints as disjunctive logic programs (**LP** ) and computing the plans (answer sets) under the stable model semantics. In order to encode the problem as a logic program, we use a well known SAT encoding of classical planning problems. Observing that obtaining the **LP** encoding for the **LTL** constraints results in an exponential blowup in the worst case, we provide compact **LP** encoding for the constraints by introducing new predicates.

## Introduction

Logic Programs are useful constructs for non-monotonic reasoning. Combinatorial search problems can be solved using answer set solvers by reducing the problem into a problem of finding the answer sets. The state of the art answer set solvers are quite efficient in grounding and finding answer sets. In this paper, we describe a way to encode planning problems with **LTL** constraints as disjunctive logic programs and compute the plans (answer sets) under the stable model semantics. In order to encode the problem as a logic program, we use a well known SAT encoding for classical planning problems. We also provide an algorithm for finding step optimal plans for problems with **LTL** constraints. The algorithm uses the PG with binary mutex propogation for finding the approximate lowest level at which a valid plan can exist. Observing that obtaining the **LP** encodings of the **LTL** constraints might result in an exponential blowup, we define compact encodings for the constraints in which case new predicates need to be introduced. In this paper, we consider only problems with non-nested **LTL** modalities defined in PDDL 3.0. For simplicity, we separate the constraints from the rest of the planning problem which is referred to as domain description. Throughout the paper, we use the terms step, level and time point interchangeably. The motivation for the **LP** encoding that we have defined in this paper comes from the following factors

1. The grounding module in the answer set solvers uses various techniques in order to avoid generating irrelevant ground instances of rules. [1] So, generating encodings with a time variable can facilitate the use of answer set solvers to generate the ground instances.
2. The relation between the answer sets of a special class of logic programs and the models of the corresponding SAT encoding. The answer sets of disjunctive logic programs without negation are exactly the minimal models of the corresponding SAT encoding. This implies that in the plans returned by the answer set solvers, the number of redundant actions, in general, would be less than those in the plans obtained using SAT encoding.
3. The use of SAT encoding principles to obtain the **LP** encodings thereby maintaining the soundness and completeness property.
4. Any logic program can be turned, in linear time, to a program whose answer sets coincide with the models. For such programs, some answer set solvers simply ground the programs and use SAT solvers to compute the models.

We start with a brief introduction of disjunctive logic programs followed by the **LP** encoding for the domain description and the **LTL** constraints. Later, we briefly discuss about safe logic programs and the use of planning graphs in the context of the paper. In the following sections, we give an algorithm to find step optimal plans followed by the soundness and completeness results for the algorithm. Finally, we provide complexity results for obtaining the **LP** encoding of the domain description followed by compact encodings for the constraints, discuss briefly about SAT based answer set solvers and the grounding module in answer set solvers before concluding.

## Disjunctive Logic Programs

**Definition 1.** *A disjunctive logic program(DLP) consists of rules of the form*

$$A_1; A_2; \ldots; A_m :- B_1, B_2, \ldots, B_k, not\, B_{k+1}, \ldots, not\, B_n \quad (1)$$

*where $A_i$ and $B_i$ are atoms.*

In the above rule ( 1), $A_1; A_2; ...; A_m$ is called the *head* and $B_1, B_2, ..., B_k, not\ B_{k+1}, ..., not\ B_n$ is called the *body*.

[1] One such technique can be found in section 4.3 of www.tcs.hut.fi/Software/smodels/lparse.ps.

**Definition 2.** *A* $constraint$ *in a disjunctive logic program is a rule of the form ( 1) where the head is empty.*

Intuitively, a constraint indicates that any answer set should not satisfy the body.

**Definition 3.** *A DLP rule without negation is a rule of the form ( 1) such that* $n = k$*.*

A $DLP^p$ program is a program consisting of DLP rules without negation.

## LP encoding of the domain description

The **LP** encoding of the domain description is obtained as follows:

1. Obtain the SAT encoding of the domain description.
2. Turning the SAT encoding into a $DLP^p$ program.

### SAT encoding of the domain description

(Kautz et al. 1996) proposes the following principles to obtain the SAT encoding of the domain description:

1. The complete initial state holds at level 0 and the end goals hold at the highest level.
2. Actions at level $i$ imply their pre-conditions at level $i$ and effects at level $i+1$.
3. Conflicting actions are mutually exclusive.
4. Explanatory frame axioms for all fluents.

We will use the SAT based encoding obtained by following the above principles in order to obtain the **LP** encoding.

### Turning the SAT encoding of the domain description to a $DLP^p$ program

The SAT encoding of the domain description can be straight-forwardly turned to a $DLP^p$ program. The only difference in the SAT encoding that we use below and the SAT encoding defined in (Kautz et al. 1996) is that the SAT encoding we use is not grounded with respect to the time point. This minor modification is necessary in order to take advantage of the grounding module in the answer set solvers.

Consider the SAT based encoding for level $n$ obtained using the conditions in the above sub-section:

1. The complete initial state holds at level 0 and the end goals hold at the highest level ($n$). Each fact is a clause here. The only change needed to turn the clauses to the form of a $DLP^p$ rule is to rewrite every negative literal $\neg f$ as ": $-$ $f$".
2. Actions imply their pre-conditions and effects. For any action $a$, this condition can be written as
$$a(T) \rightarrow preconds(T) \wedge effects(T+1)^2.$$

[2] Note that $T$ for actions ranges from 0 to $n-1$ if we are obtaining the encodings with respect to time point (or level) $n$

   Every such implication can be rewritten as a $DLP^p$ rule by first writing the implication as multiple implications in order to eliminate the conjunction in the consequent and then deleting all the negative literals in the consequent of any implication and writing them as conjunctive positive literals in the antecedent.[3] [4]
3. Conflicting actions are mutually exclusive. If $a_1$ and $a_2$ are any two conflicting actions, this condition can be written as
$$\neg a_1(T) \vee \neg a_2(T).$$
   Each such clause can be rewritten as the $DLP^p$ rule
$$:-\ a_1(T) \wedge a_2(T).$$
4. Explanatory frame axioms for all fluents. For any fluent $f$, these axioms can be written as
$$f(T) \wedge \neg f(T+1) \rightarrow \bigvee_{a:\ a\ deletes\ f} a(T)$$
   and
$$\neg f(T) \wedge f(T+1) \rightarrow \bigvee_{a:\ a\ adds\ f} a(T).$$
   These implications can be rewritten as $DLP^p$ rules by just deleting the negative literals in the antecedent and writing them as disjunctive positive literals in the consequent.

**Example 1.** *Consider a simple domain where in there are three locations A,B and C and the problem is to find a plan to go from A to C.*
*This problem can be represented as:*
*(define (domain plan-route)*
*(:predicates (at ?x)*
*(:action drive*
*:parameters (?x ?y)*
*:preconditions (and (at ?x) (not (= ?x ?y)))*
*:effects (and (at ?y) (not (at ?x)))))*

*(define (problem plan-route-A-C)*
*(:objects A B C)*
*(:init (at A))*
*(:goal (at C)))*

*Encoding the domain description as a logic program with respect to time 2, we get:*

- *Initial and Goal states:*
$$at(A,0).$$
$$at(C,2).$$
$$:-at(B,0).$$
$$:-at(C,0).$$

[3] Transforming the resulting implications to the form of a $DLP^p$ rule can be done by turning $\rightarrow$ to $:-$ and replacing $\wedge$ with "," and $\vee$ with ";".

[4] Strictly speaking, due to the presence of the variable T, we should be talking in terms of atomic formulae with possible negation in front of it instead of literals.

- *Actions imply pre-conditions and effects. Writing these conditions as $DLP^p$ rules results in rules of the form:*

$$\begin{aligned} at(A,T) &:- drive(A,B,T). \\ at(B,T+1) &:- drive(A,B,T). \\ &:- drive(A,B,T), at(A,T+1). \end{aligned}$$

- *Conflicting actions are mutually exclusive:*
  *In this domain, each pair of actions are mutually exclusive. So turning these constraints to $DLP^p$ rules, we get rules of the form*

$$:- drive(A,B,T), drive(A,C,T).$$

- *Frame Axioms for all fluents. Turning these axioms into $DLP^p$ rules results in rules of the form:*

$$\begin{aligned} at(A,T+1); drive(A,B,T); drive(A,C,T) &:- at(A,T). \\ at(A,T); drive(B,A,T); drive(C,A,T) &:- at(A,T+1). \end{aligned}$$

*The complete $DLP^p$ program thus obtained has 3 answer sets which correspond to all minimal plans consisting of 2 or less steps:*
$\{at(A,0), drive(A,B,0), at(B,1), drive(B,C,1), at(C,2)\}$.
$\{at(A,0), drive(A,C,0), at(C,1), at(C,2)\}$.
$\{at(A,0), at(A,1), drive(A,C,1), at(C,2)\}$.

## LP Encoding of LTL Constraints

In this section, we define a conversion for **LTL** constraints to disjunctive logic programs. By **LTL** formula, we mean the PDDL 3.0 **LTL** modal operator followed by propositional formulas. It is easy to see that any **LTL** constraint can be turned into an **LTL** formula.
Turning **LTL** constraints to $DLP^p$ rules with respect to level (or time point) $n$ consists of the following steps:

1. Transform the **LTL** formula into a propositional formula with respect to time point $n$.
2. Transform the resulting propositional formula into a set of $DLP^p$ rules.

Given any **LTL** formula $G$, we define an approach to encode the formula as a propositional formula with respect to a time point $n$ as follows:

- **always** $F$ is turned to

$$F(0) \wedge F(1) \wedge \cdots \wedge F(n).$$

- **sometime** $F$ is turned to

$$F(0) \vee F(1) \vee \cdots \vee F(n).$$

- **within** $t$ $F$ is turned to

$$\bigvee_{i\,=\,0\ to\ min(t,n)} F(i).$$

- **atmost-once** $F$ is turned to

$$\bigwedge_{\{i\,=\,0\ to\ n\}} \bigwedge_{\{j\,=\,min(i+1,n)\ to\ n\}} (F(i) \wedge \neg F(j) \rightarrow \bigwedge_{k>j} \neg F(k)).^{5}$$

- **sometime-after** $F$ $H$.
  This is equivalently written as

$$\bigwedge_{i\,=\,0\ to\ n} (F(i) \rightarrow \bigvee_{j\,=\,i\ to\ n} H(j)).$$

- **sometime-before** $F$ $H$.
  This is equivalent to

$$\bigwedge_{i\,=\,1\ to\ n} (F(i) \rightarrow \bigvee_{j\,=\,0\ to\ i-1} H(j)).$$

- **always-within** $t$ $F$ $H$.
  This is equivalent to

$$\bigwedge_{i=0\ to\ n} (F(i) \rightarrow \bigvee_{j=i\ to\ min(i+t,n)} H(t)).$$

- **at-end** $F$ is turned to $F(n)$.

The resulting propositional formula can be turned into a set of $DLP^p$ rules as follows:

1. Convert the formula to $CNF$.
2. Convert every clause in the $CNF$ to the form of a $DLP^p$ rule.

Any clause in a $CNF$ of a formula can be converted to a $DLP^p$ rule as follows:

1. Convert the clause to implication form such that the antecedent is a (possibly empty) conjunction of atoms and the consequent is a (possibly empty) disjunction of atoms.
2. Replace $\rightarrow$ with $:-$, $\wedge$ with ',' and $\vee$ with ';'.[6]

**Example 2.** *Consider the blocks world domain where in a block is fragile and the constraint being that any fragile block can never have another block on it.*

*The constraint can be represented as:*
*(:constraints*
*(always (forall (? x) (implies (fragile ?x) (clear ?x))))).*
*Assuming that there are 3 blocks A,B and C, grounding the constraint results in the following* **LTL** *formula:*

*always*
$(fragile(A) \rightarrow clear(A)) \wedge$
$(fragile(B) \rightarrow clear(B)) \wedge$
$(fragile(C) \rightarrow clear(C))$.

*Converting the* **LTL** *formula to a propositional formula with respect to time* 1 *gives us*
$(fragile(A,0) \rightarrow clear(A,0)) \wedge$
$(fragile(B,0) \rightarrow clear(B,0)) \wedge$
$(fragile(C,0) \rightarrow clear(C,0)) \wedge$
$(fragile(A,1) \rightarrow clear(A,1)) \wedge$
$(fragile(B,1) \rightarrow clear(B,1)) \wedge$
$(fragile(C,1) \rightarrow clear(C,1))$.

[5] $k = min(j+1,n)\ to\ n$.

[6] From now on, whenever we talk about converting a formula to $DLP^p$ program, we skip this step.

*Converting the above formula to $CNF$ gives us:*
$(\neg fragile(A, 0) \vee clear(A, 0)) \wedge$
$(\neg fragile(A, 1) \vee clear(A, 1)) \wedge$
$(\neg fragile(B, 0) \vee clear(B, 0)) \wedge$
$(\neg fragile(B, 1) \vee clear(B, 1)) \wedge$
$(\neg fragile(C, 0) \vee clear(C, 0)) \wedge$
$(\neg fragile(C, 1) \vee clear(C, 1))$

*Turning the above $CNF$ to the form of a $DLP^p$ program, we get the following program:*

$$clear(A, 0) : - fragile(A, 0) \tag{2}$$
$$clear(A, 1) : - fragile(A, 1) \tag{3}$$
$$clear(B, 0) : - fragile(B, 0) \tag{4}$$
$$clear(B, 1) : - fragile(B, 1) \tag{5}$$
$$clear(C, 0) : - fragile(C, 0) \tag{6}$$
$$clear(C, 1) : - fragile(C, 1) \tag{7}$$

**Example 3.** *Consider Example 1 but with the additional constraint that at some time the person driving has to be at $B$.*
*This constraint can be represented as:*

*(:constraints*
*(sometime (at B)))).*
*The* **LP** *encoding of the constraint gives us:*

$$at(B, 0) \vee at(B, 1) \vee at(B, 2).$$

*The program resulting from adding the above rule to the program in Example 1 has one answer set which is:*
$\{at(A, 0), drive(A, B, 0), at(B, 1), drive(B, C, 1), at(C, 2)\}$.

## Safe $DLP^p$ Programs

A $DLP^p$ rule is $safe$ if every variable occurring in the rule also occurs in the *body*. A $DLP^p$ program is $safe$ if all its rules are safe. The current answer set solvers take only safe logic programs as input. In the **LP** encoding that we have defined, all the $DLP^p$ rules are safe. [7]

## Planning Graphs (PG)

Planning graphs are data structures that have been proven to be very useful in speeding the search for finding valid plans. Apart from their other uses, PGs with binary mutex propogation provide an useful approximation on the lowest level at which the end goals can be satisfied such that no pair of them are mutually exclusive. If the first level at which all the end goals are satisfied without any pair of them being mutually exclusive is $k$, then there are no valid plans with level less than $k$. Given this, we can be assured that grounding the $DLP^p$ until level (or time point) $k - 1$ will not result in a valid plan. We can, therefore, search for a valid plan starting from time point $k$.

[7]With respect to some answer set solvers, an additional predicate needs to be added in order to specify the range of the time variable $T$

## Algorithm

Given any planning problem $< D, C, b >$, where $D$ refers to the domain description, $C$ refers to the PDDL 3.0 **LTL** constraints and $b$ refers to the bound, Algorithm 1 finds valid step-optimal plans:

**Algorithm 1** $Finding\ plans\ for\ problems\ with$ **LTL** $cons.$

```
Input: $< D, C, b >$ - planning problem with LTL constraints.
Output: A set of plans $P$.
Construct the PG until the end goals are satisfied such that no pair of them are mutually exclusive;
$k \leftarrow$ highest level of the PG;
no-model = $true$;
$m = k$;
while no-model = $true$ and $m \leq b$
    $L_{C_m} \leftarrow$ LP encoding of $C$ for level $m$;
    $L_{D_m} \leftarrow$ LP encoding of $D$ for level $m$;
    $P \leftarrow extractPlans(findAnswerSets(L_{C_m} \cup L_{D_m}))$;
    if $P$ is non-empty
        no-model = $false$;
    else
        $m = m + 1$;
    end if
end while
return $P$;
```

The number of answer sets returned by the solvers depends on the input option.

## Soundness and Completeness Results

### Some properties of a $DLP^p$ program

**Lemma 1.** *The answer sets of any $DLP^p$ program are the minimal models of any corresponding SAT encoding.*

**Proof**. This is a well known property of $DLP^p$ programs. More details can be found in (Gelfond and Lifschitz. 1991).

### Theorem on Soundness and Completeness

**Theorem 1.** *Algorithm 1 terminates with a valid plan or with no plan if there are no valid plans with $b$ or less steps.*

**Proof**.
**Soundness**:
If a corresponding SAT encoding of $L_{C_m} \cup L_{D_m}$ has a model, then the SAT encoding has to have a minimal model. Since every model of any corresponding SAT encoding of $L_{C_m} \cup L_{D_m}$ corresponds to a valid plan, every minimal model of the SAT encoding also corresponds to a valid plan. From Lemma 1, it follows that every answer set of $L_{C_m} \cup L_{D_m}$ corresponds to a valid plan. The plans are clearly step optimal since we search for the plans starting at level $k$ where $k$ is the first level in the PG at which all the end goals are satisfied such that no pair of them are mutually exclusive.
**Completeness**:
It is clear from the algorithm that it terminates without a

valid plan only if there are no valid plans until the given bound $b$.

We can obtain plans up to $b$ steps by setting $m = b$ instead of $m = k$ before the **while** loop. If all the answer sets are computed, all minimal plans up to $b$ steps can be obtained.

## Complexity Results

The following complexity results follow straight-forwardly from the **LP** encoding of the domain description.

**Proposition 1.** *If $n_a$ is the number of actions and $n_f$ is the number of fluents, then the length of the* **LP** *encoding for the domain description $D$ is bounded by $O(n_a^2 + n_a * n_f)$.*

**Proposition 2.** *If the number of clauses and the average number of literals in each clause in a grounded SAT instance of the domain description $D$ with respect to time $m$ are $n_c$ and $n_l$ respectively, the length of the corresponding grounded* **LP** *encoding is bounded by $O(n_c * n_l)$.*

Obtaining the **LP** encoding of the **LTL** constraints might result in an exponential blowup with respect to the size of the **LTL** formulas corresponding to the constraints. This could cause efficiency concerns for problems with long plans. The following section defines a compact translation of the **LTL** constraints using new predicates and special constructs known as choice rules.[8]

## Compact LP encoding of LTL constraints

Given any **LTL** formula $G$, we first transform the maximal propositional formula(s) $F$(and $H$) occurring in $G$ to a disjunctive normal form $F_1 \vee F_2 \vee \cdots \vee F_k$ (and $H_1 \vee H_2 \vee \cdots \vee H_l$). We then turn the resulting **LTL** formula to a set of logic programming rules with respect to time $n$ as follows:

- **always** $F_1 \vee F_2 \vee \cdots \vee F_k$ is turned to the set of rules

$$p(T) \; :- \; F_i(T).\ (for\ all\ i)$$
$$:- \; not\ p(T).$$

[9]

Here, $p$ is the new predicate introduced.

- **sometime** $F_1 \vee F_2 \vee \cdots \vee F_k$ is turned to the set of rules

$$p \; :- \; F_i(T).$$
$$:- \; not\ p.$$

Here, $p$ is the new predicate introduced.

- **within** $t$ $F_1 \vee F_2 \vee \cdots \vee F_k$ is turned to

$$p \; :- \; F_i(T), T \leq t.$$
$$:- \; not\ p.$$

Here, $p$ is the new predicate introduced.

- **atmost-once** $F_1 \vee F_2 \vee \cdots \vee F_k$ is turned to

$$p(T) \; :- \; F_i(T).$$
$$:- \; p(T), not\ p(T_1), p(T_2), T_1 > T, T_2 > T_1.$$

Here, $p$ is the new predicate introduced.

- **sometime-after** $(F_1 \vee F_2 \vee \cdots \vee F_k)$ $(H_1 \vee H_2 \vee \cdots \vee H_l)$. This is turned to

$$p(T) \; :- \; F_i(T).$$
$$q(T) \; :- \; H_j(T_1), p(T), T_1 \geq T.$$
$$:- \; p(T), not\ q(T).$$

Here $p$ and $q$ are the new predicates introduced.

- **sometime-before** $(F_1 \vee F_2 \vee \cdots \vee F_k)$ $(H_1 \vee H_2 \vee \cdots \vee H_l)$. This is turned to

$$p(T) \; :- \; F_i(T).$$
$$q(T) \; :- \; H_j(T_1), p(T), T_1 < T.$$
$$:- \; p(T), not\ q(T).$$

Here $p$ and $q$ are the new predicates introduced.

- **always-within** $t$ $(F_1 \vee F_2 \vee \cdots \vee F_k)$ $(H_1 \vee H_2 \vee \cdots \vee H_l)$. This is turned to

$$p(T) \; :- \; F_i(T).$$
$$q(T) \; :- \; H_j(T_1), p(T), T_1 \geq T, T_1 - T \leq t.$$
$$:- \; p(T), not\ q(T).$$

Here $p$ and $q$ are the new predicates introduced.

- **at-end** $F_1 \vee F_2 \vee \cdots \vee F_k$ is turned to

$$p \; :- \; F_i(n).$$
$$:- \; p.$$

Here $p$ is the new predicate introduced.

Note that even though we represented all new predicates by $p$ and $q$, each new predicate has to be distinct. Finally, we introduce choice rules $\{f(T) : time(T)\}$ and $\{a(T) : time(T)\}$ for each atom $f$ other than the new atoms and each action $a$. [10]

Let the **LP** encoding of the constraints thus obtained be $L'_{C_n}$. The correspondence between $L_{D_n} \cup L'_{C_n}$, where $L_{D_n}$ is the **LP** encoding of the domain $D$ with respect to time point $n$, and the valid plans is given by the following theorem.

**Theorem 2.** *Every answer set of $L_{D_n} \cup L'_{C_n}$ corresponds to a valid plan with $n$ or less steps and vice versa.*

Note that turning a formula to disjunctive normal form takes exponential time in the worst case. If the domain is large, obtaining $F_1 \vee F_2 \vee \cdots \vee F_k$ corresponding to $F$ might be very expensive since the **LTL** constraints can contain variables (before grounding). This expensive operation can be

[8]Choice rule for atom $p$ is represented as $\{p\}$ which stands for $p \vee \neg p$. $\{p(T) : time(T)\}$ stands for $\{p(t)\}$ for all time points $t$.

[9]By $F_i(T)$, we mean the conjunction of corresponding literals with an argument $T$. For Ex: if $F_i = p \wedge \neg q$, then $F_i(T) = p(T), not\ q(T)$

[10]for actions $a$, the range of $T$ is 0 to $n - 1$.

avoided as follows. Consider any **LTL** constraint with variables. This constraint can be viewed as an **LTL** formula where in the modal operator precedes one or more first order formulas. Let any maximal first order formula following the **LTL** modal operator be $F$.

- If all the variables in $F$ are universally quantified, then we can avoid grounding $F$ and directly obtain the disjunctive normal form $F_1 \vee F_2 \vee \cdots \vee F_k$ with variables.
- If grounding cannot be avoided, the disjunctive normal form can be obtained in polynomial time by introducing new predicates. In this case we need to add choice rules $\{p(T) : time(T)\}$ for every such new predicate. However, for the sake of simplicity of the presentation, we skip this step. The results in Theorems 3 and 2 apply even in the presence of this step.

Since choice rules are added for all atoms and actions other than the atoms containing the new predicates, the techniques used for grounding might be ineffective in which case computing the answer sets of $L_{D_n} \cup L'_{C_n}$ might be more expensive than computing the answer sets of the program resulting from turning $L_{D_n}$ to a program without disjunction. [11] So, we propose the following modification of $L_{D_n}$:
Transform every $DLP^p$ rule to a logic programming rule without disjunction by removing all the atoms in the *head* and adding them as conjunctive negative literals in the *body*. Let the resulting **LP** encoding be $L'_{D_n}$.

**Theorem 3.** *$L_{D_n} \cup L'_{C_n}$ and $L'_{D_n} \cup L'_{C_n}$ have the same answer sets.*

The above theorem shows the soundness and completeness of the **LP** encodings $L'_{D_n} \cup L'_{C_n}$. The process used to convert $L_{D_n}$ to $L'_{D_n}$ can be used to convert any (disjunctive)logic program $\Pi$ to a program $\Pi'$ whose answer sets correspond to the models. Given any (disjunctive) logic program $\Pi$, $\Pi'$ can be obtained as follows:

1. Turn every rule $R \in \Pi$ to constraint form, i.e, write it as an equivalent rule $R'$ such that the *head* of $R'$ is empty.
2. Add choice rules $\{p(X) : Domain(X)\}$ for each predicate $p$. Here $Domain$ is a predicate that indicates the range of $X$.

## Grounding using Answer Set Solvers

The state of the art answer set solvers are quite efficient in grounding programs. They use variety of techniques to avoid generating the irrelevant ground instances of the rules. Algorithm 1 starts with the grounded instance of the domain description and uses the answer set solvers to ground the $DLP^p$ program with respect to the time point variable $T$. If there are no valid plans at level (or time point) $k$, the algorithm has to check for plans at time point $k+1$. In order to obtain the grounded instance of the domain description with respect to time point $k+1$, it is not necessary to ground the program corresponding to the domain description starting from time point 0. The grounded instance of the domain description with respect to time point $k+1$ can be obtained as follows:

1. Ground the $DLP^p$ program corresponding to the domain description only for the single time point $k+1$.
2. Merge the resulting grounded instance with the already grounded instance with respect to time point $k$.
3. Remove the facts indicating that the end goals are true at time point $k$ and add facts indicating that the end goals are true at $k+1$.

## SAT based answer set solvers

The performance of the answer set solvers has been improving significantly over the past few years. The problem of computing answer sets can be turned into a SAT problem. This facilitates the use of the state of the art SAT solvers for computing answer sets. Though this approach works well on various classes of problems, turning a logic program to a SAT problem results in an exponential blowup in the worst case. This has led to various answer set solvers like $clasp$[12] and $CMODELS2$ directly employing the successful SAT reasoning techniques and heuristics with promising results.

## Conclusion

In this paper, we have provided an approach to encode classical planning problems with **LTL** constraints as disjunctive logic programs and compute plans using answer set solvers. The **LP** encoding is defined in such a way as to exploit the efficient grounding module in answer set solvers. We also provided an algorithm to compute step optimal plans which can be easily converted to an algorithm that just computes the plans with respect to a given bound. We have showed that obtaining the **LP** encoding of the domain description is polynomial in the number of actions and fluents. However, obtaining the **LP** encoding of the **LTL** constraints might result in an exponential blowup. This is mainly due to necessity of grounding the constraints(with respect to the time point) before turning them into logic programs. We have therefore defined compact **LP** encodings for the **LTL** constraints which do not refer to grounding. However, for these encodings, new predicates are required. Considering the performance of the state of the art answer set solvers, the **LP** encodings we have defined in this paper are worth considering for computing plans. It remains to be seen how the **LP** encodings and answer set solvers perform on the bench mark planning problems.

## Acknowledgements

Many thanks to Joohyung Lee for useful discussions on SAT based answer set solvers.

## Appendix

**Lemma 2.** *Let $F$ be any propositional formula and let $F'$ be obtained from $F$ by replacing one or more negative occurrences of sub-formula $G$ with $p$ where $p$ is a new atom.*

[11]Disjunctive logic programs belong to the complexity class $\Sigma_2^P$ where as logic programs without disjunction are $NP$-complete.

[12]http://www.cs.uni-potsdam.de/clasp/

*$X \rightarrow X \setminus \{p\}$ is the 1-1 correspondence between the answer sets of $F$ and $F' \wedge (G \leftrightarrow p)$.*

**Lemma 3.** *Let $F$ be any propositional formula and let $F'$ be obtained from $F$ by replacing one or more negative occurrences of sub-formula $G$ with $p$ where $p$ is a new atom. $X \rightarrow X \setminus \{p\}$ is the 1-1 correspondence between the answer sets of $F$ and $F' \wedge (G \rightarrow p)$.*

**Proof**. From Lemma 2, it follows that $X \rightarrow X \setminus \{p\}$ is the 1-1 correspondence between the answer sets of $F$ and $F' \wedge (G \leftrightarrow p)$.
Since all occurrences of $p$ in $F$ are negative, from Proposition 5 in (Ferraris. 2005), it follows that
$X \rightarrow X \setminus \{p\}$ is the 1-1 correspondence between the answer sets of $F$ and $F' \wedge (G \rightarrow p)$.

## Proof of Theorem 2

We will show the proof for one constraint only. The rest of the cases are similar. Let $F$ be the propositional formula corresponding to
$L_{D_n} \cup L_{C_n} \cup \{f(T) : time(T)\} \cup \{a(T) : time(T)\}$.[13]

- **always** $F_1 \vee F_2 \vee \cdots \vee F_k$.
  Any propositional formula corresponding to this constraint is classically equivalent to

$$\bigwedge_{T = 0 \; to \; n} \neg\neg(F_1(T) \vee F_2(T) \vee \cdots \vee F_k(T)).$$

  Let

$$(F_1(t) \vee F_2(t) \vee \cdots \vee F_k(t))$$

  for some time point $t$ be replaced with a new atom $p(t)$. Let the resulting formula be $F'$. From Lemma 3, it follows that
  $X \rightarrow X \setminus \{p(t)\}$ is the 1-1 correspondence between the answer sets of $F$ and

$$F' \wedge (F_1(t) \vee F_2(t) \vee \cdots \vee F_k(t) \rightarrow p(t)).$$

  Since the answer sets of $F$ are exactly the models of $F$, $X \rightarrow X \setminus \{p(t)\}$ is the 1-1 correspondence between the models of the SAT encoding corresponding to $L_{D_n} \cup L_{C_n}$ and the answer sets of $F'$.
  The 1-1 correspondence for every subsequent substitution follows straight-forwardly.

This implies that $X \rightarrow X \setminus \{p(t) : p(t) \; is \; a \; new \; atom\}$ is the 1-1 correspondence between the models of the SAT encoding corresponding to $L_{D_n} \cup L_{C_n}$ and the answer sets of $L_{D_n} \cup L'_{C_n}$.
Hence, the result follows.

## Proof of Theorem 3

Follows straight-forwardly from the properties of choice formulas.

[13]Again, for actions, the time range is from 0 to $n - 1$ only.

## References

Henry Kautz, David McAllester, Bart Selman. Encoding Plans in Propositional Logic. In *Proceedings of the fifth international conference on Principles of Knowledge Representation and Reasoning (KR)*, 1996.

Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases In *New Generation Computing*, 1991.

Paolo Ferraris. Answer Sets for Propositional Theories. In *Proceedings of Logic Programming and Non-Monotonic Reasoning (LPNMR)*, 2005.