Algorithms in Backtracking Search behind SAT and ASP

Yuliya Lierler
Introduction

We now turn out attention to search algorithms underlying ASP technology. In particular, we will focus on the techniques employed by answer set solver such as CLASP. Recall that CLASP is only one building block of an answer set system CLINGO that also incorporates grounder called GRINGO. In the scope of this course we ignore the details behind grounders, but note that these are highly nontrivial systems solving a complex and computationally intense task of intelligent instantiation.

The algorithms behind majority answer set solvers fall into group of so called backtracking search algorithms.

Backtracking is a general algorithm for finding all (or some) solutions to some computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate c ("backtracks") as soon as it determines that c cannot possibly be completed to a valid solution. (Wikipedia)

The Davis-Putnam-Logemann-Loveland (DPLL) procedure is a classic example of backtracking search algorithms. DPLL is a method for deciding the satisfiability of propositional logic formula in conjunctive normal form, or, in other words, for solving the propositional satisfiability (SAT) problem. Algorithms used by answer set solvers share a lot in common with DPLL. In this handout, we thus begin by presenting DPLL procedure. We then discuss its extensions suitable for computing answer sets of a program in place.

1 Satisfiability Solving: Davis-Putnam-Logemann-Loveland Procedure

Recall that a literal is an atom or a negated atom. A signature is a set of atoms. Given a propositional formula, the set of atoms occurring in it is considered to be its signature by default. A clause is a disjunction of literals (possibly the empty disjunction ⊥). A formula is said to be in conjunctive normal form (CNF) if it is a conjunction of clauses (possibly the empty conjunction ⊤). The task of deciding whether a CNF formula is satisfiable is called a satisfiability (SAT) problem. Recall that an interpretation/assignment over a signature is a mapping from the elements of the signature to truth values f or t. For example, given formula

\[(p \land q) \lor r\]  \hspace{1cm} (1)
there are 8 interpretations in its signature \{p, q, r\} including the following

<table>
<thead>
<tr>
<th>interpretation</th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_1)</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>(I_2)</td>
<td>f</td>
<td>t</td>
<td>f</td>
</tr>
</tbody>
</table>

A formula is called satisfiable if we can find an interpretation over its signature so that this formula is evaluated to true under this interpretation. (We assume the familiarity with the interpretation functions for the classical logic connectives \(\top, \bot, \neg, \land \text{ and } \lor\). We say that in such case an interpretation satisfies a formula and also call it a model. For instance, interpretation \(I_1\) satisfies formula (1) while \(I_2\) does not. In other words, \(I_1\) is a model of formula (1). Hence this formula is also satisfiable. It is common to identify an interpretation over signature \(\sigma\) with the set of literals and also with the set of atoms in an intuitive way. For instance, the table below presents such a mapping for interpretations \(I_1\) and \(I_2\).

<table>
<thead>
<tr>
<th>interpretation</th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>set of literals</th>
<th>set of atoms w.r.t. (\sigma = {p, q, r})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_1)</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>{\neg p, q, r}</td>
<td>{q, r}</td>
</tr>
<tr>
<td>(I_2)</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>{\neg p, q, \neg r}</td>
<td>{q}</td>
</tr>
</tbody>
</table>

Later in the discourse we frequently use the word interpretation to denote a set of literals.

The Davis-Putnam-Logemann-Loveland (DPLL) procedure is an algorithm for deciding the satisfiability of propositional logic formula in CNF. DPLL also allows to find a satisfying interpretation of a formula if it exists. Enhancements of DPLL form modern SAT solving technology.

1.1 DPLL by means of Pseudocode

For any CNF formula \(F\) and atom \(A\), \(F\mid_A\) stands for the formula obtained from \(F\) by replacing all occurrences of \(A\) by \(\top\) and simplifying the result by removing

- all clauses containing the disjunctive term \(\top\), and
- the disjunctive terms \(\neg \top\) in all remaining clauses.

Similarly, \(F\mid_{\neg A}\) is the result of replacing \(A\) in \(F\) by \(\bot\) and simplifying the result. For instance,

\[(p \lor q \lor \neg r) \land (\neg p \lor r)\mid_{\neg p} = q \lor \neg r.\]

If a CNF formula \(F\) contains a clause that consists of a single literal ("unit clause") then \(F\) can be simplified using the procedure called unit propagation (Figure 1). In this procedure, \(U\) is a set of literals that does not contain complementary pairs \(A, \neg A\). To apply unit propagation to a given CNF formula \(F_0\), UNIT-PROPAGATE is invoked with \(F = F_0\) and \(U = \emptyset\). After every

**Unit-Propagate**(*F*, \(U\))

\[
\text{while } F \text{ contains no empty clause but has a unit clause } L
\]

\[
F \leftarrow F\mid_L;
\]

\[
U \leftarrow U \cup \{L\}
\]

end

Figure 1: Unit propagation
execution of the body of the loop, the conjunction of $F$ with the literals $U$ remains equivalent to $F_0$.

For instance, to apply unit propagation to

$$p \land (\neg p \lor q) \land (\neg q \lor r)$$

we invoke Unit-Propagate with this formula as $F$ and with $\emptyset$ as $U$. After the first execution of the body of the loop,

$$F = \neg q \land (\neg q \lor r) \text{ and } U = \{p\};$$

after the second iteration

$$F = \top \text{ and } U = \{p, \neg q\}.$$ 

This computation shows that the given formula is equivalent to $p \land \neg q$.

There are two cases when the process of unit propagation alone is sufficient for solving the satisfiability problem for $F_0$. Consider the values of $F$ and $U$ upon the termination of Unit-Propagate. First, if $F = \top$, as in the example above, then $F_0$ is satisfiable, and a satisfying interpretation can be easily extracted from $U$. Second, if $F$ contains the empty clause then $F_0$ is not satisfiable.

**Problem 1.** Use unit propagation to decide whether the formula

$$p \land (p \lor q) \land (\neg p \lor \neg q) \land (q \lor r) \land (\neg q \lor \neg r)$$

is satisfiable.

The Davis-Putnam-Logemann-Loveland procedure (Figure 2) is an extension of the unit propagation method that can solve the satisfiability problem for any CNF formula. Like Unit-Propagate, it is initially invoked with $F = F_0$ and $U = \emptyset$.

**Example 1.** Consider, for instance, the application of the DPLL procedure to

$$(\neg p \lor q) \land (\neg p \lor r) \land (q \lor r) \land (\neg q \lor \neg r).$$  \hspace{1cm} (2)

First DPLL is called with this formula as $F$ and with $\emptyset$ as $U$ (Call 1). After the call to Unit-Propagate, the values of $F$ and $U$ remain the same. Assume that the literal selected as $L$ is $p$. Now DPLL is called recursively with

$$q \land r \land (q \lor r) \land (\neg q \lor \neg r)$$

---

```
DPLL(F,U)
    Unit-Propagate(F,U);
    if F contains the empty clause then return;
    if $F = \top$ then exit with a model of $U$;
    $L \leftarrow$ a literal containing an atom from $F$;
    DPLL(F|L, U \cup \{L\});
    DPLL(F|\overline{L}, U \cup \{\overline{L}\})
```

**Figure 2:** Davis-Putnam-Logemann-Loveland procedure
as \( F \) and \( \{p\} \) as \( U \) (Call 2). After the call to Unit-propagate, \( F \) turns into the empty clause. Next DPLL is called with
\[
(q \lor r) \land (\neg q \lor \neg r)
\]
as \( F \) and \( \{-p\} \) as \( U \) (Call 3). After the call to Unit-propagate, \( F \) and \( U \) remain the same. Assume that the literal selected as \( L \) is \( q \). Then DPLL is called with \( \neg r \) as \( F \) and \( \{\neg p, q\} \) as \( U \) (Call 4). After the call to Unit-propagate, \( F = \top \) and \( U = \{\neg p, q, \neg r\} \). The computation produces an interpretation satisfying the given formula (in other words, a model of this formula):

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( t )</td>
<td>( f )</td>
</tr>
</tbody>
</table>

**Problem 2.** How would this computation be affected by selecting \( \neg p \) as \( L \) in Call 1? By selecting \( \neg q \) as \( L \) in Call 3?

### 1.2 DPLL by means of Transition Systems

In the previous section we described the DPLL procedure using pseudocode. Here we use a different method to present this algorithm. In particular, we will use a transition system that can be viewed as an abstract representation of the underlying DPLL computation. This transition system captures what “states of computation” are, and what transitions between states are allowed. In this way, a transition system defines a directed graph such that every execution of the DPLL procedure corresponds to a path in this graph. Some edges may correspond to unit propagation steps, some to decision, some to backtracking. Later in the handout, we will follow this approach for describing search algorithms suitable for computing answer sets of a program.

For a set \( \sigma \) of atoms, a record relative to signature \( \sigma \) is a sequence \( M \) of distinct literals over \( \sigma \), some possibly annotated by \( \Delta \), which marks them as decision literals. A state relative to \( \sigma \) is either a distinguished state \( \text{FailState} \) or a record relative to \( \sigma \). For instance, the states relative to a singleton set \( \{p\} \) are

\[
\text{FailState}, \ \emptyset, \ p, \ \neg p, \ p^\Delta, \ \neg p^\Delta, \ p \neg p, \ p^\Delta \neg p, \ p \neg p^\Delta, \ p^\Delta \neg p, \ p \neg q \neg p, \ p \neg q \neg p^\Delta, \ p^\Delta \neg q \neg p, \ p^\Delta \neg q \neg p^\Delta.
\]

Note how a sequence of literals such as \( p \ p \) or \( p \ p^\Delta \) does not form a record. Frequently, we consider \( M \) as a set of literals, ignoring both the annotations and the order among its elements.

For a literal of the form \( a \) we say that \( \neg a \) is its complement, whereas for a literal of the form \( \neg a \), atom \( a \) is its complement.

If neither a literal \( l \) nor its complement, written \( \overline{l} \), occurs in \( M \), then \( l \) is unassigned by \( M \). We say that \( M \) is inconsistent if both an atom \( a \) and its negation \( \neg a \) occur in it. For instance, states \( p^\Delta \neg p \) and \( p \ q \neg p \) are inconsistent. Also both \( q \) and \( \neg q \) are unassigned by state \( p^\Delta \neg p \), whereas both of them are assigned by \( p \ q \neg p \).

If \( C \) is a disjunction (conjunction) of literals then by \( \overline{C} \) we understand the conjunction (disjunction) of the complements of the literals occurring in \( C \). In some situations, we will identify disjunctions and conjunctions of literals with the sets of these literals.

Each CNF formula \( F \) determines its DPLL graph \( \text{DP}_F \). The set of nodes of \( \text{DP}_F \) consists of the states relative to the signature of \( F \). The edges of the graph \( \text{DP}_F \) are specified by four
transition rules:

Unit Propagate: \( M \Rightarrow M \, l \) if \( \begin{cases} C \in F, \ l \in C, \text{ and for every} \\ \text{l'} \in C \text{ so that } l' \neq l, \\ \overline{l} \in M \end{cases} \)

Decide: \( M \Rightarrow M \, l^\Delta \) if \( l \) is unassigned by \( M \)

Fail: \( M \Rightarrow \text{FailState} \) if \( \begin{cases} M \text{ is inconsistent, and} \\ M \text{ contains no decision literals} \end{cases} \)

Backtrack: \( P \, l^\Delta \ Q \Rightarrow P \, l \) if \( \begin{cases} P \, l^\Delta \ Q \text{ is inconsistent, and} \\ Q \text{ contains no decision literals.} \end{cases} \)

A node (state) in the graph is terminal if no edge originates in it. The transition rule Unit Propagate is also often called a propagator/inference rule of DPLL.

The following proposition gathers key properties of the graph \( \text{DP}_F \).

**Proposition 1.** For any CNF formula \( F \),

(a) graph \( \text{DP}_F \) is finite and acyclic,

(b) any terminal state of \( \text{DP}_F \) other than \( \text{FailState} \) is a model of \( F \),

(c) \( \text{FailState} \) is reachable from \( \emptyset \) in \( \text{DP}_F \) if and only if \( F \) is unsatisfiable.

Thus, to decide the satisfiability of a CNF formula \( F \) it is enough to find a path leading from node \( \emptyset \) to a terminal node \( M \). If \( M = \text{FailState} \), \( F \) is unsatisfiable. Otherwise, \( F \) is satisfiable and \( M \) is a model of \( F \).

For instance, let \( F_1 = \{p \lor q, \neg p \lor r\} \). Below we show a path in \( \text{DP}_{F_1} \) with every edge annotated by the name of the transition rule that gives rise to this edge in the graph:

\[
\emptyset \xrightarrow{\text{Decide}} p^\Delta \xrightarrow{\text{Unit Propagate}} p^\Delta r \xrightarrow{\text{Decide}} p^\Delta r q^\Delta. \tag{3}
\]

The state \( p^\Delta r q^\Delta \) is terminal. Thus, Proposition 1(b) asserts that \( F_1 \) is satisfiable and \( \{p, r, q\} \) is a model of \( F_1 \). Another path in \( \text{DP}_{F_1} \) that leads us to concluding that set \( \{p, r, q\} \) is a model of \( F_1 \) follows

\[
\emptyset \xrightarrow{\text{Decide}} p^\Delta \xrightarrow{\text{Decide}} p^\Delta r^\Delta \xrightarrow{\text{Decide}} p^\Delta r^\Delta q^\Delta. \tag{4}
\]

We can view a path in the graph \( \text{DP}_F \) as a description of a process of search for a model of a formula \( F \) by applying transition rules of the graph. Therefore, we can characterize an algorithm of a SAT solver that utilizes the inference rules of \( \text{DP}_F \) by describing a strategy for choosing a path in \( \text{DP}_F \). A strategy can be based, in particular, on assigning priorities to some or all transition rules of \( \text{DP}_F \), so that a solver will never apply a transition rule in a state if a rule with higher priority is applicable to the same state. The DPLL algorithm can be captured by the following priorities:

Backtrack, Fail >> Unit Propagate >> Decide.

Note how path (5) in the graph \( \text{DP}_{F_1} \) respects priorities above, while path (4) does not. Thus DPLL will never explore the latter search trajectory given input \( F_1 \).
Problem 3. Let $G$ be formula (2). Then a pass in $\text{DP}_G$ that can be seen as capturing the computation of DPLL described in Example 1 follows:

\[
\begin{align*}
\emptyset & \xrightarrow{\text{Decide}} p^\Delta \\
p^\Delta & \xrightarrow{\text{Unit Propagate}} -q \\
q & \xrightarrow{\text{Backtrack}} -p \\
p^\Delta & \xrightarrow{\text{Unit Propagate}} -q \\
p & \xrightarrow{\text{Unit Propagate}} -p \\
q & \xrightarrow{\text{Unit Propagate}} -p \\
(q & \xrightarrow{\text{Decide}} q p^\Delta)
\end{align*}
\]

(a) List an alternative path to (5) in $\text{DP}_G$ that also can be seen as capturing the computation of DPLL. (Hint: think of nondeterminism in Unit-Propagate procedure.)

(b) Consider node $q$ in graph $\text{DP}_G$. List all the edges that leave this node in $\text{DP}_G$. Annotate these edges by transition rules that they are due. Specify nodes to which these edges lead. For instance,

\[
q \xrightarrow{\text{Decide}} q p^\Delta
\]

is one of these edges.

(c) Consider node $p^\Delta q r \neg q$ in graph $\text{DP}_G$. List all the edges that leave this node in $\text{DP}_G$ (as in the previous question).

2 From ASP to SAT

A number of transformations from logic programs under answer set semantics to SAT exist. Given a propositional logic program $\Pi$, there are two kinds of transformations:

- transformations that preserve the vocabulary of $\Pi$ and form a propositional theory $F_\Pi$ that is equivalent to $\Pi$. In other words, models of $\Pi$ and $F_\Pi$ coincide.

- transformations that may contain “new atoms” so that the answer sets for $\Pi$ can be obtained by removing these atoms from the models of constructed $F_\Pi$.

Remarkable transformation of the former kind is called completion. For a large syntactic class of programs (“tight” programs), the models of program’s completion coincide with the answer sets of a program. This fact is exploited in several state-of-the-art answer set solvers including CLASP. For example, for tight programs CLASP practically runs a (significantly enhanced) DPLL procedure on program’s completion to obtain answer sets of a program.

Answer Set Solving. We are now ready to present an extension to the DPLL algorithm that captures a family of backtrack search procedures for finding answer sets of a propositional logic program.

Recall that a propositional logic program is a finite set of rules of the form

\[
a_0 \leftarrow a_1, \ldots, a_k, \text{not } a_{k+1}, \ldots, \text{not } a_m, \text{not not } a_{m+1}, \ldots, \text{not not } a_n,
\]

where $a_0$ is a propositional atom or symbol $\bot$; $a_1, \ldots, a_n$ are propositional atoms. We sometimes identify rule (6) with a clause

\[
a_0 \lor -a_1 \lor \cdots \lor -a_k \lor a_{k+1} \lor \cdots \lor a_m \lor -a_{m+1} \lor \cdots \lor -a_n,
\]

when $a_0$ is an atom; and with a clause

\[
-a_1 \lor \cdots \lor -a_k \lor a_{k+1} \lor \cdots \lor a_m \lor -a_{m+1} \lor \cdots \lor -a_n,
\]

when $a_0$ is $\bot$. 


For a program Π, by \( \sigma_\Pi \) we denote the set of atoms occurring in it. We call \( \sigma_\Pi \) a program’s signature. For a program Π, we call an interpretation \( M \) over \( \sigma_\Pi \) a classical model of Π if it is a model of the set of rules in Π seen as clauses. For example, program

\[
p \\
r \leftarrow p, q
\]

(9)

has three classical models \( \{ p, \neg q, \neg r \} \), \( \{ p, \neg q, r \} \), and \( \{ p, q, r \} \). In a sense, a concept of a classical model generalizes the definition of what does it mean for a set of atoms to satisfy a definite program to arbitrary programs.

By \( \text{Bodies}(\Pi, a) \) we denote the set of the bodies of all rules of program Π with the head \( a \) (including the empty body identified with \( \top \)). A set \( U \) of atoms occurring in a propositional program Π is unfounded on a consistent set \( M \) of literals with respect to Π if for every \( a \in U \) and every \( B \in \text{Bodies}(\Pi, a) \), \( M \cap B \neq \emptyset \) or \( U \cap B^{\text{pos}} \neq \emptyset \), where \( B^{\text{pos}} \) denotes positive part of the body \( B \). For instance, set \( \{ r \} \) is unfounded on set \( \{ p, \neg q, r \} \) with respect to program (9), while set \( \{ q \} \) is unfounded on \( \{ p, q, r \} \) with respect to program (9).

For a set \( M \) of literals, by \( M^+ \) we denote the set composed of all the literals that occur without classical negation in \( M \). E.g., \( \{ p, q, \neg r \}^+ = \{ p, q \} \).

We now state a formal result that relates the notions of an unfounded set and answer sets. This result is crucial for understanding key inference rules used in propagators of modern answer set solvers.

**Proposition 2.** For a program Π and a set \( M \) of literals over \( \sigma_\Pi \), \( M^+ \) is an answer set of Π if and only if \( M \) is a classical model of Π and no non-empty subset of \( M^+ \) is an unfounded set on \( M \) with respect to Π.

This proposition gives an alternative characterization of an answer set. I.e., we may bypass the reference to a reduct in our argument that a set of atoms is an answer set. It is sufficient to verify that (i) this set of atoms corresponds to a classical model of a program and (ii) no non-empty subset of this set is unfounded. For example, this proposition asserts that

- classical models of program (9) are the only interpretations that may correspond to answer sets of (9)
- sets \( \{ p, \neg q, \neg r \} \), \( \{ p, \neg q, r \} \), \( \{ p, q, r \} \) of literals are the classical models of program (9). Thus, sets \( \{ p, \neg q, \neg r \}^+ = \{ p \} \), \( \{ p, \neg q, r \}^+ = \{ p, r \} \), \( \{ p, q, r \}^+ = \{ p, q, r \} \) of atoms form the candidates for being answer sets,
- sets \( \{ p, r \} \) and \( \{ p, q, r \} \) are not answer sets of the program due to unfounded sets \( \{ r \} \) and \( \{ q \} \) respectively. Set \( \{ p \} \) is an answer set (since the only nonempty subset of it, namely, \( \{ p \} \), is not an unfounded set on \( \{ p, \neg q, \neg r \} \) with respect to program (9)).

We define the transition graph \( \text{aset}_\Pi \) for a program Π as follows. The set of nodes of the graph \( \text{aset}_\Pi \) consists of the states relative to atoms occurring in Π. There are five transition rules that characterize the edges of \( \text{aset}_\Pi \). The transition rules \( \text{Unit Propagate}, \text{Decide}, \text{Fail}, \text{Backtrack} \) of the graph \( \text{dp}_\Pi \) (note that here we identify each rule of a program with a clause), and the transition rule

\[
\text{Unfounded: } M \Rightarrow M \neg a \text{ if } \left\{ \begin{array}{l} a \in U \text{ for a set } U \text{ unfounded on } M \text{ with respect to } \Pi. \end{array} \right. 
\]

The graph \( \text{aset}_\Pi \) can be used for deciding whether a logic program has answer sets:
**Proposition 3.** For any program $\Pi$,

(a) graph $\text{aset}_\Pi$ is finite and acyclic,

(b) for any terminal state $M$ of $\text{aset}_\Pi$ other than $\text{FailState}$, $M^+$ is an answer set of $\Pi$,

(c) $\text{FailState}$ is reachable from $\emptyset$ in $\text{aset}_\Pi$ if and only if $\Pi$ has no answer sets.

Extending graph $\text{aset}_\Pi$ with transition rules

- **All Rules Cancelled:** $M \Rightarrow M \neg a$ if $\mathcal{B} \cap M \neq \emptyset$ for all $B \in \text{Bodies}(\Pi, a)$,

- **Backchain True:** $M \Rightarrow M \{l\}$ if
  \[
  \begin{cases}
  a \leftarrow B \in \Pi, \\
  a \in M, \\
  \mathcal{B}' \cap M \neq \emptyset \text{ for all } B' \in \text{Bodies}(\Pi, a) \setminus \{B\}, \\
  l \in B
  \end{cases}
  \]

results in a graph that captures the computation procedure of answer set solver SMODELS (the first tool on ASP market dating to 1999). We denote the resulting graph by $\text{sm}_\Pi$. Similar statement to Proposition 3 holds for the graph $\text{sm}_\Pi$. The system SMODELS assigns priorities to the inference rules of $\text{sm}_\Pi$ as follows:

- **Backtrack, Fail >>**
- **Unit Propagate, All Rules Cancelled, Backchain True >>**
- **Unfounded >>**
- **Decide.**

**A Peek at Important Enhancements of ASP (and SAT) solvers**

The key difference of system CLINGO from the SMODELS algorithm that we presented lays in implementation of such advanced solving techniques as learning and forgetting, backjumping and restarts. Below we provide some intuitions behind these.

The learning technique allows a solver to extend its knowledge base (that originally is composed of a given program) by additional constraints so that certain inferences become readily available in the later states of search via propagation rules (eliminating the need for intermediate applications of decide rules). The forgetting allows the solver to make the learning process dynamic so that sometimes learned constraints are forgotten/removed to eliminate the chance of solver’s knowledge base becoming of a prohibitive size.

Backjumping enhances backtracking mechanism by allowing to identify the decision level different from the last one that is safe to jump to so that (i) no solution is lost and (ii) part of the search space is escaped.

Restarting allows a solver to drop currently searched path and start over again with a hope to make better choices on a new path that lead to a solution quicker.

**Problem 4.** Let $\Pi_1$ be a program

\[
\begin{align*}
& r. \\
& p \leftarrow \text{not } q, r \\
& q \leftarrow \text{not } p, r
\end{align*}
\]

(a) List all classical models of $\Pi_1$.  
(b) List all unfounded sets on set $\{r, p, q\}$ with respect to program $\Pi_1$.  
(c) List all unfounded sets on set $\{r, \neg p, \neg q\}$ with respect to program $\Pi_1$.  

8
(d) List all unfounded sets on set \( \{r, p, \neg q\} \) with respect to program \( \Pi_1 \).

(e) List all answer sets of \( \Pi_1 \).

(f) List some five states in graph \( \text{aset}_{\Pi_1} \).

(g) List some path in \( \text{aset}_{\Pi_1} \) from \( \emptyset \) to state \( r\neg q\Delta p \). Think of another possible path in this graph from \( \emptyset \) to the same state \( r\neg q\Delta p \). List that path. In both cases annotate all the transitions/edges in your path by the names of the respective rules.

(h) Is state \( r\neg q\Delta p \) terminal in the graph \( \text{aset}_{\Pi_1} \)? If so what can you conclude about program \( \Pi_1 \) and state \( r\neg q\Delta p \) given Proposition 3.

Acknowledgments

Parts of this handout follow

- the lecture notes on Logic-based AI course, UT, Spring 2011\(^1\) by Vladimir Lifschitz.


\(^1\)http://www.cs.utexas.edu/~vl/teaching/lbai