Abstract Answer Set Solver

• An abstract framework for describing algorithms to find answer sets of a logic program using “constraint propagation”, backjumping, learning and forgetting.

TodoList

• Print the rules of Fig 1.

Outline

• Notations
• Abstract Answer Set Solver
  – A first definition of graph associated with a program
  – An extended graph (catering for backjump)
  – Answer set solver
• Appendix
  – Generate reasons (in extended records)
  – Generate backjump clause
Abstract Answer Set Solver

• *States* and *transition rules* on *states* will be used, instead of pseudo-code, to describe ASP algorithms employing propagation, backjumping, learning and forgetting

States

• State: M||T or *FailState*
  – M is a record: A record relative to a program P is a list of literals over atoms of P without repetitions where each literal has an annotation, a bit that marks it as a decision literal or not.
  – T is a (multi-)set of denials

Example

| FailState | Φ||Φ | a b||Φ | ¬a₃||Φ | ¬a₄ a||Φ |
| Φ||b | a b||b | ¬a₃ b||b | ¬a₄ a||b |
| Φ||b | a b||b | ¬a₃ b||b | ¬a₄ a||b |

Record

• Record
  – By ignoring the annotations and ordering, a record M can be taken as a set of literals, i.e., a “partial assignment”
  – l is unassigned if neither l nor its complement is in M
  – A decision literal: subscripted with \(\Delta\)
  – Non-decision literal: no subscription
**Transition rules**

- Example

\[
\text{Unit Propagate:} \\
M || \Gamma \implies M a || \Gamma \quad \text{if} \quad \begin{cases} 
a \leftarrow B \in \Pi \text{ and} \\
B \subseteq M \end{cases}
\]

\[
\text{Unfounded:} \\
M || \Gamma \implies M \neg a || \Gamma \quad \text{if} \quad \begin{cases} 
\text{M is consistent and} \\
a \in U \text{ for a set } U \text{ unfounded on M w.r.t. } \Pi
\end{cases}
\]

**Graph associated to a program**

- For any program P, we define a graph G_P whose
  - Nodes are the states of P
  - Edges are “transition rules”
    - If there is a transition rule S → S’ followed by a condition such that S and S’ are states and the condition is satisfied, there is an edge between S and S’ in the graph

**Graph and answer set**

- Transition rules
- Semi-terminal state
- Result

**Transition rules**

- Basic rules
  - Rules based on satisfying the program rules
  - Rules based on unfounded set
  - Backjump (backtrack)
  - Decide
  - Fail
- Rules about learning
- Rules about forgetting
Basic rules

**Unit Propagate:**
\[ M || \Gamma \quad \Rightarrow \quad M \ a || \Gamma \quad if \quad \begin{cases} a \leftarrow B \in \Pi \quad and \quad B \subseteq M \end{cases} \]

**All Rules Cancelled:**
\[ M || \Gamma \quad \Rightarrow \quad M \neg a || \Gamma \quad if \quad B \cap M \neq \emptyset \quad for \quad all \quad B \in Bodies(\Pi, a) \]

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

A clause \( l \lor C \) is a **reason** for \( l \) to be in a list of literals \( P \lor Q \) w.r.t \( P \) if \( P \) satisfies \( l \lor C \) and \( C \subseteq P \).

**Backchain False:**
\[ M || \Gamma \quad \Rightarrow \quad M \overline{a} || \Gamma \quad if \quad \begin{cases} a \leftarrow l, B \in \Pi, \\ \overline{a} \in M \quad or \quad a = \bot, \\ B \subseteq M \end{cases} \]

**Unfounded:**
\[ M || \Gamma \quad \Rightarrow \quad M \neg a || \Gamma \quad if \quad \begin{cases} M \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } M \text{ w.r.t. } \Pi \end{cases} \]

**Decide:**
\[ M || \Gamma \quad \Rightarrow \quad M \ l^\Delta || \Gamma \quad if \quad \begin{cases} M \text{ is consistent and} \\ l \text{ is unassigned by } M \end{cases} \]

**Fail:**
\[ M || \Gamma \quad \Rightarrow \quad FailState \quad if \quad \begin{cases} M \text{ is inconsistent and} \\ M \text{ contains no decision literals} \end{cases} \]

\[ \text{what's } l' \? \]
Rules on learning and forgetting

Learn:
\[ M||\Gamma \Rightarrow M|| \leftarrow B, \Gamma \text{ if } \Pi \text{ satisfies } \overline{B} \]

Forget:
\[ M|| \leftarrow B, \Gamma \Rightarrow M||\Gamma \]

• *Semi-terminal* state: there is no edge due to one of the *basic* transition rules leaving this node.

Graph and Answer sets

• Given a program P and its graph G_P
  – every path in G_P contains only finitely many edges labeled by Basic transition rules,
  – for any semi-terminal state M||\Gamma of G_P reachable from \emptyset||\emptyset, M_+ is an answer set of P,
  – *FailState* is reachable from \emptyset||\emptyset in G_P if and only if P has no answer sets.

Example

\[
\begin{align*}
&a \leftarrow \text{not } b \\
&b \leftarrow \text{not } a \\
&c \leftarrow a \\
&d \leftarrow d.
\end{align*}
\]

\[
\begin{align*}
\emptyset||\emptyset & \Rightarrow \text{Decide) } \\
\alpha\Delta||\emptyset & \Rightarrow \text{Unit Propagate) } \\
\alpha\Delta c||\emptyset & \Rightarrow \text{All Rules Cancelled) } \\
\alpha\Delta c \Rightarrow \overline{b}||\emptyset & \Rightarrow \text{Decide) } \\
\alpha\Delta c \Rightarrow \overline{b} \alpha\Delta d||\emptyset & \Rightarrow \text{Unfounded) } \\
\alpha\Delta c \Rightarrow \overline{b} \alpha\Delta \overline{d}||\emptyset & \Rightarrow \text{Backjump) }
\end{align*}
\]
Extended graph of a program

- Backjump: in contrast to backtrack to the previous decision literal, it can backtrack to the earlier decision literal which "causes" the current conflict. Efficient in SAT solvers
- Learning and forgetting: clauses are learned from the current conflict. They can be used to prune the search space. Forgetting is necessary because too many learned clauses may slow down the solver. Again, very useful techniques in SAT solvers

Extended graph: extended state || denials, or FailState
- An extended record \( M \) relative to a program \( P \) is a list of literals over atoms in \( P \) without repetitions where
  - (i) each literal \( l \) in \( M \) is annotated either by or by a reason for \( l \) to be in \( M \),
  - (ii) for any inconsistent prefix of \( M \) its last literal is annotated by a reason.

Example: extended record

\[
\begin{aligned}
\text{program} & \quad a \leftarrow \neg b \\
\text{an extended state} & \quad b^\Delta, a^\Delta, \neg b \lor \neg \neg a
\end{aligned}
\]

Example: non extended record

\[
\begin{aligned}
a^\Delta & \quad \neg a^\Delta \\
b^\Delta & \quad a^\Delta, \neg b \lor \neg a \\
\neg \neg a^\Delta & \quad \neg b \lor \neg a \\
\end{aligned}
\]
Extended graph

- We now define a graph $G_{\uparrow P}$ for any program $P$. Its nodes are the extended states relative to $P$. The transition rules of $G_P$ are extended to $G_{\uparrow P}$ as follows: $S_1 \rightarrow S_2$ is an edge in $G_{\uparrow P}$ justified by a transition rule $T$ if and only if $S_1 \rightarrow S_2$ is an edge in $G_P$ justified by $T$.

$$S_1^\psi: \text{the state obtained by dropping reasons from } S_1$$

$$S_i^\psi: \text{the state obtained by dropping reasons from } S_i$$

$S_1: a \rightarrow b \rightarrow \nu \sigma \perp \phi$  $S_i: a \rightarrow b \perp \phi$

Proposition 1

- For any program $P$,
  - a) every path in $G_{\uparrow P}$ contains only finitely many edges labeled by Basic transition rules,
  - b) for any semi-terminal state $M \perp \Gamma$ of $G_{\uparrow P}$, $M^+$ is an answer set of $P$,
  - c) $G_{\uparrow P}$ contains an edge leading to $\text{FailState}$ if and only if $P$ has no answer sets.

Note

- Any semi-terminal state and $\text{FailState}$ is reachable from $\phi \parallel \phi$ in $G_{\uparrow P}$?

Answer set solver

- Consider finding only one answer set
- A solver using the same inference rules (unit propagate etc.) as those of $G_P$ (or $G_{\uparrow P}$) can be characterized by its strategies of traversing the graph to find a path from $\phi \parallel \phi$ to a semi-terminal or $\text{FailState}$.

SMODELS_cc

1. edges corresponding to the applications of transition rules $\text{Unit Propagate}$, $\text{All Rules Cancelled}$, $\text{Backchain True}$, $\text{Backchain False}$, and $\text{Unfounded}$ to a state in $G_P$ are considered if $\text{Backjump}$ is not applicable in this state,
2. an edge corresponding to an application of a transition rule $\text{Decide}$ to a state in $G_P$ is considered if and only if none of the rules among $\text{Unit Propagate}$, $\text{All Rules Cancelled}$, $\text{Backchain True}$, $\text{Backchain False}$, $\text{Unfounded}$, and $\text{Backjump}$ is applicable in this state,
3. an edge corresponding to an application of a transition rule $\text{Learn}$ to a state in $G_P$ is considered if and only if this state was reached by the edge $\text{Backjump}$ and a $\text{FirstUIP}$ backjump clause is learned
SUP

1–3 of SMODELS_cc

• an edge corresponding to an application of transition rule Unfounded to a state in $G_P$ is considered only if a state assigns all atoms of $P$

• Remove unfounded from 2.

Generate the reasons

Unit Propagate:

$M \parallel \Gamma \Rightarrow M \ a \parallel \Gamma$ if

\[
\begin{cases}
  a \leftarrow B \in \Pi \quad \text{and} \\
  B \subseteq M
\end{cases}
\]

reason:

$a \lor \overline{B}$ (if $B$ then $a$)

All Rules Cancelled:

$M \parallel \Gamma \Rightarrow M \ \neg a \parallel \Gamma$ if $\overline{B} \cap M \neq \emptyset$ for all $B \in \text{Bodies}(\Pi, a)$

Reason:

$\forall B$, let $f(B)$ be a literal $\in B \cap \overline{M}$

$\neg a \lor \bigvee_{B \in \text{Bodies}(\Pi, a)} f(B)$ (then $\neg a$)

Backchain True:

$M \parallel \Gamma \Rightarrow M \ l \parallel \Gamma$ if

\[
\begin{cases}
  a \leftarrow B \in \Pi, \\
  a \in M, \\
  \overline{B} \cap M \neq \emptyset \quad \text{for all } B' \in \text{Bodies}(\Pi, a) \setminus B, \\
  l \in B
\end{cases}
\]

Reason:

$l \lor \neg a \lor \bigvee_{B' \in \text{Bodies}(\Pi, a) \setminus B} f(B').$

Backchain False:

$M \parallel \Gamma \Rightarrow M \ \neg l \parallel \Gamma$ if

\[
\begin{cases}
  a \leftarrow l, B \in \Pi, \\
  \neg a \in M \quad \text{or} \quad a = \bot, \\
  B \subseteq M
\end{cases}
\]

reason:

$\neg l \lor a \lor \overline{B}$
Rules on learning and forgetting

Unfounded:

\( M \models \Gamma \Rightarrow M \models \neg \alpha \models \Gamma \) if \( M \) is consistent and
\( \alpha \in \Gamma \) for a set \( \Gamma \) unfounded on \( M \) w.r.t. \( \Pi \)

if \( \forall B \in \text{Bodies}(\Pi, U) \) such that \( U \cap B^+ = \emptyset \),
\( B \cap M \neq \emptyset \), then \( \forall a \in U, \neg \alpha \).

\( \neg \alpha \lor \lor f(B) \)

\( B \in \text{Bodies}(\Pi, U), B^+ \cap U = \emptyset \)

Backjump:

\( P \models Q \models \Gamma \Rightarrow P \models \Gamma \) if \( P \models Q \) is inconsistent and
there exists a reason for \( t' \) to be in \( P \) w.r.t. \( \Pi \)

Reason:

latter, we discuss how to find a reason.

Decide:

\( M \models \Gamma \Rightarrow M \models t^{\Delta} \models \Gamma \) if \( M \) is consistent and
\( t \) is unassigned by \( M \)

Fail:

\( M \models \Gamma \Rightarrow \text{FailState} \) if \( M \) is inconsistent and
\( M \) contains no decision literals

Learn:

\( M \models \Gamma \Rightarrow M \models \leftarrow B, \Gamma \) if \( \Pi \) satisfies \( B \)

Forget:

\( M \models \leftarrow B, \Gamma \Rightarrow M \models \Gamma \)
Backjump related notations

- We call the reason in the backjump rule \textit{backjump clause}.

\textit{Backjump:}
\[ P \vdash Q | \Gamma \quad \Rightarrow \quad P | \Gamma' | \Gamma \quad \text{if} \quad \begin{cases} P | \Gamma \text{ is inconsistent and} \\text{there exists a reason for } \Gamma' \text{ to be in } P | \Gamma \text{ w.r.t. } \Pi \end{cases} \]

- We say that a state in the graph \( G \uparrow_P \) is a \textit{backjump state} if its record is inconsistent and contains a decision literal.

- For a record \( M \), by \( \text{lcp}(M) \) we denote its largest consistent prefix.
- A clause \( C \) is \textit{conflicting} on a list \( M \) of literals if \( P \) satisfies \( C \), and \( C \subseteq \text{lcp}(M) \).

\begin{align*}
a & \leftarrow \neg b \\
b & \leftarrow \neg a, \neg c \\
c & \leftarrow \neg f \\
\neg k, \, d &
\end{align*}

\begin{align*}
a & \Delta \quad \neg b \quad \neg b \lor a \\
c & \Delta \quad \neg f \lor \neg c \\
d & \Delta \quad \neg k \lor \neg d \\
\neg l \lor \neg b \lor k & \quad \neg m \lor \neg b \lor m \lor k \lor l
\end{align*}
**CUT**

- What’s a cut
  - A cut in the implication graph is a bipartition of the graph such that all decision variables are in one set while the conflict is in the other set.
  - There are many cuts

- Each cut results in a learned clause

(Decision) backjump clause through graph

- Decision backjump clause: one set of the cut contains only decision variables

Obtain backjump clause through resolution

\[ R_1 = \neg m v l v b \]
\[ R_2 = \neg l v b v k \]
\[ R_3 = \neg k v d \]
\[ R_4 = \neg b v a \]

Decision backjump clause: \( \neg a u v d \)
Apply backjump rule

\[ a \Delta b, b \lor c, c \Delta f, f \lor c, d \Delta k, k \lor l, l \lor v, k, m \lor k \lor l \implies \]
\[ a \lor b, b \lor c, d \lor e, e \lor f, f \lor g, g \lor h, h \lor i, i \lor j, j \lor k, k \lor l, m \lor k \lor l \]

- **What's unique implication point (UIP)**
  - A literal \( l \) in an implication graph is called a *unique implication point* if every path from the decision literal at level \( l \) to the point of conflict passes through \( l \).
  - A *decision level* of a literal \( l \) is the number of decision variables when \( l \) is assigned a value.

**First UIP cut**
- The 1UIP cut of an implication graph is the cut “generated from” the unique implication points closest to the point of conflict.
  - On one set (conflict side): all variables assigned after the first UIP of current decision level reachable to the conflict
  - On the other side: everything else.
Apply backjump rule

\[ a \Delta b \lor \neg a \quad c \Delta f \lor \neg c \quad d \Delta \neg k \lor d \lor \neg l \lor k \lor m \lor m \lor k \lor l \forall \quad a \Delta b \lor \neg a \quad k \lor b \lor \emptyset \]

backjump clause: \( k \lor b \)
decision variables leading to \( \bot \):
\( \langle a, d \rangle \)
so, backjump to the decision level of \( a \).

Application of learning rule

- Reason carried by the last literal can be put into the store

References

1. An abstract answer set solver by Yuliya
2. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver, iccad 2001
Appendix

• Another implication graph

![Implication Graph](image)

Backjump clause (conflict clause)

Clauses

\[ \neg y_1 \lor \neg y_2 \lor \neg y_5 \lor \neg y_6 \lor \neg y_7 \lor \neg y_8 \lor \neg y_9 \lor \neg y_{10} \]

Implication graph (with decision literals x1, x2, x3)

![Implication Graph](image)

Notations

• Unfounded set
  A set \( U \) of atoms occurring in a program \( P \)
  is said to be **unfounded** on a consistent
  set \( M \) of literals w.r.t. \( P \) if for every
  \( a \in U \) and every
  \( B \in \text{Bodies}(P, a), \ B \cap M = \emptyset \) or
  \( U \cap B \neq \emptyset. \)