Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Potassco

Computational Aspects: Overview

1. Consequence operator
2. Computation from first principles
3. Complexity

Outline

Consequence operator

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

$$T_P X = \{head(r) \mid r \in P \text{ and } body(r) \subseteq X\}$$

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$, where
    - $T_P^0 X = X$ and
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$
- For any positive program P, we have
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Potassco

Consequence operator

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

$$T_P X = \{head(r) \mid r \in P \text{ and } body(r) \subseteq X\}$$

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$, where
    - $T_P^0 X = X$ and
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

- For any positive program P, we have
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Potassco

Consequence operator

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

    $$T_P X = \{head(r) \mid r \in P \text{ and } body(r) \subseteq X\}$$

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$, where
    - $T_P^0 X = X$ and
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$
- For any positive program P, we have
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Potassco

An example

- Consider the program

$$P = \{p \leftarrow,\ q \leftarrow,\ r \leftarrow p,\ s \leftarrow q,t,\ t \leftarrow r,\ u \leftarrow v\}$$

- We get

$$\begin{array}{lllllll}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p,q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p,q,r\} & = & T_P T_P^1\emptyset & = & T_P\{p,q\} \\
T_P^3\emptyset & = & \{p,q,r,t\} & = & T_P T_P^2\emptyset & = & T_P\{p,q,r\} \\
T_P^4\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^3\emptyset & = & T_P\{p,q,r,t\} \\
T_P^5\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^4\emptyset & = & T_P\{p,q,r,t,s\} \\
T_P^6\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^5\emptyset & = & T_P\{p,q,r,t,s\}
\end{array}$$

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

An example

- Consider the program

$$P = \{p \leftarrow,\; q \leftarrow,\; r \leftarrow p,\; s \leftarrow q, t,\; t \leftarrow r,\; u \leftarrow v\}$$

- We get

$$\begin{array}{lllllll}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p,q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p,q,r\} & = & T_P T_P^1\emptyset & = & T_P\{p,q\} \\
T_P^3\emptyset & = & \{p,q,r,t\} & = & T_P T_P^2\emptyset & = & T_P\{p,q,r\} \\
T_P^4\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^3\emptyset & = & T_P\{p,q,r,t\} \\
T_P^5\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^4\emptyset & = & T_P\{p,q,r,t,s\} \\
T_P^6\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^5\emptyset & = & T_P\{p,q,r,t,s\}
\end{array}$$

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

Potassco

An example

- Consider the program

$$P = \{p \leftarrow,\ q \leftarrow,\ r \leftarrow p,\ s \leftarrow q, t,\ t \leftarrow r,\ u \leftarrow v\}$$

- We get

$$\begin{array}{lclclcl}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p,q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p,q,r\} & = & T_P T_P^1\emptyset & = & T_P\{p,q\} \\
T_P^3\emptyset & = & \{p,q,r,t\} & = & T_P T_P^2\emptyset & = & T_P\{p,q,r\} \\
T_P^4\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^3\emptyset & = & T_P\{p,q,r,t\} \\
T_P^5\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^4\emptyset & = & T_P\{p,q,r,t,s\} \\
T_P^6\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^5\emptyset & = & T_P\{p,q,r,t,s\}
\end{array}$$

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

Outline

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
    If $X \subseteq U$, then $Cn(P^U) \subseteq X$
    If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$

  If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$
  replace U **by** $U \cap Cn(P^L)$

  until *L and U do not change anymore*

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P

  If $L \not\subseteq U$, then P has no stable model

  If $L = U$, then L is a stable model of P

Potassco

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$
  replace U **by** $U \cap Cn(P^L)$

  until *L and U do not change anymore*

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

Potassco

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$
  replace U **by** $U \cap Cn(P^L)$

  until *L and U do not change anymore*

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

Potassco

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$

  replace U **by** $U \cap Cn(P^L)$

  until *L and U do not change anymore*

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

Potassco

The simplistic expand algorithm

expand$_P(L, U)$
  repeat
    $L' \leftarrow L$
    $U' \leftarrow U$
    $L \leftarrow L' \cup Cn(P^{U'})$
    $U \leftarrow U' \cap Cn(P^{L'})$
    if $L \not\subseteq U$ **then return**
  until $L = L'$ and $U = U'$

An example

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a,b,c,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
2	$\{a\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
3	$\{a,b\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$

- Note We have $\{a,b\} \subseteq X$ and $(\mathcal{A} \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

Potassco

An example

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a,b,c,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
2	$\{a\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
3	$\{a,b\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$

- Note We have $\{a,b\} \subseteq X$ and $(\mathcal{A} \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

Potassco

An example

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a, b, c, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
2	$\{a\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
3	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$

- Note We have $\{a, b\} \subseteq X$ and $(\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

Potassco

The simplistic expand algorithm

- **expand**$_P$
  - tightens the approximation on stable models
  - is stable model preserving

Let's expand with d !

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

- Note $\{a, b, d\}$ is a stable model of P

Let's expand with d !

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a,d\}$	$\{a,b,c,d,e\}$	$\{a,b,d\}$	$\{a,b,d\}$
2	$\{a,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$
3	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$	$\{a,b,d\}$

- Note $\{a, b, d\}$ is a stable model of P

Potassco

Let's expand with d !

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

- Note $\{a, b, d\}$ is a stable model of P

Potassco

Let's expand with $\sim d$!

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

- Note $\{a, b, e\}$ is a stable model of P

Let's expand with $\sim d$!

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

- Note $\{a, b, e\}$ is a stable model of P

Let's expand with $\sim d$!

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$$

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

- Note $\{a, b, e\}$ is a stable model of P

A simplistic solving algorithm

$solve_P(L, U)$

$(L, U) \leftarrow expand_P(L, U)$ // *propagation*
if $L \not\subseteq U$ **then failure** // *failure*
if $L = U$ **then output** L // *success*
else choose $a \in U \setminus L$ // *choice*
  $solve_P(L \cup \{a\}, U)$
  $solve_P(L, U \setminus \{a\})$

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - Heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - Heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - Heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - Heuristic choices are made on atoms

Outline

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P:
  - Deciding whether X is the stable model of P is P-complete
  - Deciding whether a is in the stable model of P is P-complete
- For a normal logic program P:
  - Deciding whether X is a stable model of P is P-complete
  - Deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P:
  - Deciding whether X is the stable model of P is P-complete
  - Deciding whether a is in the stable model of P is P-complete
- For a normal logic program P:
  - Deciding whether X is a stable model of P is P-complete
  - Deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P:
  - Deciding whether X is the stable model of P is P-complete
  - Deciding whether a is in the stable model of P is P-complete
- For a normal logic program P:
  - Deciding whether X is a stable model of P is P-complete
  - Deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P:
  - Deciding whether X is the stable model of P is P-complete
  - Deciding whether a is in the stable model of P is P-complete
- For a normal logic program P:
  - Deciding whether X is a stable model of P is P-complete
  - Deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is co-NP-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive disjunctive logic program P:
  - Deciding whether X is a stable model of P is *co-NP*-complete
  - Deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P:
  - Deciding whether X is a stable model of P is *co-NP*-complete
  - Deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is $co\text{-}NP^{NP}$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_3^p-complete
- For a propositional theory Φ:
  - Deciding whether X is a stable model of Φ is *co-NP*-complete
  - Deciding whether a is in a stable model of Φ is NP^{NP}-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive disjunctive logic program P:
  - Deciding whether X is a stable model of P is *co-NP*-complete
  - Deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P:
  - Deciding whether X is a stable model of P is *co-NP*-complete
  - Deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P with optimization statements:
  - Deciding whether X is an optimal stable model of P is $co\text{-}NP^{NP}$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_3^p-complete
- For a propositional theory Φ:
  - Deciding whether X is a stable model of Φ is *co-NP*-complete
  - Deciding whether a is in a stable model of Φ is NP^{NP}-complete

Potassco

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The nomore++ approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[2] C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
Künstliche Intelligenz, 19(1):12–17, 2005.

[3] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[4] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

Potassco

[5] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[6] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[7] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.
In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[8] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.

In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[9] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.
IOS Press, 2009.

[11] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[12] K. Clark.

Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[13] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[15] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[16] M. Davis and H. Putnam.
A computing procedure for quantification theory.

Journal of the ACM, 7:201–215, 1960.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.
Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.

In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.

Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.
In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

[26] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [44], pages 250–264.

[29] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.

Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[30] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.

[31] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [68], pages 386–392.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[34] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[35] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[36] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.

[37] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[38] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[39] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[40] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[41] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[42] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[43] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.

Journal of Automated Reasoning, 36(4):345–377, 2006.

[44] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[45] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [68], pages 2318–2323.

[46] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[47] J. Lee.
A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[48] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[49] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[50] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[51] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[52] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

[53] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[54] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[55] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [10], chapter 4, pages 131–153.

[56] J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[57] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.

In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[58] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[59] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

[60] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[61] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[62] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[63] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[64] L. Ryan.

Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[65] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[66] T. Syrjänen.
Lparse 1.0 user's manual.

[67] A. Van Gelder, K. Ross, and J. Schlipf.
The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[68] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[69] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.

In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.