Answer Set Solving in Practice

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

Potassco

Conflict-driven ASP Solving: Overview

Outline

Motivation

- Goal Approach to computing stable models of logic programs, based on concepts from
  - Constraint Processing (CP) and
  - Satisfiability Testing (SAT)
- Idea View inferences in ASP as unit propagation on nogoods
- Benefits
  - A uniform constraint-based framework for different kinds of inferences in ASP
  - Advanced techniques from the areas of CP and SAT
  - Highly competitive implementation

Outline

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

  $$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

  $$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potasso

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

$$(\sigma_1, \ldots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

$$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potassco

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

  $$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

  $$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potasso

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

  $$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

  $$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potassco

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

$$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

$$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

  $$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

  $$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potassco

Assignments

- An assignment A over $dom(A) = atom(P) \cup body(P)$ is a sequence

  $$(\sigma_1, \dots, \sigma_n)$$

  of signed literals σ_i of form $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for $v \in dom(A)$ and $1 \leq i \leq n$
- $\boldsymbol{T}v$ expresses that v is *true* and $\boldsymbol{F}v$ that it is *false*
- The complement, $\overline{\sigma}$, of a literal σ is defined as $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$
- $A \circ \sigma$ stands for the result of appending σ to A
- Given $A = (\sigma_1, \dots, \sigma_{k-1}, \sigma_k, \dots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \dots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access *true* and *false* propositions in A via

  $$A^{\boldsymbol{T}} = \{v \in dom(A) \mid \boldsymbol{T}v \in A\} \text{ and } A^{\boldsymbol{F}} = \{v \in dom(A) \mid \boldsymbol{F}v \in A\}$$

Potassco

Nogoods, solutions, and unit propagation

- A nogood is a set $\{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \dots, \sigma_n$
- An assignment A such that $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} = dom(A)$ and $A^{\boldsymbol{T}} \cap A^{\boldsymbol{F}} = \emptyset$ is a solution for a set Δ of nogoods, if $\delta \not\subseteq A$ for all $\delta \in \Delta$
- For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$
- For a set Δ of nogoods and an assignment A, unit propagation is the iterated process of extending A with unit-resulting literals until no further literal is unit-resulting for any nogood in Δ

Nogoods, solutions, and unit propagation

- A nogood is a set $\{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \dots, \sigma_n$
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a solution for a set Δ of nogoods, if $\delta \not\subseteq A$ for all $\delta \in \Delta$
- For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$
- For a set Δ of nogoods and an assignment A, unit propagation is the iterated process of extending A with unit-resulting literals until no further literal is unit-resulting for any nogood in Δ

Nogoods, solutions, and unit propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a solution for a set Δ of nogoods, if $\delta \not\subseteq A$ for all $\delta \in \Delta$
- For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$
- For a set Δ of nogoods and an assignment A, unit propagation is the iterated process of extending A with unit-resulting literals until no further literal is unit-resulting for any nogood in Δ

Nogoods, solutions, and unit propagation

- A nogood is a set $\{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \dots, \sigma_n$
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a solution for a set Δ of nogoods, if $\delta \not\subseteq A$ for all $\delta \in \Delta$
- For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$
- For a set Δ of nogoods and an assignment A, unit propagation is the iterated process of extending A with unit-resulting literals until no further literal is unit-resulting for any nogood in Δ

Outline

Potassco

Outline

Nogoods from logic programs
via program completion

The completion of a logic program P can be defined as follows:

$$\begin{aligned} & \{v_B \leftrightarrow a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n \mid \\ & \qquad B \in \mathit{body}(P) \text{ and } B = \{a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n\}\} \\ \cup \; & \{a \leftrightarrow v_{B_1} \vee \cdots \vee v_{B_k} \mid \\ & \qquad a \in \mathit{atom}(P) \text{ and } \mathit{body}_P(a) = \{B_1, \ldots, B_k\}\} \; , \end{aligned}$$

where $\mathit{body}_P(a) = \{\mathit{body}(r) \mid r \in P \text{ and } \mathit{head}(r) = a\}$

Nogoods from logic programs
via program completion

- The (body-oriented) equivalence

$$v_B \leftrightarrow a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n$$

can be decomposed into two implications:

Nogoods from logic programs
via program completion

- The (body-oriented) equivalence

$$v_B \leftrightarrow a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n$$

can be decomposed into two implications:

1. $v_B \rightarrow a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n$
   is equivalent to the conjunction of

$$\neg v_B \vee a_1, \ \ldots, \ \neg v_B \vee a_m, \ \neg v_B \vee \neg a_{m+1}, \ \ldots, \ \neg v_B \vee \neg a_n$$

   and induces the set of nogoods

$$\Delta(B) = \{ \{ \boldsymbol{T}B, \boldsymbol{F}a_1 \}, \ldots, \{ \boldsymbol{T}B, \boldsymbol{F}a_m \}, \{ \boldsymbol{T}B, \boldsymbol{T}a_{m+1} \}, \ldots, \{ \boldsymbol{T}B, \boldsymbol{T}a_n \} \}$$

Nogoods from logic programs
via program completion

- The (body-oriented) equivalence

$$v_B \leftrightarrow a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n$$

can be decomposed into two implications:

2. $a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n \rightarrow v_B$
   gives rise to the nogood

$$\delta(B) = \{\boldsymbol{F}B, \boldsymbol{T}a_1, \ldots, \boldsymbol{T}a_m, \boldsymbol{F}a_{m+1}, \ldots, \boldsymbol{F}a_n\}$$

Potassco

Nogoods from logic programs
via program completion

- Analogously, the (atom-oriented) equivalence

$$a \leftrightarrow v_{B_1} \vee \cdots \vee v_{B_k}$$

yields the nogoods

1. $\Delta(a) = \{ \{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\} \}$ and
2. $\delta(a) = \{ \boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k \}$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

$\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
$\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

$\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
$\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

$$\begin{array}{|lcl|} \hline x & \leftarrow & y \\ x & \leftarrow & \sim z \\ \hline \end{array} \qquad \begin{array}{l} \{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\} \\ \{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\} \end{array}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal
  - $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
  - $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

$$\begin{array}{lcl} x & \leftarrow & y \\ x & \leftarrow & \sim z \end{array}$$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs

atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Nogoods from logic programs
atom-oriented nogoods

- For an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
atom-oriented nogoods

- For an an atom a where $body_P(a) = \{B_1, \dots, B_k\}$, we get

$$\{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\} \quad \text{and} \quad \{\{\boldsymbol{F}a, \boldsymbol{T}B_1\}, \dots, \{\boldsymbol{F}a, \boldsymbol{T}B_k\}\}$$

- Example Given Atom x with $body(x) = \{\{y\}, \{\sim z\}\}$, we obtain

x	$\leftarrow$	y
x	$\leftarrow$	$\sim z$

$$\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$$
$$\{\{\boldsymbol{F}x, \boldsymbol{T}\{y\}\}, \{\boldsymbol{F}x, \boldsymbol{T}\{\sim z\}\}\}$$

For nogood $\{\boldsymbol{T}x, \boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\}\}$, the signed literal

- $\boldsymbol{F}x$ is unit-resulting wrt assignment $(\boldsymbol{F}\{y\}, \boldsymbol{F}\{\sim z\})$ and
- $\boldsymbol{T}\{\sim z\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}\{y\})$

Potassco

Nogoods from logic programs
body-oriented nogoods

- For a body $B = \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\}$, we get

$$\{\boldsymbol{F}B, \boldsymbol{T}a_1, \dots, \boldsymbol{T}a_m, \boldsymbol{F}a_{m+1}, \dots, \boldsymbol{F}a_n\}$$
$$\{\,\{\boldsymbol{T}B, \boldsymbol{F}a_1\}, \dots, \{\boldsymbol{T}B, \boldsymbol{F}a_m\}, \{\boldsymbol{T}B, \boldsymbol{T}a_{m+1}\}, \dots, \{\boldsymbol{T}B, \boldsymbol{T}a_n\}\,\}$$

- Example Given Body $\{x, \sim y\}$, we obtain

$$\dots \leftarrow x, \sim y$$
$$\vdots$$
$$\dots \leftarrow x, \sim y$$

$$\{\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$$
$$\{\,\{\boldsymbol{T}\{x, \sim y\}, \boldsymbol{F}x\}, \{\boldsymbol{T}\{x, \sim y\}, \boldsymbol{T}y\}\,\}$$

For nogood $\delta(\{x, \sim y\}) = \{\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$, the signed literal
- $\boldsymbol{T}\{x, \sim y\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}y)$ and
- $\boldsymbol{T}y$ is unit-resulting wrt assignment $(\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x)$

Potassco

Nogoods from logic programs
body-oriented nogoods

- For a body $B = \{a_1, \dots, a_m, {\sim}a_{m+1}, \dots, {\sim}a_n\}$, we get

$$\{\boldsymbol{F}B, \boldsymbol{T}a_1, \dots, \boldsymbol{T}a_m, \boldsymbol{F}a_{m+1}, \dots, \boldsymbol{F}a_n\}$$
$$\{\,\{\boldsymbol{T}B, \boldsymbol{F}a_1\}, \dots, \{\boldsymbol{T}B, \boldsymbol{F}a_m\}, \{\boldsymbol{T}B, \boldsymbol{T}a_{m+1}\}, \dots, \{\boldsymbol{T}B, \boldsymbol{T}a_n\}\,\}$$

- Example Given Body $\{x, {\sim}y\}$, we obtain

$$\begin{array}{l} \dots \leftarrow x, {\sim}y \\ \quad\vdots \\ \dots \leftarrow x, {\sim}y \end{array}$$

$$\{\boldsymbol{F}\{x, {\sim}y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$$
$$\{\,\{\boldsymbol{T}\{x, {\sim}y\}, \boldsymbol{F}x\}, \{\boldsymbol{T}\{x, {\sim}y\}, \boldsymbol{T}y\}\,\}$$

For nogood $\delta(\{x, {\sim}y\}) = \{\boldsymbol{F}\{x, {\sim}y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$, the signed literal

- $\boldsymbol{T}\{x, {\sim}y\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}y)$ and
- $\boldsymbol{T}y$ is unit-resulting wrt assignment $(\boldsymbol{F}\{x, {\sim}y\}, \boldsymbol{T}x)$

Nogoods from logic programs
body-oriented nogoods

- For a body $B = \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\}$, we get

$$\{\boldsymbol{F}B, \boldsymbol{T}a_1, \dots, \boldsymbol{T}a_m, \boldsymbol{F}a_{m+1}, \dots, \boldsymbol{F}a_n\}$$
$$\{\,\{\boldsymbol{T}B, \boldsymbol{F}a_1\}, \dots, \{\boldsymbol{T}B, \boldsymbol{F}a_m\}, \{\boldsymbol{T}B, \boldsymbol{T}a_{m+1}\}, \dots, \{\boldsymbol{T}B, \boldsymbol{T}a_n\}\,\}$$

- Example Given Body $\{x, \sim y\}$, we obtain

$$\dots \leftarrow x, \sim y$$
$$\vdots$$
$$\dots \leftarrow x, \sim y$$

$$\{\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$$
$$\{\,\{\boldsymbol{T}\{x, \sim y\}, \boldsymbol{F}x\}, \{\boldsymbol{T}\{x, \sim y\}, \boldsymbol{T}y\}\,\}$$

For nogood $\delta(\{x, \sim y\}) = \{\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x, \boldsymbol{F}y\}$, the signed literal
- $\boldsymbol{T}\{x, \sim y\}$ is unit-resulting wrt assignment $(\boldsymbol{T}x, \boldsymbol{F}y)$ and
- $\boldsymbol{T}y$ is unit-resulting wrt assignment $(\boldsymbol{F}\{x, \sim y\}, \boldsymbol{T}x)$

Characterization of stable models

for tight logic programs

Let P be a logic program and

$$\begin{aligned}\Delta_P &= \{\delta(a) \mid a \in atom(P)\} \cup \{\delta \in \Delta(a) \mid a \in atom(P)\} \\ &\cup \{\delta(B) \mid B \in body(P)\} \cup \{\delta \in \Delta(B) \mid B \in body(P)\}\end{aligned}$$

Theorem

Let P be a tight logic program. Then,
$X \subseteq atom(P)$ is a stable model of P iff
$X = A^{\mathbf{T}} \cap atom(P)$ for a (unique) solution A for Δ_P

Characterization of stable models

for tight logic programs

Let P be a logic program and

$$\begin{array}{rcl}\Delta_P & = & \{\delta(a) \mid a \in atom(P)\} \cup \{\delta \in \Delta(a) \mid a \in atom(P)\} \\ & \cup & \{\delta(B) \mid B \in body(P)\} \cup \{\delta \in \Delta(B) \mid B \in body(P)\}\end{array}$$

Theorem

Let P be a tight logic program. Then,
$X \subseteq atom(P)$ is a stable model of P iff
$X = A^{\boldsymbol{T}} \cap atom(P)$ for a (unique) solution A for Δ_P

Characterization of stable models

for tight logic programs, ie. free of positive recursion

Let P be a logic program and

$$\begin{array}{rcl}\Delta_P & = & \{\delta(a) \mid a \in atom(P)\} \cup \{\delta \in \Delta(a) \mid a \in atom(P)\} \\ & \cup & \{\delta(B) \mid B \in body(P)\} \cup \{\delta \in \Delta(B) \mid B \in body(P)\}\end{array}$$

Theorem

Let P be a tight logic program. Then,
$X \subseteq atom(P)$ is a stable model of P iff
$X = A^{\boldsymbol{T}} \cap atom(P)$ for a (unique) solution A for Δ_P

Outline

1 Motivation

2 Boolean constraints

3 Nogoods from logic programs
- Nogoods from program completion
- Nogoods from loop formulas

4 Conflict-driven nogood learning

Nogoods from logic programs
via loop formulas

Let P be a normal logic program and recall that:

- For $L \subseteq atom(P)$, the external supports of L for P are
$$ES_P(L) = \{r \in P \mid head(r) \in L \text{ and } body(r)^+ \cap L = \emptyset\}$$
- The (disjunctive) loop formula of L for P is
$$LF_P(L) = (\bigvee_{A \in L} A) \rightarrow (\bigvee_{r \in ES_P(L)} body(r))$$
$$\leftrightarrow (\bigwedge_{r \in ES_P(L)} \neg body(r)) \rightarrow (\bigwedge_{A \in L} \neg A)$$
  - Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- The external bodies of L for P are
$$EB_P(L) = \{body(r) \mid r \in ES_P(L)\}$$

Nogoods from logic programs
via loop formulas

Let P be a normal logic program and recall that:

- For $L \subseteq atom(P)$, the external supports of L for P are

$$ES_P(L) \quad = \quad \{r \in P \mid head(r) \in L \text{ and } body(r)^+ \cap L = \emptyset\}$$

- The (disjunctive) loop formula of L for P is

$$\begin{array}{rcl} LF_P(L) & = & (\bigvee_{A \in L} A) \rightarrow (\bigvee_{r \in ES_P(L)} body(r)) \\ & \leftrightarrow & (\bigwedge_{r \in ES_P(L)} \neg body(r)) \rightarrow (\bigwedge_{A \in L} \neg A) \end{array}$$

  - Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported

- The external bodies of L for P are

$$EB_P(L) \quad = \quad \{body(r) \mid r \in ES_P(L)\}$$

Nogoods from logic programs
via loop formulas

Let P be a normal logic program and recall that:

- For $L \subseteq atom(P)$, the external supports of L for P are
  $$ES_P(L) \;=\; \{r \in P \mid head(r) \in L \text{ and } body(r)^+ \cap L = \emptyset\}$$
- The (disjunctive) loop formula of L for P is
  $$\begin{aligned} LF_P(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_P(L)} body(r)\right) \\ &\leftrightarrow \left(\bigwedge_{r \in ES_P(L)} \neg body(r)\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right) \end{aligned}$$
  - Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- The external bodies of L for P are
  $$EB_P(L) \;=\; \{body(r) \mid r \in ES_P(L)\}$$

Potassco

Nogoods from logic programs
loop nogoods

- For a logic program P and some $\emptyset \subset U \subseteq atom(P)$, define the loop nogood of an atom $a \in U$ as
$$\lambda(a, U) \quad = \quad \{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\}$$
where $EB_P(U) = \{B_1, \dots, B_k\}$
- We get the following set of loop nogoods for P:
$$\Lambda_P \quad = \quad \bigcup_{\emptyset \subset U \subseteq atom(P)} \{\lambda(a, U) \mid a \in U\}$$
- The set Λ_P of loop nogoods denies cyclic support among *true* atoms

Nogoods from logic programs
loop nogoods

- For a logic program P and some $\emptyset \subset U \subseteq atom(P)$, define the loop nogood of an atom $a \in U$ as
$$\lambda(a, U) \quad = \quad \{\boldsymbol{T}a, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}$$
where $EB_P(U) = \{B_1, \ldots, B_k\}$
- We get the following set of loop nogoods for P:
$$\Lambda_P \quad = \quad \bigcup_{\emptyset \subset U \subseteq atom(P)} \{\lambda(a, U) \mid a \in U\}$$
- The set Λ_P of loop nogoods denies cyclic support among *true* atoms

Nogoods from logic programs
loop nogoods

- For a logic program P and some $\emptyset \subset U \subseteq atom(P)$, define the loop nogood of an atom $a \in U$ as

$$\lambda(a, U) \quad = \quad \{\boldsymbol{T}a, \boldsymbol{F}B_1, \dots, \boldsymbol{F}B_k\}$$

  where $EB_P(U) = \{B_1, \dots, B_k\}$

- We get the following set of loop nogoods for P:

$$\Lambda_P \quad = \quad \bigcup_{\emptyset \subset U \subseteq atom(P)} \{\lambda(a, U) \mid a \in U\}$$

- The set Λ_P of loop nogoods denies cyclic support among *true* atoms

Example

- Consider the program

$$\left\{ \begin{array}{ll} x \leftarrow \sim y & u \leftarrow x \\ & u \leftarrow v \\ y \leftarrow \sim x & v \leftarrow u, y \end{array} \right\}$$

- For u in the set $\{u, v\}$, we obtain the loop nogood:

$$\lambda(u, \{u, v\}) \quad = \quad \{\boldsymbol{T}u, \boldsymbol{F}\{x\}\}$$

Similarly for v in $\{u, v\}$, we get:

$$\lambda(v, \{u, v\}) \quad = \quad \{\boldsymbol{T}v, \boldsymbol{F}\{x\}\}$$

Potassco

Example

- Consider the program

$$\left\{ \begin{array}{ll} x \leftarrow \sim y & u \leftarrow x \\ & u \leftarrow v \\ y \leftarrow \sim x & v \leftarrow u, y \end{array} \right\}$$

- For u in the set $\{u, v\}$, we obtain the loop nogood:

$$\lambda(u, \{u, v\}) \quad = \quad \{\boldsymbol{T}u, \boldsymbol{F}\{x\}\}$$

Similarly for v in $\{u, v\}$, we get:

$$\lambda(v, \{u, v\}) \quad = \quad \{\boldsymbol{T}v, \boldsymbol{F}\{x\}\}$$

Example

- Consider the program

$$\left\{ \begin{array}{ll} x \leftarrow \sim y & u \leftarrow x \\ & u \leftarrow v \\ y \leftarrow \sim x & v \leftarrow u, y \end{array} \right\}$$

- For u in the set $\{u, v\}$, we obtain the loop nogood:

$$\lambda(u, \{u, v\}) \quad = \quad \{\boldsymbol{T}u, \boldsymbol{F}\{x\}\}$$

Similarly for v in $\{u, v\}$, we get:

$$\lambda(v, \{u, v\}) \quad = \quad \{\boldsymbol{T}v, \boldsymbol{F}\{x\}\}$$

Characterization of stable models

Theorem

Let P be a logic program. Then,
$X \subseteq atom(P)$ *is a stable model of* P *iff*
$X = A^{\boldsymbol{T}} \cap atom(P)$ *for a (unique) solution* A *for* $\Delta_P \cup \Lambda_P$

Some remarks

- Nogoods in Λ_P augment Δ_P with conditions checking for unfounded sets, in particular, those being loops
- While $|\Delta_P|$ is linear in the size of P, Λ_P may contain exponentially many (non-redundant) loop nogoods

Characterization of stable models

Theorem

Let P be a logic program. Then,
$X \subseteq atom(P)$ *is a stable model of* P *iff*
$X = A^{\boldsymbol{T}} \cap atom(P)$ *for a (unique) solution* A *for* $\Delta_P \cup \Lambda_P$

Some remarks

- Nogoods in Λ_P augment Δ_P with conditions checking for unfounded sets, in particular, those being loops
- While $|\Delta_P|$ is linear in the size of P, Λ_P may contain exponentially many (non-redundant) loop nogoods

Outline

Towards conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

- Traditional DPLL-style approach
  (DPLL stands for 'Davis-Putnam-Logemann-Loveland')
  - (Unit) propagation
  - (Chronological) backtracking
  - in ASP, eg *smodels*
- Modern CDCL-style approach
  (CDCL stands for 'Conflict-Driven Constraint Learning')
  - (Unit) propagation
  - Conflict analysis (via resolution)
  - Learning + Backjumping + Assertion
  - in ASP, eg *clasp*

DPLL-style solving

loop

    propagate   // deterministically assign literals
    if no conflict **then**
        if all variables assigned **then return** solution
        else *decide*   // non-deterministically assign some literal
    else
        if top-level conflict **then return** unsatisfiable
        else
            backtrack   // unassign literals propagated after last decision
            flip   // assign complement of last decision literal

Potassco

CDCL-style solving

loop
- *propagate* // deterministically assign literals
- **if** no conflict **then**
  - **if** all variables assigned **then return** solution
  - **else** *decide* // non-deterministically assign some literal
- **else**
  - **if** top-level conflict **then return** unsatisfiable
  - **else**
    - *analyze* // analyze conflict and add conflict constraint
    - *backjump* // unassign literals until conflict constraint is unit

Outline

Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion [Δ_P]
  - Loop nogoods, determined and recorded on demand [Λ_P]
  - Dynamic nogoods, derived from conflicts and unfounded sets [∇]
- When a nogood in $\Delta_P \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for δ
  - Assert the complement of the UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding a stable model (a solution for $\Delta_P \cup \Lambda_P$)
  - Deriving a conflict independently of (heuristic) choices

Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion [Δ_P]
  - Loop nogoods, determined and recorded on demand [Λ_P]
  - Dynamic nogoods, derived from conflicts and unfounded sets [∇]
- When a nogood in $\Delta_P \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for δ
  - Assert the complement of the UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding a stable model (a solution for $\Delta_P \cup \Lambda_P$)
  - Deriving a conflict independently of (heuristic) choices

Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion $[\Delta_P]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_P]$
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$
- When a nogood in $\Delta_P \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for δ
  - Assert the complement of the UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding a stable model (a solution for $\Delta_P \cup \Lambda_P$)
  - Deriving a conflict independently of (heuristic) choices

Algorithm 2: CDNL-ASP

Input : A normal program P
Output : A stable model of P or "no stable model"

$A := \emptyset$ // *assignment over* $atom(P) \cup body(P)$
$\nabla := \emptyset$ // *set of recorded nogoods*
$dl := 0$ // *decision level*

loop
- $(A, \nabla) := \text{NogoodPropagation}(P, \nabla, A)$
- **if** $\varepsilon \subseteq A$ **for some** $\varepsilon \in \Delta_P \cup \nabla$ **then** // *conflict*
  - **if** $\max(\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \cup \{0\}) = 0$ **then return** no stable model
  - $(\delta, dl) := \text{ConflictAnalysis}(\varepsilon, P, \nabla, A)$
  - $\nabla := \nabla \cup \{\delta\}$ // *(temporarily) record conflict nogood*
  - $A := A \setminus \{\sigma \in A \mid dl < dlevel(\sigma)\}$ // *backjumping*
- **else if** $A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}} = atom(P) \cup body(P)$ **then** // *stable model*
  - **return** $A^{\boldsymbol{T}} \cap atom(P)$
- **else**
  - $\sigma_d := \text{Select}(P, \nabla, A)$ // *decision*
  - $dl := dl + 1$
  - $dlevel(\sigma_d) := dl$
  - $A := A \circ \sigma_d$

Potassco

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A
- For a heuristically chosen literal $\sigma_d = \boldsymbol{T}a$ or $\sigma_d = \boldsymbol{F}a$, respectively, we require $a \in (atom(P) \cup body(P)) \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of stable models
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$
  - After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation
  - No explicit flipping of heuristically chosen literals !

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A
- For a heuristically chosen literal $\sigma_d = \boldsymbol{T}a$ or $\sigma_d = \boldsymbol{F}a$, respectively, we require $a \in (atom(P) \cup body(P)) \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of stable models
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$
  - After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation
  - No explicit flipping of heuristically chosen literals !

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A
- For a heuristically chosen literal $\sigma_d = \boldsymbol{T}a$ or $\sigma_d = \boldsymbol{F}a$, respectively, we require $a \in (atom(P) \cup body(P)) \setminus (A^{\boldsymbol{T}} \cup A^{\boldsymbol{F}})$
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of stable models
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$
  - After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation
  - No explicit flipping of heuristically chosen literals !

Potassco

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{\sim x, \sim y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\mathbf{F}\{\sim y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{\sim y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✗

Potassco

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
		$\boldsymbol{T}x$	$\{\boldsymbol{T}u, \boldsymbol{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{x\}\} \in \Delta(v)$
		$\boldsymbol{F}y$	$\{\boldsymbol{T}y, \boldsymbol{F}\{\sim x\}\} = \delta(y)$
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow {\sim}y & u \leftarrow x, y & v \leftarrow x & w \leftarrow {\sim}x, {\sim}y \\ y \leftarrow {\sim}x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
		$\mathbf{T}x$	$\{\mathbf{T}u, \mathbf{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{x\}\} \in \Delta(v)$
		$\mathbf{F}y$	$\{\mathbf{T}y, \mathbf{F}\{{\sim}x\}\} = \delta(y)$
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{{\sim}x, {\sim}y\}\} = \delta(w)$

Potassco

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
		$\boldsymbol{T}x$	$\{\boldsymbol{T}u, \boldsymbol{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{x\}\} \in \Delta(v)$
		$\boldsymbol{F}y$	$\{\boldsymbol{T}y, \boldsymbol{F}\{\sim x\}\} = \delta(y)$
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$

Example: CDNL-ASP

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow {\sim}y & u \leftarrow x, y & v \leftarrow x & w \leftarrow {\sim}x, {\sim}y \\ y \leftarrow {\sim}x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
		$\boldsymbol{T}x$	$\{\boldsymbol{T}u, \boldsymbol{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{x\}\} \in \Delta(v)$
		$\boldsymbol{F}y$	$\{\boldsymbol{T}y, \boldsymbol{F}\{{\sim}x\}\} = \delta(y)$
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{{\sim}x, {\sim}y\}\} = \delta(w)$

Outline

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_P and ∇;
  - Unfounded sets $U \subseteq atom(P)$
- Note that U is unfounded if $EB_P(U) \subseteq A^{\mathbf{F}}$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{\mathbf{T}a\}) \subseteq A$
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(P) \setminus A^{\mathbf{F}})$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of P
  - Note Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $a \in U$, adding $\lambda(a, U)$ to ∇ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_P and ∇;
  - Unfounded sets $U \subseteq atom(P)$
- Note that U is unfounded if $EB_P(U) \subseteq A^{\boldsymbol{F}}$
  - Note For any $a \in U$, we have $(\lambda(a,U) \setminus \{\boldsymbol{T}a\}) \subseteq A$
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(P) \setminus A^{\boldsymbol{F}})$$

- Wrt a fixpoint of unit propagation,
  such an unfounded set contains some loop of P
  - Note Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $a \in U$, adding $\lambda(a,U)$ to ∇ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_P and ∇;
  - Unfounded sets $U \subseteq atom(P)$
- Note that U is unfounded if $EB_P(U) \subseteq A^{\boldsymbol{F}}$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{\boldsymbol{T}a\}) \subseteq A$
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(P) \setminus A^{\boldsymbol{F}})$$

- Wrt a fixpoint of unit propagation,
  such an unfounded set contains some loop of P
  - Note Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $a \in U$, adding $\lambda(a, U)$ to ∇ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_P and ∇;
  - Unfounded sets $U \subseteq atom(P)$
- Note that U is unfounded if $EB_P(U) \subseteq A^{\boldsymbol{F}}$
  - Note For any $a \in U$, we have $(\lambda(a,U) \setminus \{\boldsymbol{T}a\}) \subseteq A$
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(P) \setminus A^{\boldsymbol{F}})$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of P
  - Note Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $a \in U$, adding $\lambda(a,U)$ to ∇ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$

Algorithm 3: NogoodPropagation

Input : A normal program P, a set ∇ of nogoods, and an assignment A.
Output : An extended assignment and set of nogoods.

$U := \emptyset$ // *unfounded set*

loop
  repeat
    if $\delta \subseteq A$ **for some** $\delta \in \Delta_P \cup \nabla$ **then return** (A, ∇) // *conflict*
    $\Sigma := \{\delta \in \Delta_P \cup \nabla \mid \delta \setminus A = \{\overline{\sigma}\}, \sigma \notin A\}$ // *unit-resulting nogoods*
    if $\Sigma \neq \emptyset$ **then let** $\overline{\sigma} \in \delta \setminus A$ **for some** $\delta \in \Sigma$ **in**
      $dlevel(\sigma) := \max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\overline{\sigma}\}\} \cup \{0\})$
      $A := A \circ \sigma$
  until $\Sigma = \emptyset$
  if $loop(P) = \emptyset$ **then return** (A, ∇)
  $U := U \setminus A^{\boldsymbol{F}}$
  if $U = \emptyset$ **then** $U := \text{UnfoundedSet}(P, A)$
  if $U = \emptyset$ **then return** (A, ∇) // *no unfounded set* $\emptyset \subset U \subseteq atom(P) \setminus A^{\boldsymbol{F}}$
  let $a \in U$ **in**
    $\nabla := \nabla \cup \{\{\boldsymbol{T}a\} \cup \{\boldsymbol{F}B \mid B \in EB_P(U)\}\}$ // *record loop nogood*

Requirements for UnfoundedSet

- Implementations of UnfoundedSet must guarantee the following for a result U
  1. $U \subseteq (atom(P) \setminus A^{\mathbf{F}})$
  2. $EB_P(U) \subseteq A^{\mathbf{F}}$
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(atom(P) \setminus A^{\mathbf{F}})$
- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of P

  - Usually, the latter option is implemented in ASP solvers

Potassco

Requirements for UnfoundedSet

- Implementations of UnfoundedSet must guarantee the following for a result U
  1. $U \subseteq (atom(P) \setminus A^{\mathbf{F}})$
  2. $EB_P(U) \subseteq A^{\mathbf{F}}$
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(atom(P) \setminus A^{\mathbf{F}})$
- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of P
  - Usually, the latter option is implemented in ASP solvers

Example: NogoodPropagation

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Outline

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$
  - Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
  - If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$
  - Iterated resolution progresses in inverse order of assignment
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl
  - This literal σ is called First Unique Implication Point (First-UIP)
  - All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$
  - Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
  - If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$
  - Iterated resolution progresses in inverse order of assignment
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl
  - This literal σ is called First Unique Implication Point (First-UIP)
  - All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$
  - Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
  - If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$
  - Iterated resolution progresses in inverse order of assignment
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl
  - This literal σ is called First Unique Implication Point (First-UIP)
  - All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl

Algorithm 4: ConflictAnalysis

Input : A non-empty violated nogood δ, a normal program P, a set ∇ of nogoods, and an assignment A.

Output : A derived nogood and a decision level.

loop
  let $\sigma \in \delta$ **such that** $\delta \setminus A[\sigma] = \{\sigma\}$ **in**
    $k := \max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\})$
    if $k = dlevel(\sigma)$ **then**
      let $\varepsilon \in \Delta_P \cup \nabla$ **such that** $\varepsilon \setminus A[\sigma] = \{\overline{\sigma}\}$ **in**
        $\delta := (\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$ // *resolution*
    else return (δ, k)

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{\sim x, \sim y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{\sim y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{\sim x, \sim y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\mathbf{F}\{\sim y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{\sim y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{\sim x\}$	$\{\mathbf{F}\{\sim x\}, \mathbf{F}x\} = \delta(\{\sim x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{\sim y\}, \mathbf{F}y\} = \delta(\{\sim y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow {\sim}y & u \leftarrow x, y & v \leftarrow x & w \leftarrow {\sim}x, {\sim}y \\ y \leftarrow {\sim}x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{{\sim}x, {\sim}y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{{\sim}x, {\sim}y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{{\sim}y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{{\sim}y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{{\sim}x\}$	$\{\boldsymbol{F}\{{\sim}x\}, \boldsymbol{F}x\} = \delta(\{{\sim}x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{{\sim}y\}, \boldsymbol{F}y\} = \delta(\{{\sim}y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Potassco

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{\sim x, \sim y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{\sim y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow {\sim}y & u \leftarrow x, y & v \leftarrow x & w \leftarrow {\sim}x, {\sim}y \\ y \leftarrow {\sim}x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{{\sim}x, {\sim}y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{{\sim}x, {\sim}y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{{\sim}y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{{\sim}y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{{\sim}x\}$	$\{\boldsymbol{F}\{{\sim}x\}, \boldsymbol{F}x\} = \delta(\{{\sim}x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{{\sim}y\}, \boldsymbol{F}y\} = \delta(\{{\sim}y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{\sim x, \sim y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{\sim y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Potassco

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\boldsymbol{T}u$		
2	$\boldsymbol{F}\{\sim x, \sim y\}$		
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$
3	$\boldsymbol{F}\{\sim y\}$		
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

$\{\boldsymbol{T}u, \boldsymbol{F}x\}$

$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{\sim x, \sim y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{\sim y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Potassco

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{\sim x, \sim y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{\sim x, \sim y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{\sim y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{\sim y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x, y\}$	$\{\boldsymbol{T}\{x, y\}, \boldsymbol{F}x\} \in \Delta(\{x, y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{\sim x\}$	$\{\boldsymbol{F}\{\sim x\}, \boldsymbol{F}x\} = \delta(\{\sim x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{\sim y\}, \boldsymbol{F}y\} = \delta(\{\sim y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x, y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u, y\}$	$\{\boldsymbol{F}\{u, y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u, y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u, y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Potassco

Example: ConflictAnalysis

Consider

$$P = \left\{ \begin{array}{llll} x \leftarrow {\sim}y & u \leftarrow x, y & v \leftarrow x & w \leftarrow {\sim}x, {\sim}y \\ y \leftarrow {\sim}x & u \leftarrow v & v \leftarrow u, y & \end{array} \right\}$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\boldsymbol{T}u$			
2	$\boldsymbol{F}\{{\sim}x,{\sim}y\}$			
		$\boldsymbol{F}w$	$\{\boldsymbol{T}w, \boldsymbol{F}\{{\sim}x,{\sim}y\}\} = \delta(w)$	
3	$\boldsymbol{F}\{{\sim}y\}$			
		$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{{\sim}y\}\} = \delta(x)$	
		$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\} \in \Delta(\{x\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x\}$
		$\boldsymbol{F}\{x,y\}$	$\{\boldsymbol{T}\{x,y\}, \boldsymbol{F}x\} \in \Delta(\{x,y\})$	$\{\boldsymbol{T}u, \boldsymbol{F}x, \boldsymbol{F}\{x\}\}$
		$\boldsymbol{T}\{{\sim}x\}$	$\{\boldsymbol{F}\{{\sim}x\}, \boldsymbol{F}x\} = \delta(\{{\sim}x\})$	
		$\boldsymbol{T}y$	$\{\boldsymbol{F}\{{\sim}y\}, \boldsymbol{F}y\} = \delta(\{{\sim}y\})$	
		$\boldsymbol{T}\{v\}$	$\{\boldsymbol{T}u, \boldsymbol{F}\{x,y\}, \boldsymbol{F}\{v\}\} = \delta(u)$	
		$\boldsymbol{T}\{u,y\}$	$\{\boldsymbol{F}\{u,y\}, \boldsymbol{T}u, \boldsymbol{T}y\} = \delta(\{u,y\})$	
		$\boldsymbol{T}v$	$\{\boldsymbol{F}v, \boldsymbol{T}\{u,y\}\} \in \Delta(v)$	
			$\{\boldsymbol{T}u, \boldsymbol{F}\{x\}, \boldsymbol{F}\{x,y\}\} = \lambda(u,\{u,v\})$	✘

Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level *dl*
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$
  - After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - Such a nogood δ is called asserting
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level *dl*
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$
  - After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - Such a nogood δ is called asserting
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level *dl*
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$
  - After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - Such a nogood δ is called asserting
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level dl
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$
  - After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - Such a nogood δ is called asserting
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

[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.

[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.

Potassco

[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.

Potassco

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.