Advanced Conflict-Driven Disjunctive Answer Set Solving

Martin Gebser and **Benjamin Kaufmann** and **Torsten Schaub**[*]

Universität Potsdam, Germany

Abstract

We introduce a new approach to disjunctive ASP solving that aims at an equitable interplay between "generating" and "testing" solver units. To this end, we develop novel characterizations of answer sets and unfounded sets allowing for a bidirectional dynamic information exchange between solver units for orthogonal tasks. This results in the new multi-threaded disjunctive ASP solver *claspD-2*, greatly improving the performance of existing systems.

1 Introduction

Answer Set Programming (ASP; [Baral, 2003]) has become a popular tool for declarative problem solving. Its growing range of applications has also increased the demand for the elevated complexity of disjunctive logic programs (expressing Σ_2^P-hard problems). Although recent advances facilitate modeling such problems [Gebser *et al.*, 2011a], existing disjunctive ASP solvers [Leone *et al.*, 2006; Janhunen *et al.*, 2006; Lierler, 2005; Drescher *et al.*, 2008] still lack some features of non-disjunctive solvers. Also, the indispensable coupling of a solver unit generating answer set candidates with one checking their minimality was so far insufficiently exploited: while the "generator" is run continuously, a "tester" is repeatedly re-invoked bearing significant redundancy.

We address these shortcomings by introducing a new approach to disjunctive ASP solving that aims at an equitable interplay between generator(s) and tester(s). The idea is to launch both types of solver units only once with their respective Boolean constraint problems and to let them subsequently communicate in a bidirectional way. This enables both generators and testers to benefit from conflict-driven learning over whole runs. We begin by developing novel characterizations of answer sets and unfounded sets in terms of Boolean constraints solvable with state-of-the-art *NP* search procedures. Unlike existing encodings [Koch *et al.*, 2003; Drescher *et al.*, 2008], our specification of unfounded sets is comprehensive in applying to arbitrary (total or partial) answer set candidates. We further present a solver

architecture featuring multi-threading and incremental solving techniques, along with its implementation in the new disjunctive ASP solver *claspD-2*, seamlessly passing dynamic information between orthogonal solver units.

2 Background

A (ground disjunctive) *rule* r is of the form

$$p_1 \vee \cdots \vee p_l \leftarrow p_{l+1}, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n$$

where $p_1, \ldots, p_l, p_{l+1}, \ldots, p_m, p_{m+1}, \ldots, p_n$ are propositional *atoms* for $0 \le l \le m \le n$. By $hd(r) = \{p_1, \ldots, p_l\}$ and $bd(r) = \{p_{l+1}, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n\}$, we denote the *head* and the *body* of r, where $\sim$ stands for default negation. For any set $L = \{p_{l+1}, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n\}$, let $L^+ = \{p_{l+1}, \ldots, p_m\}$ and $L^- = \{p_{m+1}, \ldots, p_n\}$. We say that r is *applicable* wrt. a set X of atoms if $bd(r)^+ \subseteq X$ and $bd(r)^- \cap X = \emptyset$; r is *satisfied* wrt. X if $hd(r) \cap X \neq \emptyset$ when r is applicable wrt. X. Any $p \in hd(r)$ is *supported* by r wrt. X if r is applicable wrt. X and $hd(r) \cap X \subseteq \{p\}$.

A (ground disjunctive) *program* P is a set of rules. We denote atoms and bodies occurring in P by $at_P = \bigcup_{r \in P}(hd(r) \cup bd(r)^+ \cup bd(r)^-)$ and $bd_P = \{bd(r) \mid r \in P\}$. A set X of atoms is a *model* of P if every $r \in P$ is satisfied wrt. X; X is *supported* by P if every $p \in X$ is supported by some $r \in P$ wrt. X. We write $app_P(X)$ to refer to the subset of P including all applicable rules wrt. X. The *reduct* of P wrt. X is $P^X = \{hd(r) \leftarrow bd(r)^+ \mid bd(r)^- \cap X = \emptyset\}$. A model X of P is an *answer set* of P if there is no $Y \subset X$ such that Y is a model of P^X.

The (directed) *positive atom dependency graph* of P is $G_P = (at_P, \{(p, q) \mid r \in P, p \in hd(r), q \in bd(r)^+\})$. A non-empty $L \subseteq at_P$ is a *loop* of P if its induced subgraph of G_P is strongly connected. We denote the set of all loops of P by $loop_P$. As the $\subseteq$-maximal elements of $loop_P$ partition at_P and induce *strongly connected components* (SCCs) of G_P, we write scc_P to refer to the set of such loops. We call $L \in scc_P$ a *head cycle component* (HCC) of P if $|hd(r) \cap L| > 1$ for some $r \in P$. By scc_P^{hc}, we denote the subset of scc_P including all HCCs of P; P is *head-cycle-free* (HCF; [Ben-Eliyahu and Dechter, 1994]) if $scc_P^{hc} = \emptyset$, and non-HCF otherwise.

As an example, consider the following program:

$$P_1 = \left\{ \begin{array}{lll} r_1: & a \vee c \vee e \leftarrow & r_2: \quad a \leftarrow b, \sim d \\ r_3: & b \leftarrow a, \sim e & r_4: \quad b \leftarrow c, d \\ r_5: & c \vee d \leftarrow b \end{array} \right\}$$

P_1 yields $loop_{P_1} = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{b, c\},$ $\{b, d\}, \{a, b, c\}, \{a, b, d\}, \{b, c, d\}, \{a, b, c, d\}\}$ and $scc_{P_1} =$ $\{\{a, b, c, d\}, \{e\}\}$. In view of rules r_1 and r_5, we have that $scc_{P_1}^{hc} = \{\{a, b, c, d\}\}$, so that P_1 is non-HCF. One can check that $\{a, b, d\}$, $\{c\}$, and $\{e\}$ are the three answer sets of P_1.

3 Boolean Constraints

Deciding whether a (ground disjunctive) program has an answer set is Σ_2^P-complete [Eiter and Gottlob, 1995]. In fact, generating candidate models X of a program P and checking $\subseteq$-minimality wrt. P^X can both be viewed as Boolean constraint solving tasks, which can be specified as follows.

A *nogood* $\{\sigma_1, \ldots, \sigma_n\}$ is a set of *literals* σ_i of the form $\boldsymbol{T}v_i$ or $\boldsymbol{F}v_i$ for $1 \leq i \leq n$, where v_i is a (propositional) *variable*. We refer to the *complement* of a literal by $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$. For any set δ of literals, let $\delta^{\boldsymbol{T}} = \{v \mid \boldsymbol{T}v \in \delta\}$ and $\delta^{\boldsymbol{F}} = \{v \mid \boldsymbol{F}v \in \delta\}$. Given a set Δ of nogoods, we denote the variables occurring in Δ by $var(\Delta) = \bigcup_{\delta \in \Delta}(\delta^{\boldsymbol{T}} \cup \delta^{\boldsymbol{F}})$. Any subset $\boldsymbol{A}$ of $\{\boldsymbol{T}v, \boldsymbol{F}v \mid v \in var(\Delta)\}$ such that $\boldsymbol{A}^{\boldsymbol{T}} \cap \boldsymbol{A}^{\boldsymbol{F}} = \emptyset$ is an *assignment* for Δ; $\boldsymbol{A}$ is a *solution* for Δ if $\boldsymbol{A}^{\boldsymbol{T}} \cup \boldsymbol{A}^{\boldsymbol{F}} = var(\Delta)$ and $\delta \not\subseteq \boldsymbol{A}$ for any $\delta \in \Delta$.

Answer Sets. In order to compute answer sets by means of Boolean constraint solving procedures, we specify nogoods whose solutions match answer sets. In fact, the nogoods capture particular properties of answer sets, viz. satisfaction of rules, support of atoms, and minimality wrt. the reduct.

For expressing that some atom in $hd(r)$ must be true when a rule r is applicable, we map $bd(r)$ to literals making it true:

$$\beta(r) = \{\boldsymbol{T}p \mid p \in bd(r)^+\} \cup \{\boldsymbol{F}p \mid p \in bd(r)^-\}$$

Eg., for r_2 from P_1, $bd(r_2) = \{b, \sim d\}$ yields $\beta(r_2) = \{\boldsymbol{T}b, \boldsymbol{F}d\}$. We use such sets of literals to refer to (composite) variables with an intrinsic meaning (cf. Definition 3) in the following nogoods stipulating rules to be satisfied.

Definition 1. *Let P be a program. We define the rule nogood for any $r \in P$ by*

$$\phi(r) = \{\boldsymbol{F}p_1, \ldots, \boldsymbol{F}p_l, \boldsymbol{T}\beta(r) \mid hd(r) = \{p_1, \ldots, p_l\}\}.$$

The rule nogoods of P are $\Phi_P = \{\phi(r) \mid r \in P\}$.

For instance, the nogoods for rules in P_1 are as follows:

$$\Phi_{P_1} = \begin{Bmatrix} \phi(r_1) = \{\boldsymbol{F}a, \boldsymbol{F}c, \boldsymbol{F}e, \boldsymbol{T}\emptyset\}, \\ \phi(r_2) = \{\boldsymbol{F}a, \boldsymbol{T}\{\boldsymbol{T}b, \boldsymbol{F}d\}\}, \\ \phi(r_3) = \{\boldsymbol{F}b, \boldsymbol{T}\{\boldsymbol{T}a, \boldsymbol{F}e\}\}, \\ \phi(r_4) = \{\boldsymbol{F}b, \boldsymbol{T}\{\boldsymbol{T}c, \boldsymbol{T}d\}\}, \\ \phi(r_5) = \{\boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{T}\{\boldsymbol{T}b\}\} \end{Bmatrix}$$

Given that any answer set X of a program P is supported by P, for every $p \in X$, there must be some applicable rule $r \in P$ such that no $q \in hd(r) \backslash \{p\}$ belongs to X. Support can be captured by *atom-wise shifting* [Ben-Eliyahu and Dechter, 1994], mapping a rule r to $\vec{r}(p)$ and identifying $bd(\vec{r}(p))$ with an extended set $\beta(r, p)$ of literals:

$$\vec{r}(p) = p \leftarrow bd(r) \cup \{\sim q \mid q \in hd(r) \setminus \{p\}\}$$
$$\beta(r, p) = \{\boldsymbol{T}\beta(r)\} \cup \{\boldsymbol{F}q \mid q \in hd(r) \setminus \{p\}\}$$

For instance, atom-wise shifting of $r_5 = c \vee d \leftarrow b$ yields $\vec{r}_5(c) = c \leftarrow b, \sim d$ and $\vec{r}_5(d) = d \leftarrow b, \sim c$. Respective

extensions of $bd(r_5) = \{b\}$ are reflected by the literals in $\beta(r_5, c) = \{\boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{F}d\}$ and $\beta(r_5, d) = \{\boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{F}c\}$. Note that the original body representation, as included in a rule nogood, is reused, rather than repeatedly unfolding the body for each head atom. On the other hand, atom-wise shifting may reproduce the body of another program rule; eg., we have that $bd(\vec{r}_5(c)) = \{b, \sim d\} = bd(r_2)$.

In view of the previous observation, we introduce the following concept to switch between the reuse of an existing rule body or the introduction of a new set of literals for representing a body stemming from atom-wise shifting:

$$\beta_P(r, p) = \begin{cases} \beta(\vec{r}(p)) & \text{if } bd(\vec{r}(p)) \in bd_P \\ \beta(r, p) & \text{if } bd(\vec{r}(p)) \notin bd_P \end{cases}$$

Since $bd(\vec{r}_5(c)) = \{b, \sim d\} = bd(r_2)$, while $bd(\vec{r}_5(d)) = \{b, \sim c\} \notin bd_{P_1}$, the representations $\beta_{P_1}(r_5, c) = \{\boldsymbol{T}b, \boldsymbol{F}d\}$ and $\beta_{P_1}(r_5, d) = \{\boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{F}c\}$ are selected for bodies obtained through atom-wise shifting of r_5. In general, $\beta_P(r, p)$ picks the original body representation $\beta(r)$ if $hd(r) = \{p\}$, so that rules without proper disjunctive head do not lead to new variables for bodies in the support nogoods given next.

Definition 2. *Let P be a program. We define the support nogood for any $p \in at_P$ by*

$$\psi_P(p) = \{\boldsymbol{T}p\} \cup \{\boldsymbol{F}\beta_P(r, p) \mid r \in P, p \in hd(r)\}.$$

The support nogoods of P are $\Psi_P = \{\psi_P(p) \mid p \in at_P\}$.

A nogood of the form $\psi_P(p)$ expresses that p must not be true if it is not supported by any rule in P, as indicated by $\boldsymbol{F}\beta_P(r, p)$ for all rules with p in the head. For instance, we obtain the following support nogoods for the atoms of P_1:

$$\Psi_{P_1} = \begin{Bmatrix} \psi_{P_1}(a) = \{\boldsymbol{T}a, \boldsymbol{F}\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \boldsymbol{F}\{\boldsymbol{T}b, \boldsymbol{F}d\}\}, \\ \psi_{P_1}(b) = \{\boldsymbol{T}b, \boldsymbol{F}\{\boldsymbol{T}a, \boldsymbol{F}e\}, \boldsymbol{F}\{\boldsymbol{T}c, \boldsymbol{T}d\}\}, \\ \psi_{P_1}(c) = \{\boldsymbol{T}c, \boldsymbol{F}\{\boldsymbol{T}\emptyset, \boldsymbol{F}a, \boldsymbol{F}e\}, \boldsymbol{F}\{\boldsymbol{T}b, \boldsymbol{F}d\}\}, \\ \psi_{P_1}(d) = \{\boldsymbol{T}d, \boldsymbol{F}\{\boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{F}c\}\}, \\ \psi_{P_1}(e) = \{\boldsymbol{T}e, \boldsymbol{F}\{\boldsymbol{T}\emptyset, \boldsymbol{F}a, \boldsymbol{F}c\}\} \end{Bmatrix}$$

Rule and support nogoods define the atoms of a program in terms of rule bodies, possibly obtained through atom-wise shifting. Bodies in turn represent the conjunction of their contained literals, which is captured by defining the corresponding variables by means of the following nogoods.

Definition 3. *Let P be a program and β a set of literals. We define the conjunction nogoods for β by*

$$\gamma(\beta) = \{\{\boldsymbol{F}\beta\} \cup \beta\} \cup \{\{\boldsymbol{T}\beta, \overline{\sigma}\} \mid \sigma \in \beta\}.$$

The conjunction nogoods of P are

$$\Gamma_P = \bigcup_{\beta \in \{\beta(r)|r \in P\} \cup \{\beta_P(r, p)|r \in P, p \in hd(r)\}} \gamma(\beta).$$

Observe that Γ_P includes defining nogoods for literal sets $\beta(r)$ representing elements of bd_P in Φ_P as well as for extensions $\beta(r, p)$, selected via $\beta_P(r, p)$, that occur in Ψ_P. Eg., $\beta(r_1) = \emptyset$, $\beta(r_2) = \{\boldsymbol{T}b, \boldsymbol{F}d\}$, and $\beta_{P_1}(r_1, a) = \{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}$ are defined via the following subsets of Γ_{P_1}:

$$\gamma(\emptyset) = \{\{\boldsymbol{F}\emptyset\}\}$$

$$\gamma(\{\boldsymbol{T}b, \boldsymbol{F}d\}) = \begin{Bmatrix} \{\boldsymbol{F}\{\boldsymbol{T}b, \boldsymbol{F}d\}, \boldsymbol{T}b, \boldsymbol{F}d\}, \\ \{\boldsymbol{T}\{\boldsymbol{T}b, \boldsymbol{F}d\}, \boldsymbol{F}b\}, \\ \{\boldsymbol{T}\{\boldsymbol{T}b, \boldsymbol{F}d\}, \boldsymbol{T}d\} \end{Bmatrix}$$

$$\gamma(\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}) = \left\{ \begin{array}{l} \{\boldsymbol{F}\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \\ \{\boldsymbol{T}\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \boldsymbol{F}\emptyset\}, \\ \{\boldsymbol{T}\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \boldsymbol{T}c\}, \\ \{\boldsymbol{T}\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}, \boldsymbol{T}e\} \end{array} \right\}$$

In particular, $\{\boldsymbol{F}\emptyset\}$ requires $\boldsymbol{T}\emptyset$, which occurs in a nogood defining $\{\boldsymbol{T}\emptyset, \boldsymbol{F}c, \boldsymbol{F}e\}$, to belong to any solution for Γ_{P_1}.

Although the combined set $\Phi_P \cup \Psi_P \cup \Gamma_P$ of nogoods suffices to characterize models of P that are supported by P, we rely on *component-wise shifting* [Drescher *et al.*, 2008] as an a priori simplification, particularly for (complex) minimality checking detailed below. To this end, we refer to the SCCs of head atoms in a rule r by $scc_P(r) = \{L \in scc_P \mid hd(r) \cap L \neq \emptyset\}$. Component-wise shifting then decomposes rules (with proper disjunctive heads) on the basis of SCCs.

Definition 4. *Let P be a program. We define the component-shifted version of P by*

$$\vec{P} = \{hd(r) \cap L \leftarrow bd(r) \cup \{\sim p \mid p \in hd(r) \setminus L\} \mid$$
$$r \in P, L \in scc_P(r)\} \cup \{r \in P \mid hd(r) = \emptyset\}.$$

For P_1, component-wise shifting yields the following:

$$\vec{P}_1 = \begin{cases} r_0: & e \leftarrow \sim a, \sim c \\ r_1: & a \vee c \leftarrow \sim e \end{cases} \cup \{r_2, r_3, r_4, r_5\}$$

Note that the heads of rules in $\vec{P}_1$ are confined to SCCs, viz. $hd(r) \subseteq \{a, b, c, d\}$ or $hd(r) \subseteq \{e\}$ holds for all $r \in \vec{P}_1$. In general, a program P is non-HCF iff its component-shifted version $\vec{P}$ includes some rule with proper disjunctive head.

Unlike atom-wise shifting, component-wise shifting preserves the (classical) semantics of loop formulas [Lee and Lifschitz, 2003]. Hence, a program P and its component-shifted version $\vec{P}$ are equivalent.

Proposition 1. *Let P be a program and X a set of atoms. We have that X is an answer set of P iff X is an answer set of $\vec{P}$.*

The previous result tells us that nogoods aiming at answer sets of P may safely concentrate on $\vec{P}$. In fact, we define the nogoods capturing *completion* [Lee and Lifschitz, 2003] relative to the component-shifted version $\vec{P}$ of P.

Definition 5. *Let P be a program. We define the completion nogoods of P by $\Delta_P = \Phi_{\vec{P}} \cup \Psi_{\vec{P}} \cup \Gamma_{\vec{P}}$.*

For instance, the rule nogood $\{\boldsymbol{F}a, \boldsymbol{F}c, \boldsymbol{F}e, \boldsymbol{T}\emptyset\}$ from Φ_{P_1} turns into $\phi(r_0) = \{\boldsymbol{F}e, \boldsymbol{T}\{\boldsymbol{F}a, \boldsymbol{F}c\}\}$ and $\phi(r_1) = \{\boldsymbol{F}a, \boldsymbol{F}c, \boldsymbol{T}\{\boldsymbol{F}e\}\}$ in $\Phi_{\vec{P}_1}$ and Δ_{P_1}. Observe that $\phi(r_0)$ and $\phi(r_1)$ (in view of the conjunction nogood $\{\boldsymbol{F}\emptyset\}$ from Γ_{P_1}) exclude the same combination of literals over a, c, and e as $\{\boldsymbol{F}a, \boldsymbol{F}c, \boldsymbol{F}e, \boldsymbol{T}\emptyset\}$, yet relying on different variables (sets of literals) standing for rule bodies. As we describe below, the bodies obtained through component-wise shifting can be readily reused in the context of (complex) minimality checking for signaling the (non-)applicability of rules.

In order to establish correspondences between models of a program and solutions for nogoods, we map any set of atoms to an induced assignment as follows.

Definition 6. *Let P be a program and X a set of atoms. We define the induced assignment of X for P by*

$$\mathbf{A}_P^X = \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in at_P \setminus X\}$$

$$\cup \{\boldsymbol{T}\beta(r) \mid r \in app_{\vec{P}}(X)\} \cup \{\boldsymbol{F}\beta(r) \mid r \in \vec{P} \setminus app_{\vec{P}}(X)\}$$

$$\cup \{\boldsymbol{T}\beta_{\vec{P}}(r, p) \mid r \in app_{\vec{P}}(X), hd(r) \cap (X \cup \{p\}) = \{p\}\}$$

$$\cup \{\boldsymbol{F}\beta_{\vec{P}}(r, p) \mid r \in \vec{P} \setminus app_{\vec{P}}(X), p \in hd(r)\}$$

$$\cup \{\boldsymbol{F}\beta_{\vec{P}}(r, p) \mid r \in \vec{P}, p \in hd(r), hd(r) \cap X \nsubseteq \{p\}\}.$$

Beyond literals over atoms, an induced assignment $\mathbf{A}_P^X$ includes either $\boldsymbol{T}\beta(r)$ or $\boldsymbol{F}\beta(r)$ for each $r \in \vec{P}$ along with either $\boldsymbol{T}\beta(r, p)$ or $\boldsymbol{F}\beta(r, p)$ for any extension $\beta(r, p)$ obtained through atom-wise shifting and selected via $\beta_{\vec{P}}(r, p)$. For instance, the induced assignment $\mathbf{A}_{P_1}^{\{a,b,d\}}$ contains $\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c$, $\boldsymbol{T}d, \boldsymbol{F}e, \boldsymbol{T}\beta(r_5) = \boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{T}\beta_{\vec{P}_1}(r_5, d) = \boldsymbol{T}\{\boldsymbol{T}\{\boldsymbol{T}b\}, \boldsymbol{F}c\}$, and further literals over rule bodies.

As stated next, variables standing for rule bodies are defined via conjunction nogoods in $\Gamma_{\vec{P}}$ and do thus not introduce additional combinatorics regarding solutions for Δ_P.

Proposition 2. *Let P be a program and X a set of atoms. We have that $\mathbf{A}_P^{at_P \cap X}$ is the unique solution for*

$$\{\{\boldsymbol{F}p\} \mid p \in at_P \cap X\} \cup \{\{\boldsymbol{T}p\} \mid p \in at_P \setminus X\} \cup \Gamma_{\vec{P}}.$$

In view of the above result, the following yields a one-to-one correspondence between models of P that are supported by P and solutions for the completion nogoods Δ_P.

Theorem 1. *Let P be a program and X a set of atoms. We have that X is a model of P that is supported by P iff $\mathbf{A}_P^X$ is a solution for Δ_P.*

For instance, $\{a, b, c\}$ is a model of P_1 that is supported by P_1, and the support nogoods $\psi_{\vec{P}_1}(a)$, $\psi_{\vec{P}_1}(b)$, and $\psi_{\vec{P}_1}(c)$ are not contained in the induced assignment $\mathbf{A}_{P_1}^{\{a,b,c\}}$, which includes $\boldsymbol{T}\beta_{\vec{P}_1}(r_2, a) = \boldsymbol{T}\beta_{\vec{P}_1}(r_5, c) = \boldsymbol{T}\{\boldsymbol{T}b, \boldsymbol{F}d\}$ and $\boldsymbol{T}\beta_{\vec{P}_1}(r_3, b) = \boldsymbol{T}\{\boldsymbol{T}a, \boldsymbol{F}e\}$. However, $\{a, b, c\}$ is not an answer set of P_1 because $\{c\}$ is a model of $P_1^{\{a,b,c\}}$.

In order to make sure that solutions match answer sets, we still need to guarantee that true atoms are non-circularly supported. To this end, we denote the *external supports* of a set L of atoms for a program P by $\varepsilon_P(L) = \{r \in P \mid hd(r) \cap L \neq \emptyset, bd(r)^+ \cap L = \emptyset\}$. Moreover, $\rho(r, L) = \{\boldsymbol{F}\beta(r)\} \cup \{\boldsymbol{T}p \mid p \in hd(r) \setminus L\}$ collects all literals satisfying a rule r regardless of whether any atom from L is true. E.g., $\varepsilon_{\vec{P}_1}(\{a, b\})$ consists of $r_1 = a \vee c \leftarrow \sim e$ and $r_4 = b \leftarrow c, d$, whose associated literal sets are $\rho(r_1, \{a, b\}) = \{\boldsymbol{F}\{\boldsymbol{F}e\}, \boldsymbol{T}c\}$ and $\rho(r_4, \{a, b\}) = \{\boldsymbol{F}\{\boldsymbol{T}c, \boldsymbol{T}d\}\}$.

We say that a set L of atoms is *unfounded* [Leone *et al.*, 1997] for a program P wrt. an assignment $\mathbf{A}$ if $\rho(r, L) \cap \mathbf{A} \neq \emptyset$ holds for any $r \in \varepsilon_P(L)$. For instance, $\{a, b\}$ is unfounded for $\vec{P}_1$ wrt. $\mathbf{A}_{P_1}^{\{a,b,c\}}$, which includes $\boldsymbol{T}c \in \rho(r_1, \{a, b\})$ and $\boldsymbol{F}\{\boldsymbol{T}c, \boldsymbol{T}d\} \in \rho(r_4, \{a, b\})$. Unfounded sets are addressed by loop nogoods as follows.

Definition 7. *Let P be a program. We define the loop nogoods for any $L \subseteq at_P$ by*

$$\lambda_P(L) = \{\{\boldsymbol{T}p, \sigma_1, \ldots, \sigma_k\} \mid \varepsilon_P(L) = \{r_1, \ldots, r_k\},$$
$$p \in L, \sigma_1 \in \rho(r_1, L), \ldots, \sigma_k \in \rho(r_k, L)\}.$$

The loop nogoods of P are $\Lambda_P = \bigcup_{L \subseteq at_P} \lambda_{\vec{P}}(L)$.

Loop nogoods in $\lambda_P(L)$ express that every atom from L must be false if all external supports of L for P are satisfied independently of L, as indicated by some literal from $\rho(r, L)$ for each $r \in \varepsilon_P(L)$. Also note that Λ_P collects loop nogoods constructed from $\vec{P}$ to align literals over bodies of external supports with variables defined via the conjunction nogoods $\Gamma_{\vec{P}}$. Eg., Λ_{P_1} includes $\lambda_{\vec{P}_1}(\{a, b\}) = \{\{Tp, F\{Fe\}, F\{Tc, Td\}\}, \{Tp, Tc, F\{Tc, Td\}\} \mid p \in \{a, b\}\}$, among which $\{Ta, Tc, F\{Tc, Td\}\}$ and $\{Tb, Tc, F\{Tc, Td\}\}$ are contained in $\mathbf{A}_{P_1}^{\{a,b,c\}}$.

Augmenting completion with loop nogoods yields a one-to-one correspondence between solutions and answer sets.

Theorem 2. *Let P be a program and X a set of atoms. We have that X is an answer set of P iff $\mathbf{A}_P^X$ is a solution for $\Delta_P \cup \Lambda_P$.*

The above result characterizes answer sets in terms of (induced) assignments encapsulating "unfounded-free" models in the sense of Leone *et al.* (1997), who show that unfounded set checks can be localized to SCCs of G_P. Local checks are tractable for elements of $scc_P \setminus scc_P^{hc}$, but not for the HCCs in scc_P^{hc}, so that evaluating loop nogoods for the (exponentially many) subsets of HCCs is computationally complex.

Unfounded Sets. Given the complexity of identifying unfounded sets contained in HCCs, we specify nogoods having such unfounded sets as solutions. The nogoods require some true atom not belonging to the unfounded set to be present in the head of any external support whose body is not false. We thus refer to atoms that occur together with members of a set L of atoms in a proper disjunctive head for a program P by $L_P^{hc} = \{p \in hd(r) \mid r \in P, |hd(r) \cap L| > 1\}$. Given this, unfounded set nogoods for atoms L of SCCs are as follows.

Definition 8. *Let P be a program. We define the unfounded set nogoods for any $L \in scc_P$ by*

$$\Omega_P(L) = \{\{Tu_p, Ff_{\beta(r)}\} \cup \{Fu_q \mid q \in bd(r)^+ \cap L\} \cup$$

$$\{Fh_q \mid q \in hd(r) \setminus \{p\}\} \mid r \in \vec{P}, p \in hd(r) \cap L\}$$

$$\cup \{\{Th_p, Tu_p\}, \{Th_p, Ft_p\}, \{Fh_p, Fu_p, Tt_p\} \mid p \in L_{\vec{P}}^{hc}\}$$

$$\cup \{\{Fu_p \mid p \in L\}\}.$$

The basic idea of $\Omega_P(L)$ is to represent the atoms p in an unfounded subset L' of L by literals of the form Tu_p in a solution. Additional variables are used to encode unfoundedness wrt. an underlying assignment $\mathbf{A}$ for Δ_P. That is, for any $r \in \vec{P}$ such that $hd(r) \cap L \neq \emptyset$, the literal $Tf_{\beta(r)}$ signals that $\beta(r) \in \mathbf{A}^F$, so that r is inapplicable. For atoms $p \in L$ occurring in a proper disjunction in $\vec{P}$, Th_p expresses that $p \in \mathbf{A}^T \setminus L'$, where $p \in \mathbf{A}^T$ is reflected by the literal Tt_p, while Fu_p indicates that $p \notin L'$. Then, for any $r \in \vec{P}$ such that $hd(r) \cap L' \neq \emptyset$, the unfoundedness of L' is witnessed by inapplicability of r, some true atom $q \in hd(r) \setminus L'$, or an atom $p \in bd(r)^+ \cap L'$, as represented by literals of the form $Tf_{\beta(r)}$, Th_q, or Tu_p, respectively. Finally, L' must not be the (trivial) empty unfounded set.

For instance, we obtain the following unfounded set nogoods for the HCC $L_1 = \{a, b, c, d\}$ of $\vec{P}_1$:

$$\Omega_{P_1}(L_1) = \left\{ \begin{array}{l} \{Tu_a, Fh_c, Ff_{\{Fe\}}\}, \\ \{Tu_c, Fh_a, Ff_{\{Fe\}}\}, \\ \{Tu_a, Ff_{\{Tb, Fd\}}, Fu_b\}, \\ \{Tu_b, Ff_{\{Ta, Fe\}}, Fu_a\}, \\ \{Tu_b, Ff_{\{Tc, Td\}}, Fu_c, Fu_d\}, \\ \{Tu_c, Fh_d, Ff_{\{Tb\}}, Fu_b\}, \\ \{Tu_d, Fh_c, Ff_{\{Tb\}}, Fu_b\} \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} \{Th_a, Tu_a\}, \{Th_a, Ft_a\}, \{Fh_a, Fu_a, Tt_a\}, \\ \{Th_c, Tu_c\}, \{Th_c, Ft_c\}, \{Fh_c, Fu_c, Tt_c\}, \\ \{Th_d, Tu_d\}, \{Th_d, Ft_d\}, \{Fh_d, Fu_d, Tt_d\} \end{array} \right\}$$

$$\cup \{\{Fu_a, Fu_b, Fu_c, Fu_d\}\}$$

Eg., $r_5 = c \lor d \leftarrow b$ yields the nogoods $\{Tu_c, Fh_d, Ff_{\{Tb\}}, Fu_b\}$ and $\{Tu_d, Fh_c, Ff_{\{Tb\}}, Fu_b\}$ to prohibit assignments representing a subset L' of L such that $\{c, d\} \cap L' \neq \emptyset$ and $b \notin L'$, while $\{c, d\} \cap \mathbf{A}^T \subseteq L'$ and $\{Tb\} \notin \mathbf{A}^F$ hold wrt. an underlying assignment $\mathbf{A}$ for Δ_{P_1}.

The nogoods in $\Omega_P(L)$ encode (non-empty) unfounded subsets of L wrt. arbitrary assignments for Δ_P. Upon computing answer sets, however, one is interested in unfounded sets wrt. an assignment $\mathbf{A}$ at hand. Hence, we extract nogoods from $\mathbf{A}$ that restrict solutions for $\Omega_P(L)$ accordingly.

Definition 9. *Let P be a program and $\mathbf{A}$ an assignment for Δ_P. We define the assignment nogoods for any $L \in scc_P$ by*

$$\Theta_P^{\mathbf{A}}(L) = \{\{Ff_{\beta(r)}\} \mid r \in \vec{P}, hd(r) \cap L \neq \emptyset, \beta(r) \in \mathbf{A}^F\}$$

$$\cup \{\{Tf_{\beta(r)}\} \mid r \in \vec{P}, hd(r) \cap L \neq \emptyset, \beta(r) \notin \mathbf{A}^F\}$$

$$\cup \{\{Ft_p\} \mid p \in L_{\vec{P}}^{hc} \cap \mathbf{A}^T\} \cup \{\{Tt_p\} \mid p \in L_{\vec{P}}^{hc} \setminus \mathbf{A}^T\}$$

$$\cup \{\{Tu_p\} \mid p \in L \cap \mathbf{A}^F\}.$$

As every nogood in $\Theta_P^{\mathbf{A}}(L)$ is of the form $\{\sigma\}$, it immediately implies the complement $\overline{\sigma}$ of its literal σ. In particular, variables $f_{\beta(r)}$ and t_p, expressing whether bodies $\beta(r)$ are false or atoms $p \in L_{\vec{P}}^{hc}$ are true wrt. $\mathbf{A}$, are partitioned via $\mathbf{A}$ into (necessarily) true and false literals. That is, the variables representing an assignment $\mathbf{A}$ in $\Omega_P(L)$ are fixed through the assignment nogoods $\Theta_P^{\mathbf{A}}(L)$. Moreover, false members of L are excluded from unfounded sets obtainable as solutions.

For example, consider an assignment for Δ_{P_1} as follows:

$$\mathbf{A}_1 = \left\{ \begin{array}{l} Tc, F\{Fa, Fc\}, F\{Tc, Td\}, T\{Fe\}, \\ Fd, Fe, F\{T\{Tb\}, Fc\}, F\{T\{Fe\}, Fc\} \end{array} \right\}$$

In view of $\beta(r_4) = \{Tc, Td\} \in \mathbf{A}_1^F$, $c \in L_{1_{\vec{P}_1}}^{hc} \cap \mathbf{A}_1^T$, and $d \in L \cap \mathbf{A}_1^F$, we obtain the following assignment nogoods:

$$\Theta_{P_1}^{\mathbf{A}_1}(L_1) = \left\{ \begin{array}{l} \{Tf_{\{Fe\}}\}, \{Tf_{\{Tb, Fd\}}\}, \{Tf_{\{Ta, Fe\}}\}, \\ \{Ff_{\{Tc, Td\}}\}, \{Tf_{\{Tb\}}\}, \\ \{Tt_a\}, \{Ft_c\}, \{Tt_d\}, \{Tu_d\} \end{array} \right\}$$

That is, unfounded sets admitted as solutions for $\Omega_{P_1}(L_1) \cup \Theta_{P_1}^{\mathbf{A}_1}(L_1)$ must neither include d (or Tu_d, respectively) nor rely on literals of the form Th_p or $Tf_{\beta(r)}$ for $p \neq c$ or $r \neq r_4$. In fact, $\Omega_{P_1}(L_1) \cup \Theta_{P_1}^{\mathbf{A}_1}(L_1)$ has a single solution as follows:

$$\mathbf{B}_1 = \{Tu_a, Tu_b, Fu_c, Fu_d, Fh_a, Th_c, Fh_d\}$$

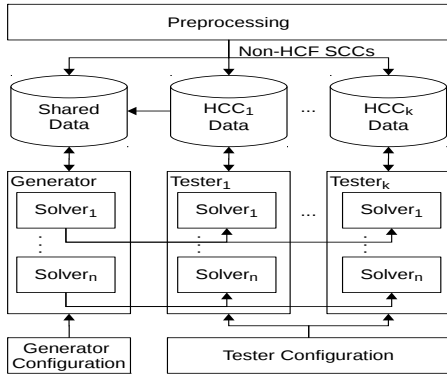$$\cup \{\overline{\sigma} \mid \{\sigma\} \in \Theta_{P_1}^{\mathbf{A}_1}(L_1)\}$$

Figure 1: System architecture of *claspD-2*

This solution yields the unfounded set $\{a, b\}$ for $\vec{P}_1$ wrt. $\mathbf{A}_1$, where the members of $\varepsilon_{\vec{P}_1}(\{a, b\}) = \{r_1, r_4\}$ are satisfied independently of a and b in view of $\boldsymbol{T}c \in \rho(r_1, \{a, b\}) \cap \mathbf{A}_1$ and $\boldsymbol{F}\{\boldsymbol{T}c, \boldsymbol{T}d\} \in \rho(r_4, \{a, b\}) \cap \mathbf{A}_1$.

In general, we have the following one-to-one correspondence between (particular) unfounded sets and solutions.

Proposition 3. *Let P be a program, $\mathbf{A}$ an assignment for Δ_P, $L \in scc_P$, and $\emptyset \subset L' \subseteq L \setminus \mathbf{A}^F$. We have that L' is unfounded for $\vec{P}$ wrt. $\mathbf{A}$ iff*

$$\mathbf{B} = \{\boldsymbol{T}\mathsf{u}_p \mid p \in L'\} \cup \{\boldsymbol{F}\mathsf{u}_p \mid p \in L \setminus L'\}$$
$$\cup \{\boldsymbol{T}\mathsf{h}_p \mid p \in L_{\vec{P}}^{hc} \cap (\mathbf{A}^T \setminus L')\}$$
$$\cup \{\boldsymbol{F}\mathsf{h}_p \mid p \in L_{\vec{P}}^{hc} \setminus (\mathbf{A}^T \setminus L')\} \cup \{\overline{\sigma} \mid \{\sigma\} \in \Theta_P^{\mathbf{A}}(L)\}$$

is the unique solution for $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ such that $\{p \in L \mid \mathsf{u}_p \in \mathbf{B}^T\} = L'$.

The confinement to non-empty unfounded subsets L' of $L \setminus \mathbf{A}^F$ for some $L \in scc_P$ is sufficient for minimality checking because circular positive dependencies cannot spread across multiple SCCs of G_P. Moreover, for any rule $r \in \vec{P}$ such that $bd(r)^+ \cap \mathbf{A}^F \neq \emptyset$, the conjunction nogoods $\Gamma_{\vec{P}}$ require $\boldsymbol{F}\beta(r)$ to belong to any solution for Δ_P, so that false atoms can be subtracted from unfounded sets. In view of Theorem 2, the correctness of minimality checking via unfounded sets within SCCs can be stated as follows.

Theorem 3. *Let P be a program and $\mathbf{A}$ a solution for Δ_P. We have that $\mathbf{A}$ is a solution for Λ_P iff $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ is unsatisfiable for any $L \in scc_P$.*

As noted above, unfounded set checks are tractable for SCCs whose atoms do not occur jointly in any proper disjunction (trivially including singletons like $\{e\} \in scc_{P_1}$). Hence, solving $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ to compute unfounded sets is appropriate for HCCs only. In the course of this, however, component-wise shifting reduces the amount of assignment specifics passed through $\Theta_P^{\mathbf{A}}(L)$, since only atoms from L are considered in addition to variables for rule bodies.

4 The *claspD-2* System

Our approaches to model generation and minimality checking are implemented in the new conflict-driven disjunctive ASP solver *claspD-2*. Its generating and testing solver units rely on Conflict-Driven Constraint Learning (CDCL; [Marques-Silva and Sakallah, 1999; Zhang *et al.*, 2001]), as originally devised for SAT. CDCL incorporates deterministic (unit) propagation, heuristic decisions, conflict resolution to derive and record a new nogood from any assignment containing some nogood, and backjumping to recover after a conflict.

The generator(s) in *claspD-2* combine propagation via completion nogoods with unfounded set checks. The latter concentrate on (non-trivial) SCCs, and complex checks by solving $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ are limited to HCCs $L \in scc_P^{hc}$.[1] For scheduling such checks, *claspD-2* distinguishes whether an assignment $\mathbf{A}$ is total, viz. $\mathbf{A}^T \cup \mathbf{A}^F = var(\Delta_P)$, or not. Before some total $\mathbf{A}$ is accepted, the unsatisfiability of $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ needs to be verified for all $L \in scc_P^{hc}$, and thus a *total check* investigating each HCC is performed. Unlike this, *partial checks* are not mandatory for soundness and can be customized in *claspD-2*. The ad hoc strategy applied by default relies on two thresholds: low is initialized with 0 and updated to the (greater) level of a decision where partial checking yields unsatisfiability of $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ for all $L \in scc_P^{hc}$; $high$ is initially ∞ and then updated to the level of a decision where a solution $\mathbf{B}$ for $\Omega_P(L) \cup \Theta_P^{\mathbf{A}}(L)$ such that $\lambda \subseteq \mathbf{A}$ for some $\lambda \in \lambda_{\vec{P}}(\{p \in L \mid \mathsf{u}_p \in \mathbf{B}^T\})$ exhibits a conflict. Given this, a partial check is scheduled for decision level $\lceil \frac{high-low}{2} \rceil$, assuming that roughly half the number of variables unassigned by backjumping from a previous conflict due to an unfounded set will be reassigned.

Figure 1 displays the system architecture of *claspD-2*. Preprocessing a (ground disjunctive) input program P includes the calculation of SCCs along with the component-shifted version $\vec{P}$ of P. This provides the basis for internal data representations of Δ_P and $\Omega_P(L_1), \ldots, \Omega_P(L_k)$ for $scc_P^{hc} = \{L_1, \ldots, L_k\}$. Such data representations are further handled by dedicated solver objects, one generating candidate models and k testers performing unfounded set checks for individual HCCs. Building on the multi-threaded design of the *clasp 2* series [Gebser *et al.*, 2012b], the assembly of a generator along with testers may be reproduced to obtain n (currently up to $n = 64$) threads running in parallel. The n generating and $k \times n$ testing solvers can be separately configured, eg., to specify portfolios of search strategies for model generation and/or minimality checking. Notably, different generators as well as the n testers working on the same HCC share common data, rather than copying it n times.

In addition to multi-threading support, the interplay between generator(s) and tester(s) has been a prime target of enhancements in *claspD-2*. To this end, the unfounded set nogoods $\Omega_P(L)$ are formulated in such a way that they remain invariant under dynamic assignment information passed through $\Theta_P^{\mathbf{A}}(L)$. Since all nogoods in $\Theta_P^{\mathbf{A}}(L)$ are singletons fixing truth values for some variables, their effect can be implemented via *assumptions* [Eén and Sörensson, 2003]. In fact, *claspD-2* utilizes the interface of *clasp* [Gebser *et al.*,

---

[1]The (linear-time) unfounded set detection algorithm of *clasp* (cf. [Gebser *et al.*, 2012a]) allows for identifying sets L' such that $\beta(r) \in \mathbf{A}^F$ applies to all external supports $r \in \varepsilon_{\vec{P}}(L')$. This is sufficient to detect any unfounded subset of a non-disjunctive SCC.

Benchmark	#	basic		partial		pfolio[4]		pfolio[8]		*claspD-1*		*cmodels*	
ConformantPlanning	23	49	(1)	50	(1)	21	(0)	15	(0)	92	(0)	395	(10)
MaximalSatisfiableSet	86	51	(3)	46	(2)	23	(0)	5	(0)	230	(12)	619	(57)
MinimalDiagnosis	58	28	(1)	4	(0)	4	(0)	4	(0)	11	(0)	900	(58)
2QBF	53	276	(14)	278	(15)	190	(7)	148	(4)	568	(32)	577	(32)
Repair	60	79	(4)	44	(2)	11	(0)	8	(0)	354	(20)	900	(60)
StrategicCompanies	32	489	(6)	473	(6)	528	(10)	521	(8)	773	(23)	862	(30)
Average Time (Timeouts)	312	162	(29)	149	(26)	129	(17)	117	(12)	338	(87)	709	(247)

Table 1: Experimental results comparing different configurations of *claspD-2* with *claspD-1* and *cmodels*

2008] for solving under assumptions to handle unfounded set checks via queries to testers in charge. Importantly, the same tester processes all queries from a generator (likewise at total and partial checks) for some HCC, so that conflict nogoods recorded when solving a query can persist and constrain future queries. In turn, an unfounded set L' from a tester always yields some loop nogood $\lambda \in \lambda_{\bar{P}}(L')$ such that $\lambda \subseteq \mathbf{A}$ for the querying generator's assignment $\mathbf{A}$ (thus indicating a conflict) or that λ can be propagated wrt. $\mathbf{A}$. In either case, λ is recorded on the generator side, which may make analogous queries (from any generator) obsolete in the sequel.

5 Experiments

We compare two single- and two multi-threaded configurations of *claspD-2* (R6582[2]) with its predecessor *claspD-1* (1.1.4) and *cmodels* (3.8.2). All these systems accept (ground disjunctive) input programs in *lparse* format [Syrjänen] and utilize CDCL technology for model generation and minimality checking. The experiments were run on a Linux machine with two Quad-Core Xeon E5520 2.27GHz processors, limiting each run to 900 seconds wall-clock time.

Table 1 shows average runtimes in seconds on 312 instances[2] from six classes: *ConformantPlanning* [Tu *et al.*, 2011], *MaximalSatisfiableSet* [Janota and Marques-Silva, 2011], *MinimalDiagnosis* [Gebser *et al.*, 2011b], *2QBF* [Peschiera *et al.*, 2010], *Repair* [Gebser *et al.*, 2010], and *StrategicCompanies* [Calimeri *et al.*, 2013]. Each instance was solved within the time limit by at least one system, while timeouts, given in parentheses, account for 900 seconds. The last row displays average runtimes over all six classes (weighted equally) along with sums of timeouts.

The difference between the single-threaded *claspD-2* configurations "basic" and "partial" is that only the latter performs partial checks for HCCs. We observe some yet moderate performance gains due to partial checks according to our ad hoc strategy sketched above. Parallel portfolios, varying the search strategies of generators along with their testers, with $n = 4$ or $n = 8$ threads in "pfolio[n]" turn out to be quite effective (except for *StrategicCompanies*, where nogood exchange distracts search), especially on *2QBF*. However, these benchmark results can give a first impression only, as neither the scheduling of partial checks nor multi-threaded operation modes of *claspD-2* are fine-tuned at present.

Although *claspD-2* still lacks fine-tuning, the comparison with *claspD-1* and *cmodels* clearly shows advantages of our

new system. For one, they are owed to using state-of-the-art solver units from the *clasp 2* series for model generation and minimality checking. For another, *claspD-2* is the first disjunctive ASP solver in which generators and testers coexist over whole runs, whereas prior approaches rely on rebuilding Boolean constraints to check different assignments.

For a complement, we also ran the backtracking-based solver *dlv* (Dec12) on the instances of *MinimalDiagnosis* and *StrategicCompanies*, whose encodings from [Calimeri *et al.*, 2013] are compatible with *dlv*'s (first-order) input language. Like *cmodels*, *dlv* could not finish any *MinimalDiagnosis* instance within 900 seconds. However, with 7 timeouts and about 522 seconds average runtime, *dlv* was competitive on *StrategicCompanies*, whose instances are randomly generated and not very amenable to conflict-driven learning.

6 Discussion

We introduced an approach to disjunctive ASP solving along with the *claspD-2* system equitably interleaving model generation and minimality checking. Unlike previous unfounded set encodings, our new formulation includes variables to represent relevant parts of candidate assignments to check. By means of assumptions, the truth values of such variables can be fixed to query for unfounded sets wrt. a particular assignment at hand. This enables the reuse of Boolean constraints characterizing unfounded sets, both static ones as well as those recorded from conflicts, in a sequence of queries.

While multi-threaded parallelism has already been exploited to instantiate (disjunctive) programs [Perri *et al.*, 2010], *claspD-2* is the first solver offering shared memory multi-threading in the search for answer sets of disjunctive programs. In this work, we presented the initial version of *claspD-2*, whose fine-tuning for Σ_2^P-hard problem solving is still progressing. For instance, we aim at more elaborate approaches to schedule partial checks than our current ad hoc strategy or the one of *dlv* [Pfeifer, 2004]. A second issue to future work is exploring in how far the search for unfounded sets can be geared towards effectiveness in pruning the space of candidate models. As a side effect of its decision heuristic, the tester of *dlv* [Maratea *et al.*, 2010] identifies $\subseteq$-minimal unfounded sets, but a systematic investigation of unfounded set "optimization" is yet missing. Finally, we envisage a native treatment (without compiling to simpler constructs) of recursive aggregates (cf. [Alviano *et al.*, 2011]) involved in HCCs as a means to increase expressiveness in the future.

---

[2] http://www.cs.uni-potsdam.de/claspD

References

[Alviano *et al.*, 2011] M. Alviano, F. Calimeri, W. Faber, N. Leone, and S. Perri. Unfounded sets and well-founded semantics of answer set programs with aggregates. *Journal of Artificial Intelligence Research*, 42:487–527, 2011.

[Baral, 2003] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge, 2003.

[Ben-Eliyahu and Dechter, 1994] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):53–87, 1994.

[Calimeri *et al.*, 2013] F. Calimeri, G. Ianni, and F. Ricca. The third open answer set programming competition. *Theory and Practice of Logic Programming*, 2013. To appear.

[Drescher *et al.*, 2008] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub. Conflict-driven disjunctive answer set solving. In *Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003.

[Eiter and Gottlob, 1995] 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.

[Gebser *et al.*, 2008] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In *Proc. International Conference on Logic Programming (ICLP'08)*, pages 190–205. Springer, 2008.

[Gebser *et al.*, 2010] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 497–507. AAAI Press, 2010.

[Gebser *et al.*, 2011a] M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011.

[Gebser *et al.*, 2011b] M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2-3):323–360, 2011.

[Gebser *et al.*, 2012a] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *AI Journal*, 187-188:52–89, 2012.

[Gebser *et al.*, 2012b] M. Gebser, B. Kaufmann, and T. Schaub. Multi-threaded ASP solving with clasp. *Theory and Practice of Logic Programming*, 12(4-5):525–545, 2012.

[Janhunen *et al.*, 2006] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[Janota and Marques-Silva, 2011] M. Janota and J. Marques-Silva. On deciding MUS membership with QBF. In *Proc. International Conference on Principles and Practice of Constraint Programming (CP'11)*, pages 414–428. Springer, 2011.

[Koch *et al.*, 2003] C. Koch, N. Leone, and G. Pfeifer. Enhancing disjunctive logic programming systems by SAT checkers. *AI Journal*, 151(1-2):177–212, 2003.

[Lee and Lifschitz, 2003] J. Lee and V. Lifschitz. Loop formulas for disjunctive logic programs. In *Proc. International Conference on Logic Programming (ICLP'03)*, pages 451–465. Springer, 2003.

[Leone *et al.*, 1997] N. Leone, P. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.

[Leone *et al.*, 2006] 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.

[Lierler, 2005] Y. Lierler. Cmodels: SAT-based disjunctive answer set solver. In *Proc. International Conference on Logic Programming and Nonmonotonic Reasoning (LP-NMR'05)*, pages 447–451. Springer, 2005.

[Maratea *et al.*, 2010] M. Maratea, F. Ricca, and P. Veltri. DLV$^{\text{MC}}$: Enhanced model checking in DLV. In *Proc. European Conference on Logics in Artificial Intelligence (JELIA'10)*, pages 365–368. Springer, 2010.

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

[Perri *et al.*, 2010] S. Perri, F. Ricca, and M. Sirianni. A parallel ASP instantiator based on DLV. In *Proc. Workshop on Declarative Aspects of Multicore Programming (DAMP'10)*, pages 73–82. ACM Press, 2010.

[Peschiera *et al.*, 2010] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The seventh QBF solvers evaluation (QBFEVAL'10). In *Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'10)*, pages 237–250. Springer, 2010.

[Pfeifer, 2004] G. Pfeifer. Improving the model generation/checking interplay to enhance the evaluation of disjunctive programs. In *Proc. International Conference on Logic Programming and Nonmonotonic Reasoning (LP-NMR'04)*, pages 220–233. Springer, 2004.

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

[Tu *et al.*, 2011] P. Tu, T. Son, M. Gelfond, and A. Morales. Approximation of action theories and its application to conformant planning. *AI Journal*, 175(1):79–119, 2011.

[Zhang *et al.*, 2001] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proc. International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.