

# <sup>1</sup> SMT-based Answer Set Solver CMODELS(DIFF) <sup>2</sup> (System Description)

<sup>3</sup> **Da Shen**

<sup>4</sup> Department of Computer Science, University of Nebraska at Omaha

<sup>5</sup> [South 67th Street, Omaha, NE 68182, USA]

<sup>6</sup> dashen@unomaha.edu

<sup>7</sup> **Yuliya Lierler**

<sup>8</sup> Department of Computer Science, University of Nebraska at Omaha

<sup>9</sup> [South 67th Street, Omaha, NE 68182, USA]

<sup>10</sup> ylierler@unomaha.edu

<sup>11</sup>  <https://orcid.org/0000-0002-6146-623X>

## <sup>12</sup> —— Abstract ——

<sup>13</sup> Many answer set solvers utilize Satisfiability solvers for search. SMT solvers extend Satisfiability  
<sup>14</sup> solvers. This paper presents the CMODELS(DIFF) system that uses SMT solvers to find answer sets  
<sup>15</sup> of a logic program. Its theoretical foundation is based on Niemala's characterization of answer  
<sup>16</sup> sets of a logic program via so called level rankings. The comparative experimental analysis  
<sup>17</sup> demonstrates that CMODELS(DIFF) is a viable answer set solver.

<sup>18</sup> **2012 ACM Subject Classification** Computing methodologies → Logic programming and answer  
<sup>19</sup> set programming, Software and its engineering → Constraint and logic languages, Theory of  
<sup>20</sup> computation → Constraint and logic programming

<sup>21</sup> **Keywords and phrases** answer set programming, satisfiability modulo theories, constraint satis-  
<sup>22</sup> faction processing

<sup>23</sup> **Digital Object Identifier** 10.4230/OASIcs.CVIT.2016.23

## <sup>24</sup> **1** Introduction

<sup>25</sup> This paper describes a new answer set solver CMODELS(DIFF). Its theoretical foundation lies  
<sup>26</sup> on the generalizations of Niemela's ideas. Niemela [19] characterized answer sets of a normal  
<sup>27</sup> logic program as models of a propositional formula called program's completion that satisfy  
<sup>28</sup> "level ranking" requirements. In this sense, this system is a close relative of an earlier answer  
<sup>29</sup> set solver LP2DIFF developed by Janhunen et al. [10]. Yet, LP2DIFF only accepts programs  
<sup>30</sup> of a very restricted form. For example, neither choice rules nor aggregate expressions are  
<sup>31</sup> allowed. Answer set solver CMODELS(DIFF) permits such important modeling constructs in  
<sup>32</sup> its input. Also, unlike LP2DIFF, the CMODELS(DIFF) system is able to generate multiple  
<sup>33</sup> solutions.

<sup>34</sup> The CMODELS(DIFF) system follows the tradition of answer set solvers such as ASSAT [16]  
<sup>35</sup> and CMODELS [11]. In place of designing specialized search procedures targeting logic  
<sup>36</sup> programs, these tools compute a program's completion and utilize Satisfiability solvers [9]  
<sup>37</sup> – systems for finding satisfying assignments for propositional formulas – for search. Since  
<sup>38</sup> not all models of a program's completion are answer sets of a program, both ASSAT and  
<sup>39</sup> CMODELS implement specialized procedures (based on loop formulas [16]) to weed out such  
<sup>40</sup> models. Satisfiability Modulo Theory (SMT) solvers [2] extend Satisfiability solvers. They  
<sup>41</sup> process formulas that go beyond propositional logic and may contain, for example, integer  
<sup>42</sup> linear expressions. The CMODELS(DIFF) system utilizes this fact and translates a logic



© Da Shen and Yuliya Lierler;  
licensed under Creative Commons License CC-BY

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:14

OpenAccess Series in Informatics

 OASIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

43 program into an SMT formula so that any model of this formula corresponds to an answer  
 44 set of the program. It then uses SMT solvers for search. Unlike CMODELS or ASSAT, the  
 45 CMODELS(DIFF) system does not need an additional step to weed out unwanted models. Also,  
 46 it utilizes SMT-LIB – a standard input language of SMT solvers [1] – to interface with these  
 47 systems. This makes its architecture open towards new developments in the realm of SMT  
 48 solving. There is practically no effort involved in incorporating a new SMT system into the  
 49 CMODELS(DIFF) implementation.

50 Creation of the CMODELS(DIFF) system was inspired by the development of recent  
 51 constraint answer set programming solver EZSMT [21] that utilizes SMT solvers for finding  
 52 solutions for "tight" constraint answer set programs. On the one hand, CMODELS(DIFF)  
 53 restricts its attention to pure answer set programs. On the other hand, it goes beyond  
 54 tight programs. In the future, we will extend CMODELS(DIFF) to accept non-tight constraint  
 55 answer set programs. The theory developed in this work paves the way for such an extension.

56 Lierler and Susman [13] demonstrate that SMT formulas are strongly related to constraint  
 57 programs [17]. Many efficient constraint solvers<sup>1</sup> exist. Majority of these systems focus  
 58 on finite-domain constraint problems. The theoretical contributions of this work provide a  
 59 foundation for developing a novel constraint-solver-based method in processing logic programs.  
 60 Currently, CMODELS(DIFF) utilizes SMT-LIB to interface with SMT solvers. By producing  
 61 output in MINIZINC – a standard input language of constraint solvers [18] – in place of  
 62 SMT-LIB, CMODELS(DIFF) will become a constraint-based answer set solver. This is another  
 63 direction of future work.

64 The outline of the paper is as follows. We start by reviewing the concepts of a logic  
 65 program, a completion, tightness and an SMT logic SMT(IL). We then present a key concept  
 66 of this work, namely, a level ranking; and state theoretical results. Section 4 presents  
 67 transformations from logic programs to SMT(IL) by means of variants of level rankings.  
 68 After that, we introduce the architecture of the CMODELS(DIFF) system and conclude with  
 69 comparative experimental analysis.

## 70 **2 Preliminaries**

71 A *vocabulary* is a finite set of propositional symbols also called atoms. As customary, a *literal*  
 72 is an atom  $a$  or its negation, denoted  $\neg a$ . A (*propositional*) *logic program*, denoted by  $\Pi$ ,  
 73 over vocabulary  $\sigma$  is a finite set of *rules* of the form

$$74 \quad a \leftarrow b_1, \dots, b_\ell, \text{ not } b_{\ell+1}, \dots, \text{ not } b_m, \text{ not not } b_{m+1}, \dots, \text{ not not } b_n \quad (1)$$

75 where  $a$  is an atom over  $\sigma$  or  $\perp$ , and each  $b_i$ ,  $1 \leq i \leq n$ , is an atom or symbol  $\top$  and  $\perp$  in  $\sigma$ .  
 76 Sometimes we use the abbreviated form of rule (1)

$$77 \quad a \leftarrow B \quad (2)$$

78 where  $B$  stands for the right hand side of an arrow in (1) and is also called a *body*. We  
 79 identify rule (1) with the propositional formula

$$80 \quad b_1 \wedge \dots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \dots \wedge \neg b_m \wedge \neg \neg b_{m+1} \wedge \dots \wedge \neg \neg b_n \rightarrow a \quad (3)$$

81 and  $B$  with the propositional formula

$$82 \quad b_1 \wedge \dots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \dots \wedge \neg b_m \wedge \neg \neg b_{m+1} \wedge \dots \wedge \neg \neg b_n. \quad (4)$$

---

<sup>1</sup> <http://www.minizinc.org/>

83 Note that (i) the order of terms in (4) is immaterial, (ii) *not* is replaced with classical  
 84 negation ( $\neg$ ), and (iii) comma is replaced with conjunction ( $\wedge$ ). When the body is empty it  
 85 corresponds to the empty conjunction or  $\top$ . Expression  $b_1 \wedge \dots \wedge b_\ell$  in formula (4) is referred  
 86 to as the *positive* part of the body and the remainder of (4) as the *negative* part of the body.

87 The expression  $a$  is the *head* of the rule. When  $a$  is  $\perp$ , we often omit it and say that the  
 88 head is empty. We denote the set of nonempty heads of rules in  $\Pi$  by  $hd(\Pi)$ . We call a rule  
 89 whose body is empty a *fact*. In such cases, we drop the arrow. We sometimes may identify a  
 90 set  $X$  of atoms with the set of facts  $\{a. \mid a \in X\}$ .

91 We say that a set  $X$  of atoms *satisfies* a rule (1) if  $X$  satisfies a formula (3). The reduct  
 92  $\Pi^X$  of a program  $\Pi$  relative to a set  $X$  of atoms is obtained by first removing all rules (1)  
 93 such that  $X$  does not satisfy its negative part  $\neg b_{\ell+1} \wedge \dots \wedge \neg b_m \wedge \neg \neg b_{m+1} \wedge \dots \wedge \neg \neg b_n$  and  
 94 replacing all of its remaining rules with  $a \leftarrow b_1, \dots, b_\ell$ . A set  $X$  of atoms is an *answer set*, if  
 95 it is a minimal set that satisfies all rules of  $\Pi^X$  [15].

96 Ferraris and Lifschitz [6] show that a choice rule  $\{a\} \leftarrow B$  can be seen as an abbreviation  
 97 for a rule  $a \leftarrow \text{not not } a, B$ . We adopt this abbreviation here. Choice rules were introduced  
 98 in [20] and are commonly used in answer set programming languages.

99 It is customary for a given vocabulary  $\sigma$ , to identify a set  $X$  of atoms over  $\sigma$  with (i)  
 100 a complete and consistent set of literals over  $\sigma$  constructed as  $X \cup \{\neg a \mid a \in \sigma \setminus X\}$ , and  
 101 respectively with (ii) an assignment function or interpretation that assigns truth value *true*  
 102 to every atom in  $X$  and *false* to every atom in  $\sigma \setminus X$ .

103 Consider sample programs listed in Figure 1. Program  $\Pi_1$  has two answer sets, namely,  
 104  $\{a, c\}$  and an empty set. Program  $\Pi_2$  has two answer sets:  $\{a, b, c\}$  and an empty set.

		$Comp(\Pi_1)$	$Comp(\Pi_2)$
$\Pi_1$	$\Pi_2$		
$\{c\}.$	$\{c\}.$	$\neg \neg c \rightarrow c.$	$\neg \neg c \rightarrow c.$
$a \leftarrow c.$	$a \leftarrow c.$	$c \rightarrow a.$	$c \rightarrow a.$
$a \leftarrow b.$		$c \rightarrow \neg \neg c.$	$b \rightarrow a.$
$b \leftarrow a.$		$a \rightarrow c.$	$a \rightarrow b.$

Figure 1 Sample programs and their completions.

## 105 Completion and Tightness

106 Let  $\sigma$  be a vocabulary and  $\Pi$  be a program over  $\sigma$ . For every atom  $a$  in  $\Pi$ , by  $Bodies(\Pi, a)$   
 107 we denote the set composed of the bodies  $B$  appearing in the rules of the form  $a \leftarrow B$  in  $\Pi$ .  
 108 The *completion* of  $\Pi$  [3], denoted by  $Comp(\Pi)$ , is the set of classical formulas that consists of  
 109 the rules (1) in  $\Pi$  (recall that we identify rule (1) with implication (3)) and the implications

$$110 \quad a \rightarrow \bigvee_{a \leftarrow B \in \Pi} B \quad (5)$$

111 for all atoms  $a$  in  $\sigma$ . When set  $Bodies(\Pi, a)$  is empty, the implication (5) has the form  
 112  $a \rightarrow \perp$ . When a rule (2) is a fact  $a.$ , then we identify this rule with the unit clause  $a$ .

113 For example, completions of programs  $\Pi_1$  and  $\Pi_2$  are presented in Figure 1.

114 For the large class of logic programs, called *tight*, their answer sets coincide with models  
 115 of their completion [5, 4]. This is the case for program  $\Pi_1$  (we illustrate that  $\Pi_1$  is tight,  
 116 shortly). Yet, for non-tight programs, every answer set is a model of completion but not

117 necessarily the other way around. For instance, set  $\{a, b\}$  is a model of  $Comp(\Pi_2)$ , but not  
 118 an answer set of  $\Pi_2$ . It turns out that  $\Pi_2$  is not tight.

119 Tightness is a syntactic condition on a program that can be verified by means of program's  
 120 dependency graph. The *dependency graph* of  $\Pi$  is the directed graph  $G$  such that

- 121   ■ the nodes of  $G$  are the atoms occurring in  $\Pi$ , and
- 122   ■ for every rule (1) in  $\Pi$  whose head is an atom,  $G$  has an edge from atom  $a$  to each atom  
 123     $b_1, \dots, b_\ell$ .

124 A program is called *tight* if its dependency graph is acyclic.

125 For example, the dependency graph of program  $\Pi_1$  consists of two nodes, namely,  $a$  and  
 126  $c$ , and a single edge from  $a$  to  $c$ . This graph is acyclic and hence  $\Pi_1$  is tight. On the other  
 127 hand, it is easy to see that the graph of  $\Pi_2$  is not acyclic.

## 128 Logic SMT(IL)

129 We now introduce the notion of Satisfiability Modulo Theory (SMT) [2] for the case when  
 130 Linear Integer Arithmetic is a considered theory. We denote this SMT instance by SMT(IL).

131 Let  $\sigma$  be a vocabulary and  $\chi$  be a finite set of integer variables. The set of *atomic formulas*  
 132 of SMT(IL) consists of propositions in  $\sigma$  and linear constraints of the form

$$133 \quad a_1 x_1 \pm \dots \pm a_n x_n \bowtie a_{n+1} \quad (6)$$

134 where  $a_1, \dots, a_{n+1}$  are integers and  $x_1, \dots, x_n$  are variables in  $\chi$ ,  $\pm$  stands for  $+$  or  $-$ , and  $\bowtie$   
 135 belongs to  $\{<, >, \leq, \geq, =, \neq\}$ . When  $a_i = 1$  ( $1 \leq i \leq n$ ) we may omit it from the expression.  
 136 The set of SMT(IL) formulas is the smallest set that contains the atomic formulas and is  
 137 closed under  $\neg$  and conjunction  $\wedge$ . Other connectives such as  $\top, \perp, \vee, \rightarrow$ , and  $\leftrightarrow$  can be  
 138 defined in terms of  $\neg$  and  $\wedge$  as customary.

139 A valuation  $\tau$  consists of a pair of functions

- 140   ■  $\tau_\sigma : \sigma \rightarrow \{\text{true}, \text{false}\}$  and
- 141   ■  $\tau_\chi : \chi \rightarrow \mathbb{Z}$ , where  $\mathbb{Z}$  is the set of integers.

142 A valuation interprets all SMT(IL) formulas by defining

- 143   ■  $\tau(p) = \tau_\sigma(p)$  when  $p \in \sigma$ ,
- 144   ■  $\tau(a_1 x_1 \pm \dots \pm a_n x_n \bowtie a_{n+1}) = \text{true}$  iff  $a_1 \tau_\chi(x_1) \pm \dots \pm a_n \tau_\chi(x_n) \bowtie a_{n+1}$  holds,  
 145 and applying the usual rules for the Boolean connectives.

146 We say that an SMT(IL) formula  $\Phi$  is satisfied by a valuation  $\tau$  when  $\tau(\Phi) = \text{true}$ . A set  
 147 of SMT(IL) formulas is satisfied by a valuation when every formula in the set is satisfied by  
 148 this valuation. We call a valuation that satisfies an SMT(IL) formula a *model*.

## 149 3 Level Rankings

150 Niemela [19] characterized answer sets of "normal" logic programs in terms of "level rankings."  
 151 *Normal* programs consist of rules of the form (1), where  $n = m$  and  $a$  is an atom. Lierler and  
 152 Susman [13] generalized the concept of level ranking to programs considered in this paper  
 153 that include choice rules and denials (rules with empty head).

154 By  $\mathbb{N}$  we denote the set of natural numbers. For a rule (2), by  $B^+$  we denote its positive  
 155 part and sometimes identify it with the set of atoms that occur in it, i.e.,  $\{b_1, \dots, b_l\}$ . For a  
 156 program  $\Pi$ , by  $At(\Pi)$  we denote the set of atoms occurring in it.

157 ► **Definition 1.** For a logic program  $\Pi$  and a set  $X$  of atoms over  $At(\Pi)$ , a function  $lr$ :  
 158  $X \rightarrow \mathbb{N}$  is a *level ranking* of  $X$  for  $\Pi$  when for each  $a \in X$ , there is  $B$  in  $Bodies(\Pi, a)$  such  
 159 that  $X$  satisfies  $B$  and for every  $b \in B^+$  it holds that  $lr(a) - 1 \geq lr(b)$ .

160 Niemela [19] observed that for a normal logic program, a model  $X$  of its completion is also  
 161 its answer set when there is a level ranking of  $X$  for the program. Lierler and Susman [13]  
 162 generalized this result to programs with double negation *not not*:

163 ► **Theorem 2** (Theorem 1 [13]). *For a program  $\Pi$  and a set  $X$  of atoms that is a model of  
 164 its completion  $Comp(\Pi)$ ,  $X$  is an answer set of  $\Pi$  if and only if there is a level ranking of  $X$   
 165 for  $\Pi$ .*

166 The nature of a level ranking is such that there is an infinite number of level rankings for  
 167 the same answer set of a program. Theorem below illustrates that we can add a *single* linear  
 168 constraint to limit the number of level rankings by utilizing the size of a program.

169 ► **Theorem 3.** *For a logic program  $\Pi$  and its answer set  $X$ , we can always construct a level  
 170 ranking of  $X$  for  $\Pi$  such that, for every  $a \in X$ ,  $lr(a) \leq |At(\Pi)|$ .*

171 **Proof.** Since there is an answer set  $X$ , by Theorem 2 there exists some level ranking  $lr'$  of  
 172  $X$  for  $\Pi$ . Then, we can always use the level ranking  $lr'$  to construct a level ranking  $lr$  of  $X$   
 173 for  $\Pi$  such that, for every  $a \in X$ ,  $lr(a) \leq |At(\Pi)|$ . Below we describe the method.

For an integer  $y$ , by  $s(y)$  we denote the following set of atoms

$$\{a \mid a \in X, lr'(a) = y\}.$$

Let  $Y$  be the set of integers so that

$$\{y \mid a \in X, lr'(a) = y\}.$$

174 Let  $Y^s$  denote the sorted list  $[y_1, \dots, y_k]$  constructed from all integers of  $Y$ , such that  
 175  $y_1 < y_2 < \dots < y_k$ . Note that  $y_i > y_j$  if and only if  $i > j$ . Obviously,  $|Y| \leq |At(\Pi)|$ . Thus,  
 176  $k \leq |At(\Pi)|$ . For every element  $y_i$  in  $Y^s$  and every atom  $a \in s(y_i)$ , we assign  $lr(a) = i$ .  
 177 Consequently,  $lr(a) \leq |At(\Pi)|$ .

178 Now we prove that  $lr$  is indeed a level ranking. According to the definition of  $lr'$ , for each  
 179 atom  $a \in X$ , there exists  $B$  in  $Bodies(\Pi, a)$  such that  $X$  satisfies  $B$  and for every  $b \in B^+$   
 180 it holds that  $lr'(a) - 1 \geq lr'(b)$ . We show that  $lr(a) - 1 \geq lr(b)$  also holds for each  $b$  in this  
 181  $B^+$ . Atoms  $a, b$  belong to some sets  $s(y_{k_a})$  and  $s(y_{k_b})$  respectively, where  $k_a, k_b \leq k$ . By  
 182 the definition of  $s(\cdot)$ ,  $y_{k_a} = lr'(a)$  and  $y_{k_b} = lr'(b)$ . Since  $lr'(a) > lr'(b)$ ,  $y_{k_a} > y_{k_b}$ . Since  
 183 for any  $i$  and  $j$ ,  $y_i > y_j$  if and only if  $i > j$ , we derive that  $k_a > k_b$ . By the construction of  
 184  $lr$ ,  $lr(a) = k_a$  and  $lr(b) = k_b$ . Consequently,  $lr(a) - 1 \geq lr(b)$  also holds. Thus,  $lr$  is a level  
 185 ranking by definition. ◀

### 186 Strong level ranking

187 Niemela [19] introduced the concept of a strong level ranking so that only one strong level  
 188 ranking exists for an answer set. It is obviously stricter than the condition captured in  
 189 Theorem 3. Yet, the number of linear constraints in formulating the conditions of strong  
 190 level ranking is substantially greater. We now generalize the concept of a strong level ranking  
 191 to the case of logic programs considered here and then state the formal result on the relation  
 192 of answer sets and strong level rankings.

193 ► **Definition 4.** For a logic program  $\Pi$  and a set  $X$  of atoms over  $At(\Pi)$ , a function  $lr$ :  
 194  $X \rightarrow \mathbb{N}$  is a *strong level ranking* of  $X$  for  $\Pi$  when  $lr$  is a level ranking and for each  $a \in X$  the  
 195 following conditions hold:

196 1. If there is  $B$  in  $Bodies(\Pi, a)$  such that  $X$  satisfies  $B$  and  $B^+$  is empty, then  $lr(a) = 1$ .

197 2. For every  $B$  in  $Bodies(\Pi, a)$  such that  $X$  satisfies  $B$  and  $B^+$  is not empty, there is at  
198 least one  $b \in B^+$  such that  $lr(b) + 1 \geq lr(a)$ .

199 ▶ **Theorem 5.** *For a program  $\Pi$  and a set  $X$  of atoms that is a model of its completion  
200  $Comp(\Pi)$ ,  $X$  is an answer set of  $\Pi$  if and only if there is a strong level ranking of  $X$  for  $\Pi$ .*

**Proof.** This proof follows the argument provided for Theorem 2 in [19], but respects the terminology used here. We start by defining an operator  $T_\Pi(I)$  for a program  $\Pi$  and a set  $I$  over  $At(\Pi) \cup \perp$  as follows:

$$T_\Pi(I) = \{a \mid a \leftarrow B \in \Pi, I \text{ satisfies } B\}.$$

For this operator we define

$$T_\Pi \uparrow 0 = \emptyset,$$

and for  $i = 0, 1, 2, \dots$

$$T_\Pi \uparrow (i+1) = T_\Pi(T_\Pi \uparrow i).$$

201 Left-to-right: Assume  $X$  is an answer set of  $\Pi$ . We can construct a strong level ranking  $lr$   
202 of  $X$  for  $\Pi$  using the  $T_{\Pi^X}(\cdot)$  operator. As  $X$  is an answer set of  $\Pi$ , we know that  $X = T_{\Pi^X} \uparrow \omega$   
203 and for each  $a \in X$  there is a unique  $i$  such that  $a \in T_{\Pi^X} \uparrow i$ , but  $a \notin T_{\Pi^X} \uparrow (i-1)$ . Let  
204  $lr(a) = i$ . We now illustrate that  $lr$  is indeed a strong level ranking.

205 First, we illustrate that  $lr$  is a level ranking. For  $a \in X$  there is a rule  $a \leftarrow B$  of the  
206 form (1) such that  $a \leftarrow b_1, \dots, b_l \in \Pi^X$  and  $T_{\Pi^X} \uparrow (i-1)$  satisfies  $b_1 \wedge \dots \wedge b_l$ . Consequently,  
207 for every  $b_j$  in  $\{b_1, \dots, b_l\}$ ,  $lr(b_j) \leq i-1$ . Thus,  $lr(a) - 1 \geq lr(b_j)$ . Also, from the way the  
208 reduct is constructed, it follows that  $X$  satisfies body  $B$  of rule  $a \leftarrow B$ .

209 Second, we show that Condition 1 of the definition of strong level ranking holds for  $lr$ .  
210 If there is  $a \leftarrow B \in \Pi$  such that  $X$  satisfies  $B$  and  $B^+$  is empty, then  $a \leftarrow \top$  is in  $\Pi^X$ . By  
211 definition of the  $T_{\Pi^X}(\cdot)$  operator,  $a \in T_{\Pi^X} \uparrow 1$ . Consequently,  $lr(a) = 1$  holds.

212 Third, we demonstrate that Condition 2 holds for  $lr$ . For  $a \in X$ , by the construction of  $lr$   
213 using the  $T_{\Pi^X}(\cdot)$  operator we know that there is a unique  $i$  such that  $lr(a) = i$ ,  $a \in T_{\Pi^X} \uparrow i$ ,  
214 but  $a \notin T_{\Pi^X} \uparrow (i-1)$ . Proof by contradiction. Assume that there is a rule  $a \leftarrow B \in \Pi$   
215 such that  $X$  satisfies  $B$  and  $B^+$  is not empty, but for all  $b \in B^+$ ,  $lr(b) + 1 < lr(a)$  holds.  
216 Then for all  $b \in B^+$ ,  $lr(b) < lr(a) - 1$ . Thus,  $lr(b) < i - 1$ . It follows that all  $b \in B^+$   
217 belong to  $T_{\Pi^X} \uparrow (i-2)$ . Hence, by the definition of  $T_{\Pi^X}(\cdot)$  operator,  $a \in T_{\Pi^X} \uparrow (i-1)$ ,  
218 which contradicts that  $a \notin T_{\Pi^X} \uparrow (i-1)$ . Thus, there is at least one  $b \in B^+$  such that  
219  $lr(b) + 1 \geq lr(a)$ .

220 Right-to-left: Assume that there is a strong level ranking of  $X$  for  $\Pi$ . By the definition,  
221 it is also a level ranking. Recall that  $X$  is a model of  $Comp(\Pi)$ . By Theorem 2,  $X$  is an  
222 answer set of  $\Pi$ . ◀

### 223 SCC level ranking

224 Niemela [19] illustrated how one can utilize the structure of the dependency graph corresponding  
225 to a normal program to reduce the number of linear constraints in capturing conditions  
226 similar to these of level ranking. We now generalize these results to logic programs with  
227 doubly negated atoms and denials.

228 Recall that a strongly connected component of a directed graph is a maximal set  $V$  of  
229 nodes such that each pair of nodes in  $V$  is reachable from each other. We call a set of atoms  
230 in a program  $\Pi$  a *strongly connected component* (SCC) of  $\Pi$  when it is a strongly connected  
231 component in the dependency graph of  $\Pi$ . The SCC including an atom  $a$  is denoted by  
232  $SCC(a)$ . A *non-trivial* SCC is an SCC that consists of at least two atoms. We denote the  
233 set of atoms in all non-trivial SCCs of  $\Pi$  by  $NT(\Pi)$ .

234 ► **Definition 6.** For a logic program  $\Pi$  and a set  $X$  of atoms over  $At(\Pi)$ , a function  $lr: X \cap NT(\Pi) \rightarrow \mathbb{N}$  is a *SCC level ranking* of  $X$  for  $\Pi$  when for each  $a \in X \cap NT(\Pi)$ , there is  $B$  in  $Bodies(\Pi, a)$  such that  $X$  satisfies  $B$  and for every  $b \in B^+ \cap SCC(a)$  it holds that  $lr(a) - 1 \geq lr(b)$ .

238 The byproduct of the definition of SCC level rankings is that for tight programs SCC  
239 level ranking trivially exists since it is a function whose domain is empty. Thus no linear  
240 constraints are produced.

241 ► **Theorem 7.** For a program  $\Pi$  and a set  $X$  of atoms that is a model of its completion  
242  $Comp(\Pi)$ ,  $X$  is an answer set of  $\Pi$  if and only if there is an SCC level ranking of  $X$  for  $\Pi$ .

243 This is a generalization of Theorem 4 in [19]. Its proof follows the lines of the proof presented  
244 there with the reference to Theorem 2.

245 ► **Theorem 8.** For a satisfiable logic program  $\Pi$  and its answer set  $X$ , we can always  
246 construct an SCC level ranking of  $X$  for  $\Pi$  such that, for every  $a \in X$ ,  $lr(a) \leq |SCC(a)|$ .

247 This theorem can be proved by applying the similar argument as in the proof of Theorem 3  
248 to each SCC. This result allows us to set minimal upper bounds for  $lr(a)$  in order to reduce  
249 search space.

250 Further, Niemela [19] introduces the concept of strong SCC level ranking and states a  
251 similar result to Theorem 7 for that concept. It is straightforward to generalize these results  
252 to logic programs considered here.

## 253 4 From Logic Programs to SMT(IL)

254 In this section we present a mapping from a logic program to SMT(IL) such that the models  
255 of a constructed SMT(IL) theory are in one-to-one correspondence with answer sets of the  
256 program. Thus, any SMT solver capable of processing SMT(IL) expressions can be used to  
257 find answer sets of logic programs. The developed mappings generalize the ones presented by  
258 Niemela [19].

259 For a rule  $a \leftarrow B$  of the form (1), the auxiliary atom  $\beta_B$ , equivalent to its body, is defined  
260 as

$$261 \quad \beta_B \leftrightarrow b_1 \wedge \dots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \dots \wedge \neg b_m \wedge b_{m+1} \wedge \dots \wedge b_n \quad (7)$$

262 When the body of a rule consist of a single element, no auxiliary atom is introduced (the  
263 single element itself serves the role of an auxiliary atom).

264 Let  $\Pi$  be a program. We say that an atom  $a$  is a *head atom* in  $\Pi$  if it is the head of some  
265 rule. Any atom  $a$  in  $\Pi$  such that

266 ■ it is a head atom, or  
267 ■ it occurs in some positive part of the body of some rule whose head is an atom,  
268 we associate with an integer variable denoted by  $lr_a$ . We call such variables level ranking  
269 variables. For each head atom  $a$  in  $\Pi$ , we construct an SMT(IL) formula

$$270 \quad a \rightarrow \bigvee_{a \leftarrow B \in \Pi} \left( \beta_B \wedge \bigwedge_{b \in B^+} lr_a - 1 \geq lr_b \right). \quad (8)$$

271 We call the conjunction of formulas (8) for the head atoms in program  $\Pi$  a *level ranking  
272 formula* of  $\Pi$ .

273 For example, the level ranking formula of program  $\Pi_2$  in Figure 1 follows

$$274 \quad (c \rightarrow \neg \neg c) \wedge (a \rightarrow (c \wedge lr_a - 1 \geq lr_c) \vee (b \wedge lr_a - 1 \geq lr_b)) \wedge (b \rightarrow a \wedge lr_b - 1 \geq lr_a). \quad (9)$$

275 ► **Theorem 9.** For a logic program  $\Pi$  and the set  $F$  of SMT(IL) formulas composed of  
 276  $\text{Comp}(\Pi)$  and a level ranking formula of  $\Pi$

277 1. If a set  $X$  of atoms is an answer set of  $\Pi$ , then there is a satisfying valuation  $\tau$  for  $F$   
 278 such that  $X = \{a \mid a \in \text{At}(\Pi) \text{ and } \tau(a) = \text{true}\}$ .  
 279 2. If valuation  $\tau$  is satisfying for  $F$ , then the set  $\{a \mid a \in \text{At}(\Pi) \text{ and } \tau(a) = \text{true}\}$  is an  
 280 answer set for  $\Pi$ .

281 This is a generalization of Theorem 6 in [19]. Its proof follows the lines of the proof presented  
 282 there with the reference to Theorem 2.

283 **SCC level ranking**

284 For each atom  $a$  in the set  $NT(\Pi)$ , we introduce an auxiliary atom  $ext_a$ . If there exists some  
 285 rule  $a \leftarrow B$  in  $\Pi$  such that  $B^+ \cap SCC(a) = \emptyset$ , then we construct an SMT(IL) formula

$$286 \quad ext_a \leftrightarrow \bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \cap SCC(a) = \emptyset} \beta_B; \quad (10)$$

287 otherwise, we construct a formula

$$288 \quad \neg ext_a. \quad (11)$$

289 We also introduce an SMT(IL) formula:

$$290 \quad a \rightarrow ext_a \vee \bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \cap SCC(a) \neq \emptyset} (\beta_B \wedge \bigwedge_{b \in B^+ \cap SCC(a)} lr_a - 1 \geq lr_b). \quad (12)$$

291 We call the conjunction of formulas (10), (11) and (12) a *SCC level ranking formula* of  $\Pi$ .

292 For instance,  $NT(\Pi_1)$  is empty, so we introduce no SCC level ranking formula for program  
 293  $\Pi_1$ . The SCC level ranking formula of program  $\Pi_2$  follows

$$294 \quad (ext_a \leftrightarrow c) \wedge \neg ext_b \wedge (a \rightarrow ext_a \vee (b \wedge lr_a - 1 \geq lr_b)) \wedge (b \rightarrow ext_b \vee (a \wedge lr_b - 1 \geq lr_a)). \quad (13)$$

295 The claim of Theorem 9 holds also when we replace *a level ranking formula of  $\Pi$*  with *an*  
 296 *SCC level ranking formula of  $\Pi$*  in its statement.

297 **Strong level ranking**

298 For each rule  $a \leftarrow B$  in program  $\Pi$  we construct an SMT(IL) formula

$$299 \quad \begin{aligned} a \wedge \beta_B \rightarrow lr_a = 1 & \quad \text{when } B^+ = \emptyset, \\ a \wedge \beta_B \rightarrow \bigvee_{b \in B^+} lr_b + 1 \geq lr_a & \quad \text{otherwise.} \end{aligned} \quad (14)$$

300 We call the conjunction of formulas (8) and (14) a *strong level ranking formula* of  $\Pi$ .

For example, the strong level ranking formula of program  $\Pi_2$  is a conjunction of formula (9)  
 and formula

$$(c \wedge \neg \neg c \rightarrow lr_c = 1) \wedge (a \wedge c \rightarrow lr_c + 1 \geq lr_a) \wedge \\ (a \wedge b \rightarrow lr_b + 1 \geq lr_a) \wedge (b \wedge a \rightarrow lr_a + 1 \geq lr_b).$$

301 We now state a similar result to Theorem 9 that makes an additional claim on one-to-one  
 302 correspondence between the models of a constructed SMT(IL) formula with the use of strong  
 303 level ranking formula and answer sets of a program.

304 ► **Theorem 10.** For a logic program  $\Pi$  and the set  $F$  of SMT(IL) formulas composed of  
 305  $\text{Comp}(\Pi)$  and a strong level ranking formula of  $\Pi$

306 1. If a set  $X$  of atoms is an answer set of  $\Pi$ , then there is a satisfying valuation  $\tau$  for  $F$   
 307 such that  $X = \{a \mid a \in \text{At}(\Pi) \text{ and } \tau(a) = \text{true}\}$ .

308 2. If valuation  $\tau$  is satisfying for  $F$ , then the set  $\{a \mid a \in \text{At}(\Pi) \text{ and } \tau(a) = \text{true}\}$  is an  
 309 answer set for  $\Pi$ .

310 3. If valuations  $\tau$  and  $\tau'$  satisfy  $F$  and are distinct, then

$$\{a \mid a \in \text{At}(\Pi) \text{ and } \tau(a) = \text{true}\} \neq \{a \mid a \in \text{At}(\Pi) \text{ and } \tau'(a) = \text{true}\}.$$

310 **Strong SCC level ranking**

311 For each atom  $a \in NT(\Pi)$ , we construct a formula

$$312 \quad ext_a \rightarrow lr_a = 1, \quad (15)$$

313 and for each rule  $a \leftarrow B$  such that  $B^+ \cap SCC(a) \neq \emptyset$ , we introduce a formula

$$314 \quad a \wedge \beta_B \rightarrow \bigvee_{b \in B^+ \cap SCC(a)} lr_b + 1 \geq lr_a. \quad (16)$$

315 We call the conjunction of formulas (10), (11), (12), (15) and (16) a *strong SCC level ranking*  
 316 *formulas* of  $\Pi$ .

For instance,  $NT(\Pi_1)$  is empty, so we introduce no strong SCC level ranking formula for program  $\Pi_1$ . The strong SCC level ranking formula of program  $\Pi_2$  is a conjunction of formula (13) and formula

$$(ext_a \rightarrow lr_a = 1) \wedge (ext_b \rightarrow lr_b = 1) \wedge (a \wedge b \rightarrow lr_b + 1 \geq lr_a) \wedge (b \wedge a \rightarrow lr_a + 1 \geq lr_b).$$

317 The claim of Theorem 10 holds also when we replace a *strong level ranking formula* of  $\Pi$   
 318 with a *strong SCC level ranking formula* of  $\Pi$  in its statement.

319 **5 The CMODELS(DIFF) system**

320 We are now ready to describe the the CMODELS(DIFF)<sup>2</sup> system in detail. It is an extension of  
 321 the CMODELS [11] system. Figure 2 illustrates the pipeline architecture of CMODELS(DIFF).  
 322 This system takes an arbitrary (tight or non-tight) logic program in the language supported  
 323 by CMODELS as an input. These logic programs may contain such features as choice rules  
 324 and aggregate expressions. The rules with these features are translated by CMODELS [11]  
 325 into rules considered here. The CMODELS(DIFF) system translates a logic program into  
 326 SMT(IL) formulas, after which an SMT solver is called to find models of these formulas (that  
 327 correspond to answer sets).

<sup>2</sup> CMODELS(DIFF) is posted at <https://www.unomaha.edu/college-of-information-science-and-technology/natural-language-processing-and-knowledge-representation-lab/software/cmodels-diff.php>

## 23:10 SMT-based Answer Set Solver CMODELS(DIFF) (System Description)

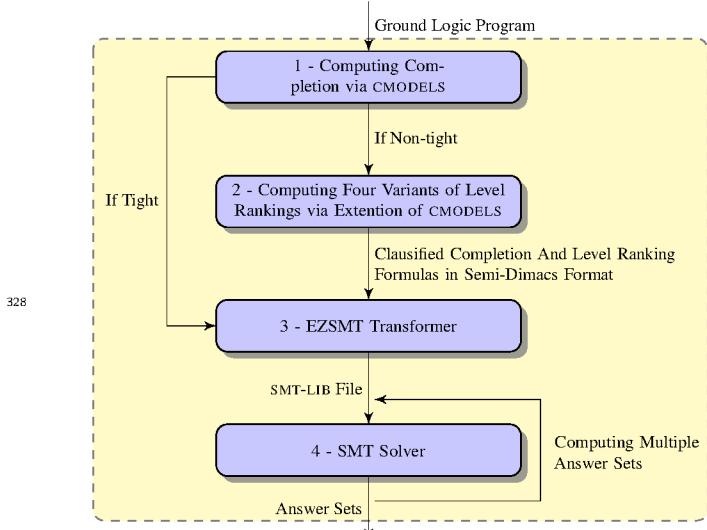


Figure 2 CMODELS(DIFF) Pipeline

328  
 329 Flags `-levelRanking`, `-levelRankingStrong`, `-SCClevelRanking`, and  
 330 `-SCClevelRankingStrong` instruct CMODELS(DIFF) to construct a level ranking formula,  
 331 a strong level ranking formula, a SCC level ranking formula, and a strong SCC level ranking  
 332 formula, respectively. And, `-SCClevelRanking` is chosen by default. Finally, completion and  
 333 the level ranking formula are classified using the same technique as in original CMODELS.  
 334 The CMODELS(DIFF) system outputs the resulting clauses into a text file in semi-Dimacs  
 335 format [21].

### 336 (3, 4) Transformation and Solving

337 The transformer is taken from EZSMT v1.1. It converts the semi-Dimacs output from step (2)  
 338 into SMT-LIB syntax (SMT-LIB is a standard input language for SMT solvers [1]). By default,  
 339 the SMT-LIB output contains an instruction that sets the logic of SMT solvers to Linear  
 340 Integer Arithmetic. If the transformer is invoked with the parameter `difference-logic`,  
 341 then the SMT-LIB output sets the logic of SMT solvers to Difference Logic instead.

342 Finally, one of the SMT solvers CVC4, z3, or YICES is called to compute models by  
 343 using flags `-cvc4`, `-z3`, or `-yices`. (In fact, any other SMT solver supporting SMT-LIB  
 344 can be utilized.) The CMODELS(DIFF) system post-processes the output of the SMT solvers  
 345 mentioned above to produce answer sets in a typical format disregarding any auxiliary atoms  
 346 or integer variables that are created during the system's execution.

347 The CMODELS(DIFF) system allows us to compute multiple answer sets. Currently, SMT  
 348 solvers typically find only a single model. We design a process to enumerate all models.  
 349 For a logic program  $\Pi$ , after an SMT solver finds a model and exits, the CMODELS(DIFF)  
 350 system constructs a clause that consists of (i) atoms in  $At(\Pi)$  that are assigned *false* by the  
 351 model and (ii) negations of atoms in  $At(\Pi)$  that are assigned *true* by the model. This clause  
 352 is added into the SMT-LIB formula previously computed. Then, the SMT solver is called  
 353 again taking the new input. The process is performed repeatedly, until the SMT-LIB formula  
 354 becomes unsatisfiable.

355 In summary, CMODELS(DIFF) has eight possible configurations. We can choose one from  
 356 the four variants of level ranking formulas, and choose a logic from either Linear Integer  
 357 Arithmetic or Difference Logic for the invoked SMT solver.

### (1,2) Computing Completion and Level Ranking Formulas

The CMODELS(DIFF) system utilizes the original algorithm of CMODELS to compute completion, during which CMODELS determines whether the program is tight or not. If the program is not tight, the corresponding level ranking formula is added.

358 **6 Experiments**

359 We benchmark CMODELS(DIFF) on seven problems, to compare its performance with that of  
 360 other ASP solvers, namely CMODELS and CLASP [7]. All considered benchmarks are *non-tight*  
 361 programs. The first two benchmarks are Labyrinth and Connected Still Life, which are  
 362 obtained from the Fifth Answer Set Programming Competition<sup>3</sup>. We note that the original  
 363 encoding of Still Life is an optimization problem, and we turn it into a decision one. The next  
 364 three benchmarks originate from Asparagus<sup>4</sup>. The selected problems are RandomNonTight,  
 365 Hamiltonian Cycle and Wire Routing. We also consider five instances of Wire Routing  
 366 from RST Construction<sup>5</sup>. Then, we use Bounded Models as the sixth benchmark<sup>6</sup>. Our  
 367 last benchmark, Mutual Exclusion, comes from Synthesis Benchmarks<sup>7</sup>. We rewrite the  
 368 seven encodings to fit the syntax of GRINGO 4, and call GRINGO v. 4.5.3<sup>8</sup> to produce ground  
 369 programs serving as input to all benchmarked systems. All benchmarks are posted at the  
 370 CMODELS(DIFF) website provided at Footnote 2.

371 All benchmarks are run on an Ubuntu 16.04.1 LTS (64-bit) system with an Intel core  
 372 i5-4250U processor. The resource allocated for each benchmark is limited to one cpu core  
 373 and 4GB RAM. We set a timeout of 1800 seconds. No problems are solved simultaneously.

374 Numbers of instances are shown in parentheses after names of benchmarks. We present  
 375 cumulative time of all instances for each benchmark with numbers of unsolved instances  
 376 due to timeout or insufficient memory inside parentheses. All the steps involved, including  
 377 grounding and transformation, are reported as parts of solving time.

378 Five distinct solvers are benchmarked: (1) CMODELS(DIFF) invoking SMT solver CVC4 v.  
 379 1.4; (2) CMODELS(DIFF) invoking SMT solver z3 v. 4.5.1; (3) CMODELS(DIFF) invoking SMT  
 380 solver YICES v. 2.5.4; (4) CLASP v. 3.1.3; (5) CMODELS v. 3.86.1 with Satisfiability solver  
 381 Minisat v. 2.0 beta. We use DIFF-CVC4, DIFF-Z3, and DIFF-YICES to denote three variants of  
 382 CMODELS(DIFF) used in the experiments.

■ **Table 1** Experimental Summary

Benchmark	DIFF-CVC4 LIA	DIFF-Z3 LIA	DIFF-YICES LIA	DIFF-Z3 DL	DIFF-YICES DL	CMODELS	CLASP
<b>Still Life (26)</b>	731	5423(1)	203	899	194	<u>647</u>	<b>10.8</b>
<b>Ham. Cycl. (50)</b>	15.39	9.78	4.54	6.61	3.57	<u>1.19</u>	<b>0.53</b>
<b>Wire Rout. (10)</b>	1378	562.36	1562	2983(1)	2089(1)	<u>409</u>	<b>12.5</b>
<b>Bound. Mod. (8)</b>	6.08	4.30	2.34	2.93	2.20	<u>1.59</u>	<b>1.38</b>
Labyrinth (30)	19543(8)	27794(12)	20425(10)	22023(9)	21836(9)	<u>16408(7)</u>	<b>5826(2)</b>
Rand. Nont. (20)	27.8	8.65	6.84	7.72	6.47	<u>1.39</u>	3.52
Mut. Excl. (5)	5.26	2.72	1.70	2.28	1.50	<u>0.30</u>	<b>0.13</b>

383 Table 1 summarizes main results. Under the name of variants of the CMODELS(DIFF)  
 384 systems, we state the configuration used for this solver. Namely, "LIA" and "DL" denote that  
 385 the logic of SMT solvers is set to Linear Integer Arithmetic and Difference Logic, respectively.  
 386 All DIFF systems in the table are invoked with flag `-SCClevelRanking`. Systems CLASP

<sup>3</sup> <https://www.mat.unical.it/aspcomp2014/>

<sup>4</sup> <https://asp.haiti.cs.uni-potsdam.de/>

<sup>5</sup> <http://people.sabanciuniv.edu/~esraerdem/ASP-benchmarks/rst-basic.html>

<sup>6</sup> <http://users.ics.aalto.fi/~kepa/experiments/boundsmodels/>

<sup>7</sup> <http://www2.informatik.uni-stuttgart.de/fmi/szs/research/projects/synthesis/benchmarks030923.html>

<sup>8</sup> <http://potassco.sourceforge.net/>

387 and CMODELS are run with default settings. We benchmarked CMODELS(DIFF) with all  
 388 eight possible configurations. Yet, we do not present all of the data here. CMODELS(DIFF)  
 389 invoked with `-levelRanking` and `-levelRankingStrong` flags shows worse performance  
 390 than settings `-SCClevelRanking` and `-SCClevelRankingStrong`, respectively. That is why  
 391 we avoid presenting the results on configurations `-levelRanking` and `-levelRankingStrong`.  
 392 Also, adding constraints for strong level ranking typically slightly degrades the performance so  
 393 we do not present the results for the `-SCClevelRankingStrong` configuration. We note that  
 394 SMT solver CVC4 implements the same procedure for processing Difference Logic statement  
 395 and Linear Integer Arithmetic statements.

### 396 Observations

397 We observe that system CLASP almost always displays the best results. This is not surprising  
 398 as this is one of the best *native* answer set solvers currently available. Its search method is  
 399 attuned towards processing logic programs. Given that SMT solvers are agnostic towards  
 400 specifics of logic programs it is remarkable how good the performance of CMODELS(DIFF) is.  
 401 In some cases it is comparable to that of CLASP.

402 It is the case that many Satisfiability solvers and answer set solvers share a lot in common [12]. For example, answer set solver CLASP starts by computing clausified programs  
 403 completion and then later applies to it *Unit* propagator search technique stemming from  
 404 Satisfiability solving. That is reminiscent of the process that system CMODELS(DIFF) undertakes. It also computes program's completion so that *Unit* propagator of SMT solvers is  
 405 applicable to it.

406 We conjecture that the greatest difference between CMODELS(DIFF) and CLASP lies in the  
 407 fact that

- 408   ■ in CMODELS(DIFF) integer linear constraints encode the conditions to weed out unwanted  
   409   models of completion; SMT solvers implement search techniques/propagators to target  
   410   these integer linear constraint;
- 411   ■ in CLASP the structure of the program is taken into account by the so called *Unfounded*  
   412   propagator for this task.

413 In case of Still Life, Hamiltonian Cycle, Wire Routing, and Bounded Models benchmarks  
 414 (marked in bold in Table 1) there is *one more substantial difference*. These encodings contain  
 415 aggregates. CLASP implements specialized search techniques to benefit from the compact  
 416 representations that aggregates provide. System CMODELS(DIFF) translates aggregates  
 417 away, which often results in a bigger problem encoding that the system has to deal with.  
 418 System CMODELS also translates aggregates away. This is why we underline the solving  
 419 times of CMODELS, as it is insightful to compare the performance of CMODELS to that  
 420 of CMODELS(DIFF) alone. Indeed, CMODELS(DIFF) utilizes the routines of CMODELS for  
 421 eliminating aggregates and computing the completion of the resulting program. Thus, the  
 422 only difference between these systems is in how they eliminate models of completion that are  
 423 not answer sets. System CMODELS(DIFF) utilizes level rankings for that. System CMODELS  
 424 implements a propagator in spirit of *Unfounded* propagator of CLASP, but the propagator of  
 425 CMODELS is only used when a model of completion is found; CLASP utilizes this propagator  
 426 as frequently as it utilizes *Unit* propagator [14, Section 5]. We believe that when we observe  
 427 a big difference in performance of CMODELS(DIFF) and CLASP, this attributes to the benefits  
 428 gained by the utilization of specialized *Unfounded* and "aggregate" propagators by CLASP.  
 429 Yet, level ranking formulas seem to provide a viable alternative to *Unfounded* propagator  
 430 and open a door for utilization of SMT solvers for dealing with non-tight programs. This  
 431 gives us grounds to believe that the future work on extending constraint answer set solver

434 EZSMT to accept non-tight programs is a viable direction.

435 As we noted earlier SCC level rankings yield best performance among the four variants  
 436 of level rankings. Furthermore, Table 1 illustrates the following. The logic of SMT solvers  
 437 does not make an essential difference. Overall, CMODELS(DIFF)-YICES with Linear Integer  
 438 Arithmetic logic performs best within the presented CMODELS(DIFF) configurations. Obvi-  
 439 ously, utilizing better SMT solvers can improve the performance of CMODELS(DIFF) in the  
 440 future. Notably, this does not require modifications to CMODELS(DIFF), since SMT-LIB used  
 441 by CMODELS(DIFF) is a standard input language of SMT solvers.

442 **7 Conclusion**

443 In this paper we presents the CMODELS(DIFF) system that takes a logic program and translates  
 444 it into an SMT-LIB formula which is then solved by an SMT solver to find answer sets of the  
 445 given program. Our work parallels the efforts of an earlier answer set solver LP2DIFF [10]. The  
 446 CMODELS(DIFF) system allows richer syntax such as choice rules and aggregate expressions,  
 447 and enables computation of multiple solutions. (In this work we extended the theory of  
 448 level rankings to the case of programs with choice rules and denials.) We note that the  
 449 LP2NORMAL<sup>9</sup> tool can be used as a preprocessor for LP2DIFF in order to enable this system  
 450 to process logic programs with richer syntax. In the future, we will compare performance of  
 451 CMODELS(DIFF) and LP2DIFF experimentally. Yet, we do not expect to see great difference  
 452 in their performance when the same SMT solver is used as a backend. Also, we would like to  
 453 conduct more extensive experimental analysis to support our conjecture on the benefits of  
 454 specialized "aggregate" propagator and *Unfounded* propagator employed by CLASP.

455 The technique implemented by CMODELS(DIFF) for enumerating multiple answer sets of  
 456 a program is basic. In the future we would like to adopt the nontrivial methods for model  
 457 enumeration discussed in [8] to our settings. The theory developed in this paper provides  
 458 a foundation to extend the recent constraint answer set programming solver EZSMT [21] to  
 459 accept non-tight constraint answer set programs. The contributions of this work also open a  
 460 door to the development of a novel constraint-based method in processing logic programs  
 461 by producing intermediate output in MINIZINC [18] in place of SMT-LIB. We believe our  
 462 work will boost the cross-fertilization between the three areas: SMT, constraint answer set  
 463 programming, and constraint programming.

464 **Acknowledgements**

465 We are grateful to Cesare Tinelli for valuable discussions on the subject of the paper and  
 466 for the insights on the cvc4 system. We are also thankful to Ben Susman. Da Shen was  
 467 supported by the 2017-FUSE (Fund for Undergraduate Scholarly Experiences) Grant from  
 468 the University of Nebraska at Omaha. Yuliya Lierler was partially supported by the NSF  
 469 1707371 grant.

470 ————— **References** —————

471 1 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5.  
 472 Technical report, Department of Computer Science, The University of Iowa, 2015.

473 2 Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund Clarke, Tom  
 474 Henzinger, and Helmut Veith, editors, *Handbook of Model Checking*. Springer, 2014.

<sup>9</sup> <https://research.ics.aalto.fi/software/asp/lp2normal/>

475 3 Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and*  
 476 *Data Bases*, pages 293–322. Plenum Press, New York, 1978.

477 4 Esra Erdem and Vladimir Lifschitz. Fages’ theorem for programs with nested expressions.  
 478 In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 242–254,  
 479 2001.

480 5 François Fages. Consistency of Clark’s completion and existence of stable models. *Journal*  
 481 *of Methods of Logic in Computer Science*, 1:51–60, 1994.

482 6 Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory*  
 483 *and Practice of Logic Programming*, 5:45–74, 2005.

484 7 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten  
 485 Schaub. Progress in CLASP series 3. In *Proceedings of the Thirteenth International Confer-*  
 486 *ence on Logic Programming and Nonmonotonic Reasoning (LPNMR’15)*, 2015.

487 8 Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-  
 488 driven answer set enumeration. In *Proceedings of the 9th International Conference on*  
 489 *Logic Programming and Nonmonotonic Reasoning*, LPNMR’07, pages 136–148, Berlin, Hei-  
 490 delberg, 2007. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=1758481.1758496>.

491 9 Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In  
 492 Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge*  
 493 *Representation*, pages 89–134. Elsevier, 2008.

494 10 Tomi Janhunen, Ilkka Niemelä, and Mark Sevalnev. Computing stable models via re-  
 495 ductions to difference logic. In *Logic Programming and Nonmonotonic Reasoning*, pages  
 496 142–154. Springer Berlin Heidelberg, 2009.

497 11 Yuliya Lierler. *SAT-based Answer Set Programming*. PhD thesis, University of Texas at  
 498 Austin, 2010.

499 12 Yuliya Lierler. What is answer set programming to propositional satisfiability. *Con-*  
 500 *straints*, pages 1–31, 2016. URL: <http://dx.doi.org/10.1007/s10601-016-9257-7>, doi:10.  
 501 1007/s10601-016-9257-7.

502 13 Yuliya Lierler and Benjamin Susman. On relation between constraint answer set pro-  
 503 gramming and satisfiability modulo theories. *Theory and Practice of Logic Programming*,  
 504 17(4):559–590, 2017.

505 14 Yuliya Lierler and Mirosław Truszczyński. Transition systems for model generators — a  
 506 unifying approach. *Theory and Practice of Logic Programming*, 27th Int’l. Conference on  
 507 *Logic Programming (ICLP) Special Issue*, 11(4-5):629–646, 2011.

508 15 Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic  
 509 programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

510 16 Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat  
 511 solvers. *Artificial Intelligence*, 157:115–137, 2004.

512 17 Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT  
 513 Press, 1998.

514 18 N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, , and G. Tack. Minizinc:  
 515 Towards a standard cp modelling language. In *Proceedings of the 13th International Con-*  
 516 *ference on Principles and Practice of Constraint Programming*, page 529–543, 2007.

517 19 Ilkka Niemela. Stable models and difference logic. *Annals of Mathematics and Artificial*  
 518 *Intelligence*, 53:313–329, 2008.

519 20 Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and  
 520 weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–  
 521 521. Kluwer, 2000.

522 21 Benjamin Susman and Yuliya Lierler. SMT-Based Constraint Answer Set Solver EZSMT  
 523 (System Description). In *Technical Communications of the 32nd International Conference*  
 524 *on Logic Programming (ICLP 2016)*, volume 52, pages 1:1–1:15, 2016.