Introduction to Difference Logic

Satisfiability Checking Seminar

Moritz Daniel Clever
Supervision: Gereon Kremer

SS 2017

Abstract

Difference logic is a fragment of linear arithmetic which is successfully used in a wide range of system verification and program analysis for many years. Common application areas are the invariant generation in Petri nets, scheduling problems, as well as termination and non-termination proving. In this paper, I present a short introduction in difference logic with the corresponding part of graph theory. Then introduce a strategy to decide the satisfiability problem and finally provide a rough sketch of the main idea of Satisfiability Modulo Theory (SMT) solver in association with difference logic.

1 Introduction

As a fragment of linear arithmetic, difference logic is successfully used in a wide range of system verification and program analysis for many years [7]. Common application areas are the invariant generation in Petri nets [14], scheduling problems [15], as well as termination [10] and non-termination proving [9].

In difference logic literals are inequalities of the form $u - v \leq k$. This subset of linear arithmetic has lower complexity but is still powerful to express many encountered real-life problems.

Deciding difference logic is an NP-complete problem [13]. The current state-of-the-art procedure to solve linear arithmetic are solvers based on the Satisfiability Modulo Theory (SMT). SMT starts with a transformation of the difference logic formula in an equi-satisfiable boolean formula and tries to solve it with an off-the-shelf boolean SAT solver. A occurring unsatisfiable of the abstraction is already the prove of the unsatisfiable of the original difference logic formula. Otherwise, the corresponding inequalities of the difference formula needs to be investigated. This step needs detailed theory informations and is the main topic of this paper.

The rest of the paper is organized as follows: in Chapter 2 I introduce terms, theorems and definitions used in this paper. In Chapter 3 I point out the relation between difference logic and graph theory. I introduce the general proceed of a SMT based solver and demonstrate a simple decision procedure, furthermore I present some current and future work concerning difference logic.

2 Preliminaries

2.1 Difference Logic

In this section I define formally the class of formulas we consider. The class is called *difference logic* (DL) [5] and is a subset of *linear arithmetic* and a *quantifier-free first order logic* based on difference inequalities.

Definition 2.1 (Difference Logic) *Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be a set of propositional atoms and $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ a set of integer variables. A difference inequality, also called literal is an inequality $\varphi \in DL$ of the following form:*

$$u - v \leq k$$

where $u, v \in \mathcal{X}$ and $k \in \mathbb{Z}$ for Integer Difference Logic (IDL).

Consider that equalities of the form $u - v = k$ can be replaced by $u - v \leq k \wedge u - v \geq k$. And inequality's such as $u - v \neq k$ can be replaced by $u - v < k \vee u - v > k$. Furthermore, a literal like $u - v \geq k$ is rewritten as $v - u \leq -k$. And finally, as our domain is $\mathbb{Z}$ in IDL, the set of integers, inequalities of the form $u - v < k$ can be rewritten as $u - v \leq k - 1$. For a different domain than $\mathbb{Z}$, we can replace the literal by $u - v \leq k - \delta$, where δ is a sufficient small real. A specific application for this case is the Rational Differecen Logic (IDL) where $k \in \mathbb{R}$.
Therefore, in this paper we assume all inequalities are in the form $u - v \leq k$.

Definition 2.2 (Difference system) *Sets (conjunctions) of disjunctions of inequalities are called difference system.*

Modern SAT solvers only apply on conjunctive normal form (CNF). Intuitively this normal form means that a difference system, a set of conjunctions is only satisfiable if all clauses hold. CNF is extremely well suited for identify unsatisfiability faster. It is sufficient to find a single unsatisfiable clause to determine unsatisfiable. Furthermore, the CNF is necessary to apply resolution [11].

Definition 2.3 (CNF) *A formula $\Phi \subset DL$ is in conjunctive normal form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals $\varphi_{i,j}$ (inequalities), therefore formulas of the form:*

$$\bigwedge_i \bigvee_j \varphi_{i,j}$$

where $\varphi_{i,j}$ is the j-th literal in the i-th clause.

Example. On one hand, the difference system $\Phi = ((x_1 + x_2 \leq c_1 \vee x_2 + x_3 \leq c_1) \wedge (y_1 + y_2 \leq c_2))$ is in conjunctive normal form. On the other hand, the formula $\Phi = ((x_1 + x_2 \leq c_1 \wedge x_2 + x_3 \leq c_1) \vee (y_1 + y_2 \leq c_2))$ is based on the definition not in CNF because the two terms are connected with a disjunction.

In propositional boolean logic there exist for each formula φ an equivalent formula φ_C in CNF. By using Tseitin transformation [2] an equi-satisfiable formula can be generated in linear time by using auxiliary variables, also in difference logic. Equi-satisfiable indicates that the formula φ_C is satisfiable whenever φ is satisfiable. This restriction is nevertheless sufficient for our purpose to prove the satisfiability of φ.

An important step is to evaluate the satisfiability of a DL formula. For this it is necessary to define the valuation of a formula. Like in every other logic that's done by an assignment.

Definition 2.4 (Assignment) *A (difference logic) assignment is a function $\alpha : x \to \mathbb{Z}$ used to map the variables $x \in \mathcal{X}$ of an inequality to an integer value. This mapping is necessary to evaluate the satisfiability of an inequality. In addition to evaluate the satisfiability of a formula, we can extend the function to evaluate inequalities to propositional atoms $p \in \mathcal{P}$ as $\alpha : p \to \{True, False\}$. An assignment defines the valuation of an inequality by:*

$$\alpha(x_i - x_j \leq c) = \textit{ True iff } \alpha(x_{-i}) - \alpha(x_j) \leq c$$

Boolean connectives $(\wedge, \vee, \neg)$ gets applied by the obvious rules. A formula Φ is satisfiable if it has a satisfying assignment α. Hereafter we also call this α satisfies Φ or $\alpha \vDash \Phi$. α is also referred to as a model of Φ.

Example. Let $\varphi \in DL$ be defined as $(a - b \leq 4)$ and let α be an assignment with $\alpha(a) = 3, \alpha(b) = 2$, then the valuation of the inequality φ is defined as

$$\alpha(\varphi) = \alpha(\alpha(a) - \alpha(b) \leq 4) = \alpha(3 - 2 \leq 4) = \alpha(1 \leq 4) = True$$

Thus α is in this example a satisfying assignment for φ. $(\alpha \vDash \varphi)$

3 Difference Logic and Graph Theory

Sets of inequalities can also be represented as a graph. In the following is graph theory used to perform consistency checks.

The graph representation of inequalities in difference logic is called *constraint graph.*

Definition 3.1 (Constraint Graph) *Let $G = (V, E)$ be a directed weighted graph over a given set of inequalities $\mathcal{S} \subset DL$. With the set of integer variables $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ of $\mathcal{S}$, we consider the vertices $\{x_1, x_2, \ldots, x_m\} \in V$, edges $e = (x_i, x_j) \in E$ and a weight function $w : e \to k$ for each inequality $(x_i - x_j \leq k) \in \mathcal{S}$. Hereafter I will quote inequalities also as $x_i \xrightarrow{k} x_j$. The graph $G(\mathcal{S})$ is called the constraint graph of $\mathcal{S}$.*

Example. Given set $\mathcal{S}$ of inequalities $\mathcal{S} := \{a - b \leq 2, b - c \leq 3, c - a \leq -7, c - d \leq 5, b - d \leq -1\}$ construct constraint graph $G(\mathcal{S})$:

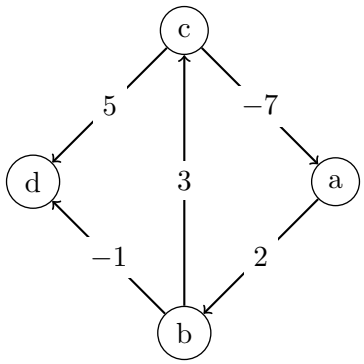

Figure 1: Constraint graph.

In the next section we want to inspect an important property of the constraint graph, the negative cycles.

3.1 Negative Cycles

Definition 3.2 (Negative Cycles) *A negative cycle in the constraint graph is a path of edges and vertices wherein a vertex is reachable from itself and the total weight of the path is negative.*

$$x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} \ldots \rightarrow x_n \xrightarrow{k_n} x_1, \text{ where } \sum_{i=1}^{n} k_i < 0$$

Example. The following graph is composed of a single cycle $a \xrightarrow{2} b \xrightarrow{3} c \xrightarrow{-7} a$. By Definition 3.2 it's weight is calculated by $3+2+(-7) = (-2)$. According to the condition, that the total weight of the path is less than zero, it is a negative cycle.

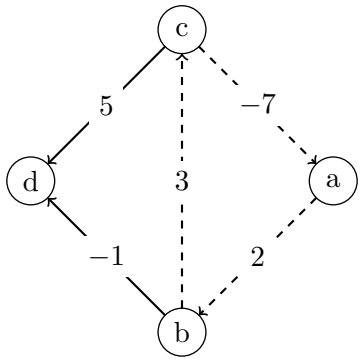

Figure 2: Simple negative cycle.

Normally the detection of negative cycles is a side effect of a shortest path algorithm. Hence, the detection can be performed by using the Bellman-Ford algorithm in time $O(|V| \times |E|)$ [1].

Algorithm 1 Bellman-Ford-Algorithm for negative cycle detection

```
BELLMAND-FORD ( Vertices, Edges )
 1  for  vertex v ∈ Vertices
 2  do distance[v] ← ∞
 3     predecessor[v] ← ∞
 4     for  i = 0 to (|Vertices| - 1 )
 5  do for   edge (u, v) ∈ Edges
 6     do if distance[v] > distance[u] + weight(u, v)
 7           then distance[v] ← distance[u] + weight(u, v)
 8                predecessor[v] ← u
 9                for  edge (u, v) ∈ Edges
10  do if distance[v] > distance[u] + weight(u, v)
11        then  Negative cycle detected
12              Cycle reconstructed following predecessor
```

The Bellman-Ford algorithm (Algorithm 1) is one of the simplest algorithm for negative cycle detection and therefore it is well suited to illustrate consistency checks.

At first (lines 1-3) the algorithm initializes the distance of all nodes to infinity. The next step (lines 4-8) calculates the shortest distances. For $|Vertices| - 1$ times, for each

edge, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value. The longest possible path without a cycle can be $|Vertices|-1$ edges. If in the final scan (lines 9-12) a shorter path of length $|V|$ has found, a negative cycle was detected.

Example. Given inequality set $\mathcal{S} := \{a - b \leq 2, b - c \leq 3, c - a \leq -7, c - d \leq 5, b - d \leq -1\}$ construct constraint graph $G(\mathcal{S})$ (Figure 3). After applying Bellman-Ford, the negative cycle is detected (Figure 4). The negative cycle contains the vertices's {a,b,c} and the corresponding edges are dashed.

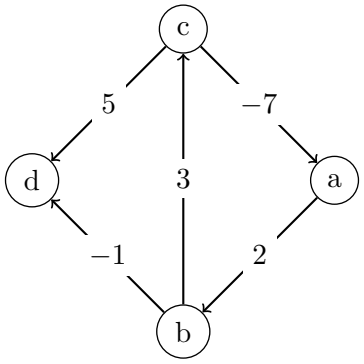

Figure 3: Constraint graph.

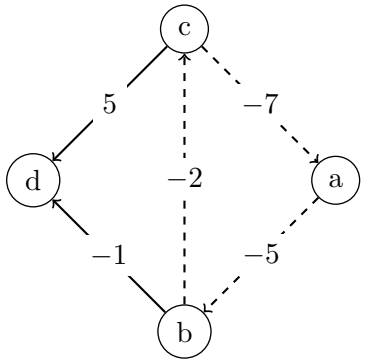

Figure 4: Graph (Figure 3) after applying the Bellman-Ford algorithm.

Nowadays there exist more efficient variants for negative cycle detection [6] [16]. One approach uses reduced costs, since they are not negative, and computes the shortest path with Dijkstra's algorithm, a Greedy algorithm which is much more efficient. The step to produce non-negative weights is essential for this approach because Dijkstra only works for graphs with non-negative weights.

Basically any algorithm that is able to detect negative cycles in weighted graphs can be used. The detection of a negative cycle decides already the satisfiability problem of a difference system.

Theorem 3.1 *Let Φ be a difference system, and $G(\Phi)$ its constraint graph. Then Φ has no solution iff $G(\Phi)$ has a negative cycle.*

Proof 3.1 *Any negative cycle $x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} \ldots \rightarrow x_n \xrightarrow{k_n} x_1$ corresponds to a set of literals $\{x_1 - x_2 \leq k_1, x_2 - x_3 \leq k_2, \ldots, x_n - x_1 \leq k_n\}$. If we add all of them, we get $0 \leq k_1 + k_2 + \cdots + k_n$, which is inconsistent since negative cycle implies $k_1 + k_2 + \cdots + k_n < 0$.*

The detection of negative cycles can be used as a tool to process consistency checks.

3.2 Infeasible subset

In Chapter 3.1 I introduced an approach to decide the satisfiability of a difference system. But if we found a negative cycle and thus we know that the given difference system is not satisfiable the occurring question concerns the explanation for inconsistency. The explanation will be given by a set of inequalities of the original difference system called infeasible subset.

Definition 3.3 (Infeasible subset) *A infeasible subset is a set of inequalities of the difference system, which contains all inequalities which make the whole difference system infeasible.*

A infeasible subset is still not always minimal. Example, we can remove inequalities from it but still having an infeasible subset. Therefore, the set of all original input inequalities is a legit infeasible subset. But the goal is to achieve a much smaller subset, which is in general the case.

As already mentioned in the Bellman-Ford algorithm, the infeasible subset can be reconstructed by following the path of predecessors of a negative cycle. The literals $\{x_1 - x_2 \leq k_1, x_2 - x_3 \leq k_2, \ldots, x_n - x_1 \leq k_n\} \subseteq \Phi$ corresponding to the negative cycle $x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} \ldots \to x_n \xrightarrow{k_n} x_1$ are the infeasible subset of the given difference system Φ.

Example. Given set of inequalities $\mathcal{S} := \{a - b \leq 2, b - c \leq 3, c - a \leq -7, c - d \leq 5, b - d \leq -1\}$ construct after applying Bellman-Ford algorithm the constraint graph $G(\mathcal{S})$. (see Example 3.1, Figure 1)

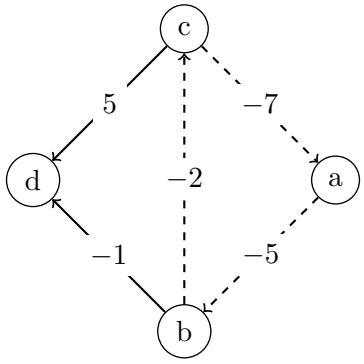

Figure 5: Constraint graph example.

The negative cycle in this example is $c \xrightarrow{-7} a \xrightarrow{-5} b \xrightarrow{-2} c$, which lead to an infeasible subset of $\{a - b \leq 2, b - c \leq 3, c - a \leq -7\} \subset \mathcal{S}$. This three inequalities are the explanation for in-feasibility of the inequality set $\mathcal{S}$.

To identify this infeasible subsets is vital for using this strategy as a theory satisfiability checking procedure, such as the SMT algorithm.

3.3 Model

If a constraint graph $G(\mathcal{S}) = (V, E)$ of a set of inequalities $\mathcal{S}$ does not contain a negative cycle, then $\mathcal{S}$ is satisfiable. If we extend the constraint graph by an additional vertex x_0, which has an edge to each vertex $\{x_1, x_2, \ldots, x_m\} \in V$. Then we are able to find an assignment α for each variable $x_1, x_2, \ldots, x_m$ by calculating the shortest path δ from x_0 to the variable x_n. This calculation can be done by any shortest path algorithm that can handle with negative weights, the Bellman-Ford algorithm for example.

Definition 3.4 (Extended Constraint Graph) *Let $G_E(\mathcal{S}) = (V, E)$ be a constraint graph with vertices $V = \{x_1, \ldots, x_n\} \cup \{x_0\}$ and edges $E = \{(x_j, x_i) | x_i - x_j \in \mathcal{S}\} \cup \{(x_0, x_i) | 1 \leq i \leq n\}$.*

Example. Given set of inequalities $S := \{a-b \leq 2, b-c \leq 3, c-a \leq 4, c-d \leq 5, b-d \leq -1\}$ construct the constraint graphs $G(\mathcal{S})$ and $G_E(\mathcal{S})$.

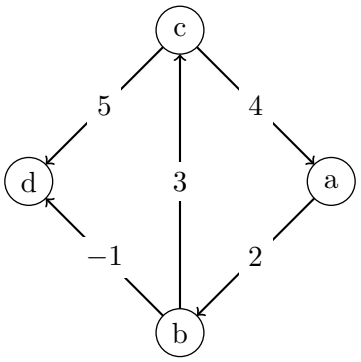

Figure 6: Constraint graph $G(\mathcal{S})$

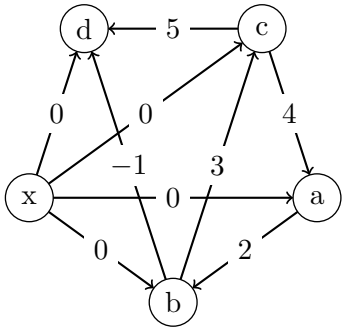

Figure 7: Extended graph $G_E(\mathcal{S})$

Shortest path algorithms can now calculate the shortest path δ from x to each of the other vertices, which leads to a satisfying assignment α.

$$\alpha(a) = \delta(x,a) = 0, \alpha(b) = \delta(x,b) = 0, \alpha(c) = \delta(x,c) = 0, \alpha(d) = \delta(x,d) = -1$$

3.4 Satisfiability Modulo Theories

The state-of-the-art procedure to solve linear arithmetic problems are solvers based on Satisfiability Modulo Theories (SMT) [12]. There exists different kinds of SMT approaches, but in this Chapter I want to focus only on the less lazy SMT-solving strategy.

SMT is such a big subject that I am only able to roughly sketch the main idea of this strategy. But, this should be sufficient to recognize the relation between this paper, general SAT solving and current research approaches as mentioned in Chapter 3.5.

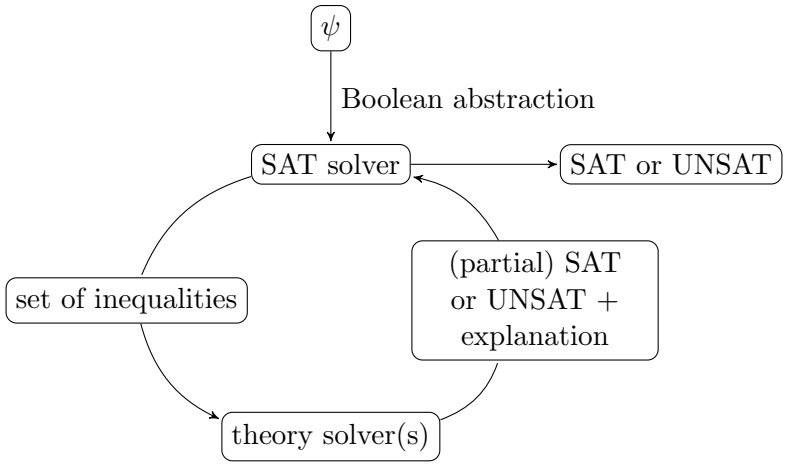

Figure 8: Less lazy SMT-solving.[1]

[1]Image source: lecture 'Satisfiability Checking' by Prof. Dr. Erika Ábrahám

SMT starts by doing a boolean abstraction of the original CNF formula, which translates the difference system Φ into a propositional formula Φ_P.

Example. $\Phi = \underbrace{a - b \leq 2}_{a_1} \wedge \underbrace{b - c \leq 3}_{a_2} \wedge (\underbrace{c - a \leq -7}_{a_3} \vee \underbrace{c - d \leq 5}_{a_4})$

$$\Phi_P = a_1 \wedge a_2 \wedge (a_3 \vee a_4)$$

For efficiency reasons, the translation into the abstract formula should be so smart, that the relation between literals and their negation is made explicit. For example, if $u - v \leq k$ and $u > k + v$ occurs, they should be abstracted by a propositional variable and its negation.

The SAT-solver evaluates Φ_P until a satisfiable assignment α is found. If no satisfying assignment exists, the difference system is unsatisfiable.

$$\alpha(a_1) = 1, \alpha(a_2) = 1, \alpha(a_3) = 1, \alpha(a_4) = 0$$

Otherwise, the related set $\mathcal{S}$, consisting of all inequalities which corresponding propositional boolean variables of the abstraction are TRUE, is used as an input for the theory solver, which proves the satisfiability of this (partial) set of the original difference logic formula.

$$\mathcal{S} := \{a - b \leq 2, b - c \leq 3, c - a \leq -7\}$$

If the set is not satisfiable, the theory solver returns the explanation as a minimal infeasible subset $\mathcal{S}_I$, otherwise it returns SAT (satisfiable). If all propositional variables of Φ_P had already been assigned, the satisfiability of the original difference system Φ has been shown and the SAT-solver returns SAT.

$$\mathcal{S}_I := \{a - b \leq 2, b - c \leq 3, c - a \leq -7\}$$

Otherwise, like in this example, the appearing infeasible subset $\mathcal{S}_I$ gets remapped into the corresponding propositional variables of Φ and added as a negated conjunction to Φ_P. This should avoid the same unsatisfiable assignment.

$$\mathcal{S} = a_1 \wedge a_2 \wedge (a_3 \vee a_4) \wedge (\boldsymbol{\neg a_1 \vee \neg a_2 \vee \neg a_3})$$

After adding these restrictions of the theory solver to the SAT-solver, it needs to re-evaluate the abstraction. And the procedure begins anew, until the satisfiability is determined.

To complete the example, lets do a second run. The SAT-solver evaluates the extended Φ_P again until a satisfiable assignment α is again found.

$$\alpha(a_1) = 1, \alpha(a_2) = 1, \alpha(a_3) = 0, \alpha(a_4) = 1$$

The new corresponding set $\mathcal{S}$ of inequalities is one more time used as an input for the theory solver.

$$\mathcal{S} := \{a - b \leq 2, b - c \leq 3, c - d \leq 5\}$$

This time the new set $\mathcal{S}$ is satisfiable.

$$\alpha(a) = 0, \alpha(b) = 0, \alpha(c) = 0, \alpha(d) = 0$$

The satisfying assignment α will be returned to the SAT-solver which now returns SAT (satisfiable), cause the boolean abstraction Φ_P was also satisfied and all variables were assigned.

This less lazy style of SMT-solving require three properties of the theory solver.

1. **Incrementality**: The less lazy approach extends the difference system and for efficiency the solver should make use of the previous satisfiability check for the check of the extended set.

2. **Infeasible subsets**: It is necessary that the theory solver compute a reason for unsatisfiability. The usual way is to determine an unsatisfiable subset of the constraints which is ideally minimal in the sense that if we remove a constraint the remaining ones become satisfiable.

3. **Backtracking**: The theory solver should be able to backtrack, i.e. remove constraints in inverse chronological order.

The previous introduced procedure to detect unsatisfiability on the basis of negative cycle detection (Chapter 3.1) and the procedure to find a model for a satisfiable set (Chapter 3.3) supports all of these properties. Therefore, the described procedures represent a possible and functional SMT theory solver.

The performance of a SMT based solver depends on choice and implementation of the SAT-solver as well as on the theory-solver.

3.5 Current Research

As mentioned in the introduction of this paper, the current trend is to gain efficiency of a certain logic. Therefore, the focus of current researches on difference logic is on speeding up the decision of the satisfiability problem.

The introduced solution of the negative cycle detection is a comprehensive one, but not the first choice if speed is significant. The recurrently graph construction for the negative cycle detection can take a considerable amount of time.

One approach tries to pass the bottleneck by propose a new encoding of difference logic graph theory for generating constraints based on templates [4]. These encoding leads to non-linear satisfiability modulo theories (SMT) problems. Instead of a heavyweight non-linear SMT solver the authors propose to employ a much cheaper SMT solver for difference logic. The consistency checks can be done less frequent and faster by the new encoding.

Another approach I did not mention yet, is the famous Simplex algorithm [8]. It was designed for solving optimization problems, specified by a set of linear arithmetic constraints. Normally solutions of the Simplex algorithm are maximal (or minimal) values for the constraint variables, which is a much harder problem to solve. But experimental results are promise that current Simplex-based extensions are competitive for solving linear integer arithmetic with state-of-the-art SMT solvers [3].

4 Conclusion

Difference logic, as a fragment of linear arithmetic is up-to-date and the topic of different kinds of current research papers. The wide range of relevant application areas are the reasons, why the topic is still today so important. The researches furthermore tend to speeding up the decision of the satisfiability problem based on various approaches, such as DPLL, SMT and Simplex-based extensions optimized for DL.

References

[1] R. Bellman. On a routing problem. *Quart. Appl. Math. 16 (1958), 87-90*, 1958.

[2] A. Biere, M. Hele, H. van Maaren, and T. Walsh. *Handbook of Satisfiability.* IOS Press, Amsterdamm, 2009.

[3] F. Bobot, S. Conchon, E. Contejean, M. Iguernelala, A. Mahboubi, A. Mebsout, and G. Melquiond. *A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic*, pages 67–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[4] L. Candeago, D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. *Speeding up the Constraint-Based Method in Difference Logic*, pages 284–301. Springer International Publishing, Cham, 2016.

[5] S. Cotton, E. Asarin, O. Maler, and P. Niebert. *Some Progress in Satisfiability Checking for Difference Logic*, pages 263–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[6] S. Cotton and O. Maler. *Fast and Flexible Difference Constraint Propagation for DPLL(T)*, pages 170–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, May 1991.

[8] B. Dutertre and L. de Moura. *A Fast Linear-Arithmetic Solver for DPLL(T)*, pages 81–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] D. Larraz, K. Nimkar, A. Oliveras, E. Rodrıguez-Carbonell, and A. Rubio. *Proving Non-termination Using Max-SMT*, pages 779–796. Springer-Verlag New York, Inc., New York, NY, USA, 2014.

[10] D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. *Proving termination of imperative programs using Max-SMT*, pages 218–225. IEEE, Oct 2013.

[11] G. Mints. Resolution calculus for the first order linear logic. *Journal of Logic, Language and Information*, 2(1):59–83, 1993.

[12] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. AC*, 53(6):937–977, nov 2006.

[13] A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel. The small model property: How small can it be? *Information and Computation*, 178(1):279 – 293, 2002.

[14] S. Sankaranarayanan, H. Sipma, and Z. Manna. *Petri Net Analysis Using Invariant Generation*, pages 682–701. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[15] J. P. Stinson, E. W. Davis, and B. M. Khumawala. Multiple resource–constrained scheduling using branch and bound. *A I I E Tran.*

[16] C. Wang, F. Ivančić, M. Ganai, and A. Gupta. *Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination*, pages 322–336. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.