

Ario: A Linear Integer Arithmetic Logic Solver

Hossein M. Sheini

Electrical Engineering and Computer Science Dept.
University of Michigan, Ann Arbor, MI 48109, USA
Email: hsheini@umich.edu

Karem A. Sakallah

Electrical Engineering and Computer Science Dept.
University of Michigan, Ann Arbor, MI 48109, USA
Email: karem@umich.edu

Abstract—Ario is a solver for systems of linear integer arithmetic logic. Such systems are commonly used in design verification applications and are classified under Satisfiability Modulo Theories (SMT) problems. Recognizing the fact that in many such applications the majority of atoms are equalities or integer unit-two-variable inequalities (UTVPIs), we present a framework that integrates specialized theory solvers for those atoms within a SAT solver. The unique feature of our strategy is its simultaneous adoption of both a congruence-closure equality solver and a transitive-closure UTVPI solver to find a satisfiable set of those atoms. A full-scale ILP solver is then utilized to check the consistency of all integer constraints within the solution. Other notable features of our solver include its combined deduction and learning schemes that collectively make our solver distinct among similar solvers.

I. INTRODUCTION

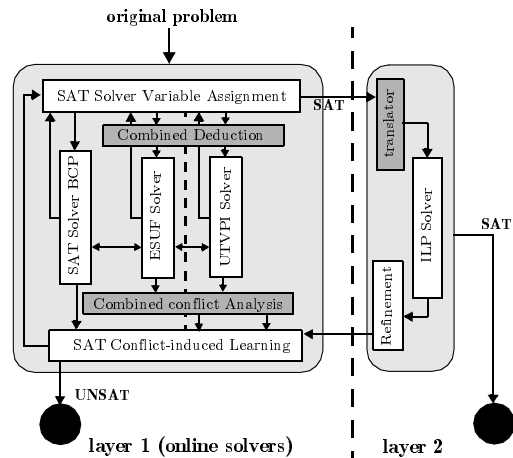
In the past few years there has been a noticeable surge in the introduction and application of various logics to model systems of integer constraints. Different design verification problems are routinely cast in terms of logical problems whose atoms are constraints over integer variables.

Ario *version 1.2* is a solver for checking the satisfiability of quantifier-free formulas in linear integer arithmetic logic¹. Ario essentially is able to deal with two main logics: i) equality logic with successors and uninterpreted functions (ESUF), whose atoms are in the form of $t_i = t_j + c$ where $c \in \mathbb{Z}$ and t_i, t_j are terms, that are recursively defined to be either integer variables or applications of uninterpreted functions over terms, and ii) linear integer arithmetic logic (LIA) whose atoms are linear constraints over integer variables. We assume that ESUF and LIA atoms within the input problem do not share variables; i.e. an equality over two integer variables, $t_i = t_j + c$, is an ESUF atom only if t_i and t_j are not present in any inequalities; otherwise it is replaced with a conjunction of two LIA atoms: $t_i - t_j \leq c$ and $t_j - t_i \leq -c$. We additionally categorize LIA atoms into two types, each to be treated differently: i) Unit-Two-Variable-Per-Inequality (UTVPI) atoms in the form of $a_i x_i + a_j x_j \leq b$ where x_i, x_j are integer variables, $a_i, a_j \in \{0, \pm 1\}$ and $b \in \mathbb{Z}$, and ii) non-UTVPI atoms in the form of $\sum_{i=1}^n a_i x_i \leq b$ where $a_i, b \in \mathbb{Z}$ and x_i 's are integer variables.

Ario adopts a generic CNF SAT solver to reason about and analyze the logical structure of the problem. Following the DPLL framework, the SAT solver incrementally builds a satisfiable set of atoms and utilizes specialized *theory solvers*

¹For complete details on the overall algorithm of Ario and its different techniques the reader is referred to [1] and [2].

Fig. 1. The organization of solving algorithms within Ario



to maintain the consistencies of the conjunctions of various *theory atoms* within that solution. Upon activating ESUF or UTVPI atoms, the consistencies of those atoms are incrementally checked using, respectively, a congruence-closure or a transitive-closure procedure. The overall consistency of all LIA atoms is established after/if a satisfiable solution to the rest of the problem is built.

Related Work: The idea of integrating different theory solvers within a propositional SAT solver was first suggested in [3] and then further improved in [4], [5], [6], [2]. Two of the most common integration strategies employed in these solvers include, i) the *layered approach* [5], based on invoking theory solvers in the order of their solving capabilities, and ii) the online [4] or DPLL(T) [6] approach, where a combined DPLL reasoning and learning procedure is applied to all atoms and the consistency of the theory atoms are maintained throughout the SAT search. The latter is essentially a form of “early pruning” as introduced in [3].

II. THE SOLVER ARCHITECTURE

The overall architecture of Ario is demonstrated in Figure 1. In this section we describe each of Ario’s theory solvers, and the framework for utilizing these solvers within SAT.

A. Theory Solvers

The following three solving procedures are utilized in Ario, each capable to decide the consistency of a conjunction of specific theory atoms. Considering that the ESUF and UTVPI

solvers are integrated within the SAT solver, their associated algorithms are both incremental and backtrackable.

1) *Solving Equalities with Successors and Uninterpreted Functions (ESUF)*: This solver initially replaces all applications of the successor functions by simple addition of their arguments with integer constants and subsequently adopts a reimplementation of the congruence-closure algorithm of [7] to solve the conjunction of ESUF atoms.

2) *Solving UTVPI Constraints*: Upon activating a UTVPI atom by the SAT solver, that atom is added to a transitively-closed and tightened set of constraints as in [8]. A conflict is detected if a constraint of the form $0 \leq -1$ is implied.

3) *Solving Integer Linear Constraints*: This solver adopts a generic Simplex/Branch-and-Bound (BNB) methods to establish the satisfiability of the integer constraints. Activated non-UTVPI atoms together with only those UTVPI atoms that share variables with them are solved with this solver [2].

B. Integration within SAT

In Ario we implemented a *hybrid approach* [1] to integrate theory solvers within SAT. In this framework, similar to early pruning approach of [3], those theory solvers that can efficiently adapt to the DPLL SAT procedure are integrated online and the rest are utilized in separate layers and are only applied to complete SAT solutions. More specifically, the ESUF and UTVPI solvers are integrated within SAT while non-UTVPI constraints are solved in an offline layer (as demonstrated in Figure 1 and further described in [2]). Our hybrid method enables the solver to efficiently process ESUF and UTVPI atoms and only check the consistency of hard non-UTVPI integer constraints when it is absolutely necessary, i.e. when a satisfiable assignment to Boolean, ESUF and UTVPI atoms is found. Further enhancements due to adopting our combined deduction and learning schemes are as follows.

1) *Combined Deduction Scheme*: Ario utilizes an inter-logic deduction scheme [1] outside the theory solvers that builds an implication graph taking into account all types of atoms. Implications in this graph are both due to unit-clause-propagation of the SAT solver and the linear combination of integer constraints. In this scheme all possible non-negative linear combinations of equality and UTVPI constraints are generated and implied. Non-UTVPI constraints are only combined with equality or UTVPI constraints if a variable could be eliminated and they are not combined with other non-UTVPI constraints. These combinations could generate new atoms that are not present in the original formula to be added to their respective theory solvers on the fly. This method enables the solver to reason about theory atoms outside their specific theory solvers and helps each solver within our framework to prune infeasible solutions due to other types of theory atoms. For further details, the reader is referred to [1].

2) *Combined Learning Scheme*: By analyzing the combined deduction scheme at each conflict, this method learns clauses in terms of different types of theory atoms. For instance, if a conflict is detected among non-UTVPI atoms, the combinations of non-UTVPI and UTVPI atoms are considered

to learn a clause in terms of only UTVPI atoms. This clause can then be used in the online search to reduce the number of calls to the costly ILP solver. This is further explained in [2].

III. THE SOLVER IMPLEMENTATION

Ario is implemented in C++ and is compiled using gcc version 3.2.2. Ario is not capable to solve instances with big integers, i.e. integers that need more than 32 bits to represent. Even though Ario is specifically designed to solve problems in the theory of integers, it is also capable to solve problems in the theory of reals with some limitations. The logics that Ario supports are presented in Table I.

TABLE I
LOGICS SUPPORTED BY ARIO AND THE CORRESPONDING SOLVERS

Logic	Activated Solvers		
	equality	UTVPI	Simplex/BNB
QF_UF	X		
QF_UFIDL	X	X	
QF_UFLIA	X	X	X
QF_IDL	X	X	
QF_LIA	X	X	X
QF_RDL	X	X	
QF_LRA	X	X	X

It is important to note that since Ario utilizes a Simplex/BNB solver to handle linear constraints in QF_LRA and QF_LIA logics, its completeness is affected by the precision of such solvers (majority of these solvers find solution within a tolerance). More specifically in QF_LRA instances with *strict inequalities* Ario is *not complete* and may wrongly prove the (un)satisfiability of such instances.

IV. SUMMARY

Ario adopts a hybrid integration approach, i.e. it applies the online integration strategy to check the consistency of ESUF and UTVPI atoms within SAT and the layered approach for non-UTVPI integer constraints. This approach for categorizing integer constraints together with our framework specifically adapt to applications in design verification.

REFERENCES

- [1] H. M. Sheini and K. A. Sakallah, "A progressive simplifier for satisfiability modulo theories." in *SAT*, 2006, pp. 184–197.
- [2] —, "A scalable method for solving satisfiability of integer linear arithmetic logic." in *SAT*, 2005, pp. 241–256.
- [3] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani, "A SAT based approach for solving formulas over boolean and linear mathematical propositions," in *CADE-18*, 2002, pp. 195–210.
- [4] S. Berezin, V. Ganesh, and D. L. Dill, "An online proof-producing decision procedure for mixed-integer linear arithmetic," in *TACAS*, 2003, pp. 521–536.
- [5] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani, "An incremental and layered procedure for the satisfiability of linear arithmetic logic." in *TACAS*, 2005, pp. 317–333.
- [6] R. Nieuwenhuis and A. Oliveras, "DPLL(T) with exhaustive theory propagation and its application to difference logic." in *CAV*, 2005, pp. 321–334.
- [7] —, "Proof-Producing Congruence Closure," in *RTA*, 2005, pp. 453–468.
- [8] J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap, "Beyond finite domains," in *Workshop on Principles and Practice of Constraint Programming*, 1994, pp. 86–94.