Fx7 or In Software, It Is All About Quantifiers

Michał Moskal*
University of Wrocław, Poland

1 Introduction

Fx7 is an SMT solver for first order formulas, built over linear arithmetic and uninterpreted function symbols. Fx7 is implemented in the Nemerle language and runs on the .NET platform, specifically the solver was developed and tested using Mono, an open source .NET implementation. Fx7 is available under a BSD-like license from `http://nemerle.org/fx7/`.

It was designed for software verification queries, like the ones generated by ESC/Java [6] and Boogie [1] tools. These queries do not put much pressure on arithmetic, instead they make heavy use of quantifiers in axiomatization of the checked programming language semantics. Fx7 is therefore mainly a research vehicle for testing different strategies of dealing with quantifiers. We want to compete **only in the AUFLIA** division and expect to do well in terms of number of benchmarks solved and not-so-well in terms of total time taken. Our answer to the random seed question is (obviously) **42**.

"SAT" Answer: During the competition the solver will never say the formula is satisfiable. This is due to inherent incompleteness of quantifier instantiation and high penalty for giving wrong answer. On the other hand, in the "real world" applications we have found it very important to answer promptly that the formula is satisfiable with some degree of confidence. We feel that the next SMT competition should somehow accommodate for this fact in the scoring rules.

2 Bits and Pieces

MiniSat: The solver uses a C# port of the MiniSat [5] to drive DPLL-like search. In some modes of operations it also calls the normal C++ MiniSat (the reasons to do that are only historical). The search algorithm can be classified as being in between DPLL(T) and lazy proof explication. We let MiniSat find a satisfying assignment, assert the trail to the theories, putting "push" marks in the places where MiniSat left them, generate theory conflict clause and let MiniSat continue. Then, when MiniSat finds next satisfying assignment, we rollback as much as we need (depending on how much did the MiniSat backtrack), and assert the rest.

The solver extends MiniSat with a specialized variable selection function, that makes sure variables are not assigned truth value, unless some of the clauses require that. MiniSat would in general assign truth value to all the variables, which generates spurious instances of quantified formulas. Other remedies for this problem include lazy CNF conversion, used in Simplify [3] or marking parts of the formula relevant, as seen in Z3 [2].

*Partially supported by Polish Ministry of Science and Education grant 3 T11C 042 30.

Linear Arithmetic: Fx7 handles linear rational arithmetic, using a modified Simplex algorithm [4]. Several heuristics are applied to make the solver handle integers, but it is still integer-incomplete. The main problem with the algorithm is equality generation, which takes a considerable amount of time.

Partial Orders: The solver can scan the input formula automatically for the transitivity and antisymmetry axioms, and if they are found, the decision procedure is used instead of them. This mode is however not used during competition, as the general quantifier instantiation allows for solving more problem (albeit in longer time).

Arrays: At some point in time, there was a dedicated decision procedure for arrays. It was however dropped, as it was found to be slower than the general quantifier instantiation.

Uninterpreted Functions and Combination: The theory of uninterpreted functions is handled by a variant of congruence closure algorithm [10]. The theories cooperate through the Nelson-Oppen combination.

Quantifier Instantiation: Quantifiers are handled through instantiation, much like in Simplfy [3]. Instantiation is driven by triggers, which are subterms of the quantified formula. The formula is instantiated when one of its triggers appear (modulo the current congruence) somewhere in the goal. The problem of finding such occurrences is solved using two novel algorithms [9]. The current version of SMT-LIB standard does not contain a syntax for specifying triggers, we therefore use heuristics to guess them. However, as the axiomatizations in both ESC/Java and Boogie are designed with triggers in mind, it seems wasteful not to use that information, so we hope this will be fixed in the next release of the standard.

Instance Management: The solver uses two level SAT solver [7] to deal with instances. Whenever theories are unable to detect a conflict, a separate SAT solver is spawned and instantiations are added to it. When a conflict involving instantiations is found, the solver looks for literals from the main SAT solver, that imply the conflict. The architecture accommodates any number of nested SAT solvers, but using more than two levels was not found to be beneficial.

Input Language: Fx7 does not have its own input language. It reads formulas in SMT-LIB concrete syntax as well as in Simplify syntax. The solver does not distinguish between sorts.

Fx8 Branch: Fx8 is a fork of Fx7 that does not use MiniSat or N-level SAT and instead implements search algorithm much like the one used in Simplify [3]. It improves upon it, by adding conflict analysis and conflict learning. It is significantly slower than Fx7, but on the other hand it has the ability to generate proofs (complete with theory conflicts and skolemization) that are checked by an external tool [8]. We were able to generate formal proofs of over 95% of problems in the AUFLIA division.

Applications: The solver has been integrated in the ESC/Java2 tool, through a general `sortedProver` interface. The interface is a realization of SMT-LIB abstract syntax, where formulas are created on the fly and never serialized to text. The same interface is going to be used by the FreeBoogie tool, currently under development at the University College, Dublin.

People: The main developer is Michał Moskal (Institute of Computer Science, University of Wrocław, Poland). Jakub Łopuszański (also from Wrocław) developed one of the matching algorithms. We would also like to thank Joe Kiniry from the University College, Dublin for making test machines available.

References

[1] Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *Proceeding of CASSIS 2004*, volume 3362 of *Lecture Notes in Computer Science*. Springer–Verlag, 2004.

[2] Leonardo de Moura and Nikolaj Bjorner. Efficient E-matching for SMT solvers. In *Proceedings of the 21st International Conference on Automated Deduction (CADE-21)*. Springer, 2007, to appear.

[3] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[4] Bruno Dutertre and Leonardo de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proceedings of the 18th Computer-Aided Verification conference*, volume 4144 of *LNCS*, pages 81–94. Springer-Verlag, 2006.

[5] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT 2003*, 2003.

[6] Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended static checking for Java. In *ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI'2002)*, pages 234–245, 2002.

[7] K. Rustan M. Leino, Madan Musuvathi, and Xinming Ou. A two-tier technique for supporting quantifiers in a lazily proof-explicating theorem prover. In *TACAS*, pages 334–348, 2005.

[8] Michał Moskal. Rocket-fast proof checking, to appear somewhere in 2007.

[9] Michał Moskal, Jakub Łopuszański, and Joseph R. Kiniry. E-matching for fun and profit. In *Proceedings of the SMT workshop*, 2007.

[10] R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. In J. Giesl, editor, *16th International Conference on Term Rewriting and Applications, RTA'05*, volume 3467 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 2005.