TRANSALG: A Tool for Translating Procedural Descriptions of Discrete Functions to SAT

Ilya Otpuschennikov, Alexander Semenov, and Stepan Kochemazov

Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia

Abstract. In this paper we present the Transalg system, designed to produce SAT encodings for discrete functions, written as programs in a specific language. Translation of such programs to SAT is based on propositional encoding methods for formal computing models and on the concept of symbolic execution. We used the Transalg system to make SAT encodings for a number of cryptographic functions.

Introduction

Many new methods for solving Boolean Satisfiability Problem (SAT) were introduced in the past two decades. These methods make it possible to solve combinatorial problems from various areas [2]. One can use different approaches to encode an original problem to SAT [14]. Often each particular problem requires researchers to develop and implement special encoding technique. Recently a number of systems that automate procedures of encoding combinatorial problems to SAT were developed [6], [7], [12], [17], [20].

In our paper, we present the Transalg system that translates procedural descriptions of discrete functions to SAT. Translation mechanisms employed in Transalg are based on the ideas of S.A. Cook on propositional encoding of Turing Machine programs [3] and ideas of J.S. King on symbolic execution [8]. At this time we mainly use Transalg to produce SAT encodings for cryptographic functions. Inversion problems for such functions are usually computationally hard. Recent works show that the study of cryptanalysis problems in the context of SAT approach can yield promising results [5], [10], [13], [16], [18], [19]. In our opinion, Transalg might become a powerful tool in the research of various cryptographic functions.

To be processed by Transalg the function description should be written in the special TA-language. The TA-language is a procedural domain specific language (DSL) with C-like syntax. Therefore, it is usually sufficient to introduce minor changes to C-implementation of an algorithm to produce its description in the TA-language.

It should be noted that the base concept of Transalg makes it possible to produce SAT encodings for arbitrary discrete functions computable in polynomial time. We believe that Transalg will be useful for many researchers who want to use SAT approach in their studies but do not have time or desire to develop special encoding techniques for each individual problem.

The TA-Language for Description of Discrete Functions

As we already mentioned above, the Transalg system uses special DSL, named TA-language, to describe discrete functions. Below by translation we mean the process of obtaining a propositional encoding for a TA-program.

The translation of a TA-program has two main stages. At the first stage, Transalg parses a source code of a TA-program and constructs a syntax tree using standard techniques of compilation theory [1]. At the second stage, the system employs the concept of symbolic execution [8] to construct a propositional encoding for a TA-program considered. Transalg can output an encoding obtained in any of standard forms (CNF, DNF, ANF).

The TA-language has block structure. A block (composite operator) is a list of instructions. Every block is delimited by curly braces '{' and '}'. Within a block instructions are separated by ';' symbol. Each block has its own (local) scope. In the TA-language nested blocks are allowed with

no limit on depth. During the analysis of a program Transalg constructs a scope tree with global scope at its root. Every identifier in a TA-program belongs to some scope. Variables and arrays declared outside of any block and also all functions belong to a global scope and therefore can be accessed in any point of a program.

A TA-program is essentially a list of functions. The main function is an entry point so it must exist in any program. The TA-language supports base constructions used in procedural languages (variable declarations, assignment operators, conditional operators, loops, function calls, function returns etc.), various integer operations and bit operations including bit shifting and comparison.

Main data type in the TA-language is the bit type. Transalg uses this type to establish links between variables used in a TA-program and Boolean variables included into corresponding propositional encoding. It is important to distinguish between these two sets of variables. Below we will refer to variables that appear in a TA-program as *program variables*. All variables included in a propositional encoding are called *encoding variables*. Upon the translation of an arbitrary instruction that contains a program variable of the bit type, Transalg links this program variable with a corresponding encoding variable. Transalg establishes such links only for program variables of the bit type. Variables of other types, in particular of the int type and the void type are used only as service variables, for example as loop counters or to specify functions without return value.

Declarations of global bit variables can have __in or __out attribute. Attribute __in marks variables that contain input data for an algorithm. Attribute __out marks variables that contain an output of an algorithm. Local bit variables cannot be declared with these attributes.

Translation of TA-programs to SAT

Let us consider a sequence of computations defined by an arbitrary TA-program as a sequence of data modifications in a memory of an abstract computing machine in moments $\{0,1,...,e\}$. At every moment $i \in \{0,1,...,e\}$ Transalg associates a set X^i of encoding variables with program variables of the bit type. Denote $X = \bigcup_{i=0}^{e} X^i$. Suppose that X^{in} is formed by encoding variables that correspond to input data, and X^{out} contains encoding variables corresponding to the output of a discrete function considered. It is easy to see that $X^{in} \subseteq X^0$ and $X^{out} \subseteq X$.

Transalg uses the translation concept that makes it possible to reduce the redundancy of propositional encoding. We will explain this concept on the following example.

Example 1. Consider an encoding of a linear feedback shift register (LFSR) [11] with Transalg. On fig. 1 we show the TA-program for the LFSR with feedback polynomial $P(z) = z^{19} + z^{18} + z^{17} + z^{14} + 1$ over GF(2) (here z is a formal variable).
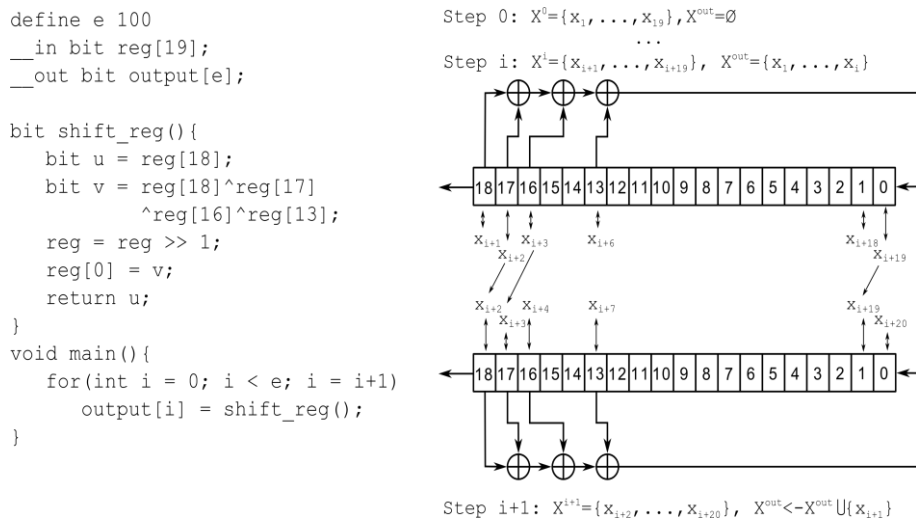

Fig. 1. TA-program for the LFSR.

Note that during the translation of transition from step i to step $i+1$ it is not necessary to create new encoding variables for every cell of the register. If we copy data from one register cell to another, then we can use the same encoding variable to encode corresponding data value at steps i and $i+1$. Therefore, at each step Transalg creates only one new encoding variable and links it with program variable `reg[0]`. All the other program variables get linked with encoding variables created at previous steps. In accordance with the above, a set of encoding variables corresponding to initial values of the register is $X^{in} = X^0 = \{x_1, x_2, ..., x_{19}\}$. After each shift we encode values of register cells with sets

$$X^1 = \{x_2, x_3, ..., x_{20}\}, \ X^2 = \{x_3, x_4, ..., x_{21}\}, \ \cdots, \ X^e = \{x_{e+1}, x_{e+2}, ..., x_{e+19}\}.$$

It is clear that $X^{out} = \{x_1, ..., x_e\}$. Thus the set of encoding variables for this program is $X = \{x_1, x_2, ..., x_{e+19}\}$, and the propositional encoding is the following set of Boolean formulae

$$x_{20} \equiv x_1 \oplus x_2 \oplus x_3 \oplus x_6$$
$$...$$
$$x_{e+19} \equiv x_e \oplus x_{e+1} \oplus x_{e+2} \oplus x_{e+5}.$$

An important feature of the Transalg system is full support of conditional operators. Consider an arbitrary expression $\Phi(z_1, ..., z_k)$ of the TA language. Here $z_1, ..., z_k$ are program variables of the `bit` type. Suppose that they are linked with encoding variables $x_1, ..., x_k$. Also suppose that Boolean formula $\phi(x_1, ..., x_k)$ is obtained as a result of translation of an expression $\Phi(z_1, ..., z_k)$. Below we say that an expression $\Phi(z_1, ..., z_k)$ is associated with a Boolean formula $\phi(x_1, ..., x_k)$.

The BNF of conditional operator has the following form

```
<if_statement> := if (<expression>) <statement> [else <statement>]
```

In this BNF `<expression>` is a predicate of a conditional operator. Let `<expression>` be Φ. Suppose that expression Φ is associated with formula ϕ. Denote by Δ_1 and Δ_2 some expressions of a TA-program, associated with Boolean formulae δ_1 and δ_2, respectively. Now suppose that program variable z is the left operand in the assignment operator $z = \Delta_1$, performed in the first branch of the conditional operator and also the left operand of the assignment operator $z = \Delta_2$, performed in the second (else-) branch of the conditional operator. Also, suppose that at the previous translation step z was linked with encoding variable x. During the translation of such conditional operator Transalg creates new encoding variable x', links it with program variable z and adds the following formula to the propositional encoding:

$$x' \equiv \phi \cdot \delta_1 \vee \neg \phi \cdot \delta_2. \tag{1}$$

If there is no assignment $z = \Delta_2$ in the else-branch, or if there is no else-branch, then formula (1) transforms into $x' \equiv \phi \cdot \delta_1 \vee \neg \phi \cdot x$. Likewise if there is no assignment $z = \Delta_1$ in the first branch then formula (1) transforms into $x' \equiv \phi \cdot x \vee \neg \phi \cdot \delta_2$.

Note that according to the BNF-definition any operator can be a branch of conditional operator. In particular, we can consider the construction of n nested conditional operators. Without the loss of generality, suppose that we have the following operator

```
if (Φ₁) z = Δ₁
```

```
else if (Φ₂) z = Δ₂
```

...

```
else if (Φₙ) z = Δₙ
```

```
else z = Δₙ₊₁
```

Suppose that each expression Φ_j, $j = 1,...,n$, is associated with Boolean formula ϕ_j. Then during the translation of this operator Transalg will create new encoding variable x' and add the following formula to the propositional encoding

$$x' \equiv \phi_1 \cdot \delta_1 \vee \neg\phi_1 \cdot \phi_2 \cdot \delta_2 \vee ... \vee \neg\phi_1 \cdot \neg\phi_2 \cdot ... \cdot \phi_n \cdot \delta_n \vee \neg\phi_1 \cdot \neg\phi_2 \cdot ... \cdot \neg\phi_n \cdot \delta_{n+1}. \tag{2}$$

Here Boolean formulae δ_j, $j = 1,...,n+1$, are associated with expressions Δ_j. Note that (2) follows from (1).

Encoding Cryptanalysis Problems with Transalg

In recent years, a number of papers about the application of SAT-solvers to solving cryptanalysis problems [5], [10], [13], [16], [18], [19] were published. We used Transalg to make propositional encodings for a number of cryptographic functions.


```
define len 128
__in bit regA[19];
__in bit regB[22];
__in bit regC[23];
__out bit stream[len];
int midA = 8;
int midB = 10;
int midC = 10;

bit majority(bit x, bit y, bit z){
  return x & y | x & z | y & z;
}

void main(){
  for(int i = 0; i < len; i = i + 1){
    bit maj = majority(regA[midA],regB[midB],regC[midC]);
    if(!(maj^regA[midA])) shift_regA();
    if(!(maj^regB[midB])) shift_regB();
    if(!(maj^regC[midC])) shift_regC();
    stream[i] = regA[18]^regB[21]^regC[22];
  }
}
```
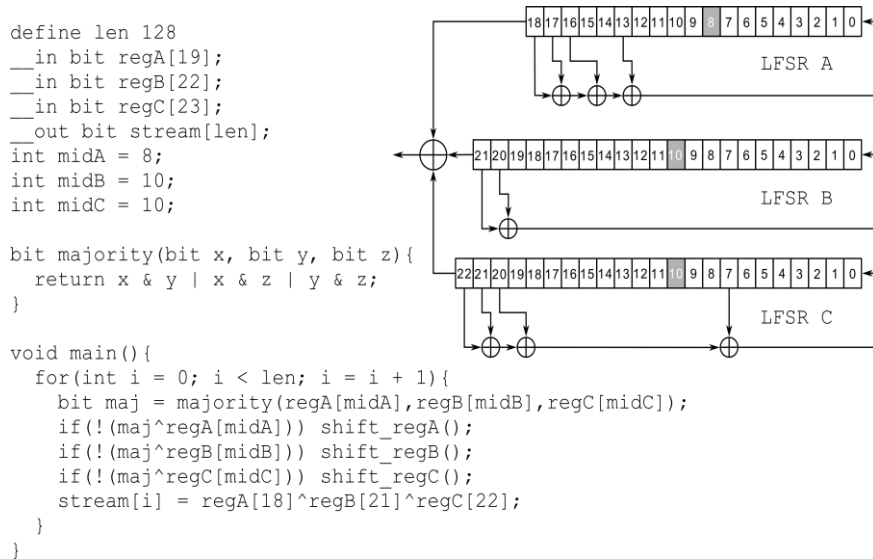
Fig. 2. Fragment of the TA-program for the A5/1 keystream generator.

In Fig. 2 we show the fragment of the TA-program for the A5/1 generator outputting 128 bits of keystream. Here we suppose that functions `shift_regA`, `shift_regB` and `shift_regC` are implemented in a similar way as `shift_reg` function in Fig. 1. To translate Boolean formulae to CNF it is sufficient to use only Tseitin transformations [21]. For the TA-program in Fig. 2 this approach gives us the CNF with 41600 clauses over the set of 10816 variables.

However, in some cases the use of specialized Boolean optimization libraries makes it possible to significantly decrease the size of propositional encoding. In particular, the Transalg system uses the Espresso[1] Boolean minimization library. In Espresso the Boolean formulas are minimized via the manipulation with their truth tables, that is why the complexity of the minimization procedure grows exponentially on the number of variables in the formula. In practice, it means that in order to spend reasonable time on minimization, it is best to minimize only formulas that contain limited number of variables. We use the following approach: if during the translation of the TA-program

---

[1] http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm

there arise the formulas over the set with more than 12 variables then such formulas are divided into subformulas using the Tseitin transformations.

With the help of Espresso the size of the propositional encoding of the A5/1 keystream generator outputting 128 keystream bits was reduced to 39936 clauses over the set of 8768 variables. Note, that to obtain the encoding for the A5/1 generator with different size of keystream one only needs to change the value of `len` constant in the TA-program in fig. 2 and repeat the translation. A5/1 encodings produced with Transalg were used to solve the problem of logical cryptanalysis of A5/1 in distributed computing environments [15, 16].

We also applied Transalg to encode the problem of cryptanalysis of the DES cipher to SAT. DES algorithm uses a lot of bit shifting and permutation operations. According to the translation concept used by Transalg, the system does not create new encoding variables during the translation of these operations. Propositional encodings of DES obtained with Transalg turned out to be significantly more compact than encodings from [9]. For example, if we consider the DES algorithm that takes 1 block of plaintext (64 bits), the corresponding CNF obtained with Transalg has 26400 clauses over the set of 1912 variables. In [9] CNF for the same problem has 61935 clauses over the set of 10336 variables.

The source code of the Transalg[2] system, the examples of TA-programs and the corresponding encodings[3] are freely accessible online. They include SAT encodings for problems of cryptanalysis of A5/1, A5/2, Bivium, Trivium, Geffe, Gifford, E0 and Grain keystream generators, DES algorithm, and also for hash algorithms MD4, MD5, SHA-1 and SHA-2.

Related Work

In recent years the SAT community have developed many encoding techniques that can be applied to a wide class of combinatorial problems. A lot of references to key papers in this area can be found in [14].

There is a number of systems for automated encoding of Constraint Satisfaction Problem (CSP) to SAT [6], [12], [17], [20]. Some methods and approaches to encoding of pseudoboolean constraints were described in [4].

In [18] there was presented the Grain of Salt tool that can be used to make SAT encodings for cryptanalysis of keystream generators. The Grain of Salt tool uses the special declarative language to describe keystream generators. This language was designed specifically to produce compact descriptions of configurations of shifting registers. The Transalg system, on the contrary, was designed as a general tool, and it can be applied to encoding much wider class of functions than that covered by keystream generators. The ideology of Transalg is based on modern results in programming language theory and theory of symbolic execution. The most close analogue of the Transalg system is, apparently, the URSA system [7]. In comparison to this tool, the distinctive feature of the Transalg system consists in ability to encode conditional operators.

The Transalg system showed high effectiveness in obtaining propositional encodings of various cryptographic functions. In particular, the SAT encodings of corresponding functions produced by Transalg are much more compact than known analogues.

Acknowledgements

References

[2] http://gitorious.org/transalg/
[3] http://gitorious.org/satencodings/

[1] Aho A.V., Lam M.S., Sethi R., Ullman J.D. *Compilers: Principles, Techniques and Tools* (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[2] Biere A., Heule M., van Maaren H., Walsh T. (eds.). *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 2009.

[3] Cook, S.A. The complexity of theorem-proving procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151—158, STOC'71, ACM, New York, NY, USA, 1971.

[4] Een N., Sorensson N.: Translating pseudo-boolean constraints into sat. *JSAT* 2(1-4), pp. 1-26, 2006.

[5] Eibach T., Pilz E., Volkel G. Attacking Bivium Using SAT Solvers. Buning H.K., Zhao X. (eds.) SAT. *Lecture Notes in Computer Science*, vol. 4996, pp. 63-76, Springer, 2008.

[6] Hebrard E., O'Mahony E., O'Sullivan B. Constraint programming and combinatorial optimisation in numberjack. *Lecture Notes in Computer Science*, vol. 6140, pp. 181-185, Springer, 2010.

[7] Janicic P. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, Vol. 8(3:30), pp. 1–39, 2012.

[8] King J.C. Symbolic execution and program testing. Commun. ACM 19(7), pp. 385-394, 1976.

[9] Massacci F., Marraro L. Logical cryptanalysis as a sat problem. J. Autom. Reasoning 24(1/2), pp. 165-203, 2000.

[10] Mcdonald C., Charnes C., Pieprzyk J. Attacking Bivium with MiniSat. Tech. Rep. 2007/040, ECRYPT Stream Cipher Project, 2007.

[11] Menezes A.J., Vanstone S.A., Oorschot P.C.V. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edn., 1996.

[12] Metodi A., Codish M. Compiling finite domain constraints to sat with bee. TPLP 12(4-5), pp. 465-483, 2012.

[13] Mironov I., Zhang L. Applications of sat solvers to cryptanalysis of hash functions. *Lecture Notes in Computer Science*, vol. 4121, pp. 102-115, Springer, 2006.

[14] Prestwich S.D. CNF encodings. *Handbook of Satisfiability*, IOS Press, 2009, pp. 75—97.

[15] Semenov A., Zaikin O. On estimating total time to solve SAT in distributed computing environments: Application to the SAT@home project. CoRR abs/1308.0761, 2013.

[16] Semenov A., Zaikin O., Bespalov D., Posypkin M. Parallel logical cryptanalysis of the generator A5/1 in BNB-grid system. In: Malyshkin, V. (ed.) PaCT. *Lecture Notes in Computer Science*, vol. 6873, pp. 473-483, Springer, 2011.

[17] Soh T., Tamura N., Banbara M. Scarab: A rapid prototyping tool for SAT-based constraint programming systems. *Lecture Notes in Computer Science*, vol. 7962, pp. 429-436, Springer, 2013.

[18] Soos M. Grain of Salt - an Automated Way to Test Stream Cipher through SAT Solvers. Tools'10: Proceedings of the Workshop on Tools for Cryptanalysis. pp. 131-144, 2010.

[19] Soos M., Nohl K., Castelluccia C. Extending sat solvers to cryptographic problems. *Lecture Notes in Computer Science*, vol. 5584, pp. 244-257, Springer, 2009.

[20] Tamura N., Tanjo T., Banbara M. System description of a SAT-based CSP solver sugar. Proceedings of the Third International CSP Solver Competition pp. 71-75, 2008.

[21] Tseitin G.S. On the complexity of derivation in propositional calculus. Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp. 466—483, Springer, Berlin, Heidelberg, 1983.