Automated Deduction with XOR Constraints

Graham Steel
School of Informatics, University of Edinburgh, Scotland
graham.steel@ed.ac.uk
June 8, 2005

1 Introduction

This paper describes first-order theorem proving with XOR constraints. The purpose of these constraints is to facilitate efficient reasoning with models of APIs of hardware security modules (HSMs), which frequently employ XOR as a cryptographic operation. The self-inverse and associative-commutative properties of XOR cause a large combinatorial blow-up in the number of possible actions an attacker can perform when trying to break into an HSM. Furthermore, it is precisely these properties that have been exploited in previously discovered API attacks, [1]. In order to create models which we can reason with efficiently, it is vital that we account for these properties without sustaining the associated blow-up.

2 XOR Constraints

XOR constraints specify that two terms in a clause must be equal modulo XOR. Semantically, the constraints are a restriction on the instantiation of the universally quantified variables in the clauses. Rather than generating instantiations to solve the constraints directly, we allow the search to proceed normally, but check after each deduction that the constraints remain soluble.

2.1 Deduction with XOR Constraints

We employ the constrained resolution/paramodulation calculus of [3], with the addition of XOR constraints. The XOR constraints of two resolving clauses are combined in the resolvent using logical AND, just like the pre-existing ordering constraints and substitution constraints. When generating new inferences, we apply the substitution required for the inference to the constraints before checking for solubility. Solubility of an XOR constraint \( s_1 \oplus \ldots \oplus s_m =_{XOR} t_1 \oplus \ldots \oplus t_n \) is checked like this:

1. If any of \( s_1, \ldots, s_m, t_1, \ldots, t_n \) contain variables, the constraint is regarded as soluble.

2. If all \( s_1, \ldots, s_m, t_1, \ldots, t_n \) are ground, then first discard zeros, and then count up the number of occurrences of each term in the set \( \{s_1, \ldots, s_m, t_1, \ldots, t_n\} \). If all terms occur an even number of times, the constraint is soluble. If not, it is insoluble.

Only inferences producing clauses with soluble constraints are permitted. These simple syntactic checks can be made very quickly. Note that due to condition 1, we may well keep some inferences with constraints which cannot be solved in the theory in question. However, we will certainly only discard genuinely insoluble constraints. We can eliminate some further insoluble sets of XOR constraints by a simple pairwise check for inconsistency. If two constraints, both attached to the same clause, equate the
same variables to different ground terms, then they are inconsistent and the clause can be pruned away. Again, this is a highly incomplete check, but it is quick to perform and seems useful in practice.

2.2 Subsumption with XOR Constraints

Simplification and checking for redundancy are of paramount importance in practical theorem proving. Modifications to the resolution/paramodulation calculus which prevent the use of subsumption checking rules are usually useless in practice, whatever their apparent theoretical advantages. The use of XOR constraints allows subsumption checking, though with the following restriction: the solutions of the constraints in a clause that is subsumed must be a subset of the solutions of the clause we retain. For XOR constraints, this occurs just when:

1. The more general clause has no XOR constraint, or
2. The XOR constraints of the subsumed clause and the subsuming clause are identical (modulo AC), after any substitution required to make the subsumption has been applied to the XOR constraint.

This is similar to the standard rule for subsumption in the constrained calculus, [3, §5.1]. To see why condition 2 is not only sufficient but necessary to ensure that all solutions of some constraint, $T_1$, are solutions of another, $T_2$, the reader is invited to write down two identical constraints, and then add a single variable or ground term to either. Immediately it is possible to construct solutions of the first which are not solutions of the second, and vice versa.

In practice, many checks for subsumption are ruled out because of incompatible XOR constraints. Fortunately, the check can be made quite quickly.

2.3 Simplification with XOR Constraints

Reduction of newly produced clauses by demodulation, clausal simplification, etc. is also permissible for XOR-constrained clauses. In our implementation, we only allow demodulation by clauses with no XOR constraint, since in our experiments, it seems to be only these clauses we would like to use.

3 Results

We have implemented XOR constraints in the theorem prover daTac, [3], which has allowed us to rediscover two of Bond’s attacks on the IBM 4758 HSM, using a model of the entire key-management command set\(^1\). More details of our results are available in [2].

References


---

\(^1\)These attacks have been fixed since version 2.41 of the 4758 CCA API.