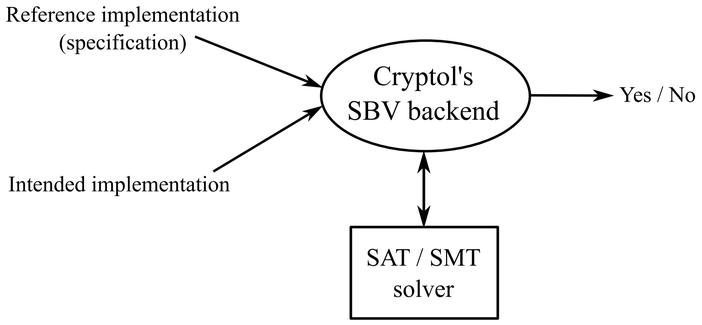
Using Term Rewriting to Solve Bit-Vector Arithmetic Problems

Iago Abal[1*] Alcino Cunha[1] Joe Hurd[2] Jorge Sousa Pinto[1]

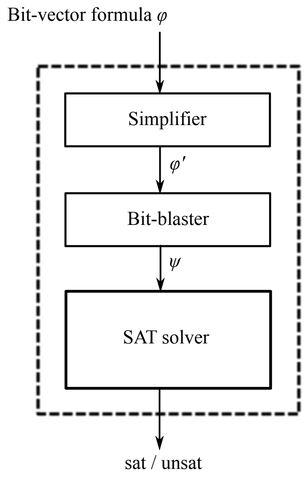
[1]HASLab / INESC TEC & Universidade do Minho, Braga, Portugal
[2]Galois, Inc., Portland, OR, USA

Equivalence checking in Cryptol

Cryptol is a domain-specific language (DSL) and toolset for cryptography developed by Galois, Inc.; providing an SMT backend that relies on bit-vector decision procedures to certify the correctness of cryptographic specifications.

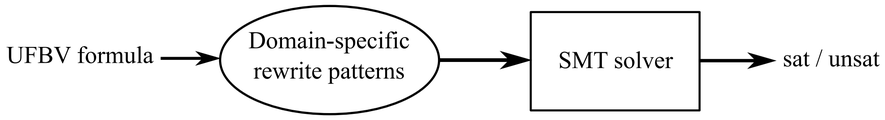


This approach works fine for many cases, including symmetric ciphers, but it cannot handle a wide range of interesting problems such as equivalence of public-key ciphers. Unfortunately bit-blasting does not scale very well, especially for bit-vector multiplication which is a key ingredient of public-key cryptography (PKC). This causes many equivalence problems to be transformed into propositional SAT problems that are intractable by current SAT solvers. The SMT backend also presents similar limitations since virtually all state-of-the-art SMT solvers rely on bit-blasting to decide bit-vector arithmetic.



Modern SMT solvers include a simplification phase that performs some rewriting on the input problem prior to bit-blasting, aiming to reduce the overall solving time. Nevertheless, SMT solvers have to deal with a wide range of application domains, and hence the set of rewrite rules employed for simplification is quite restricted.

A domain-specific rewriting-based approach



We address this problem by combining custom rewrite patterns, somehow encapsulating domain-specific proof strategies, with standard bit-vector decision procedures.

Very basic rules

These are cheap and generally applicable rewrite rules. Most, but not all, are already considered by SMT solvers. It is worth noting that very basic simplifications regarding modular arithmetic are not considered by state-of-the-art SMT solvers.
E.g.

$$
\begin{array}{lrcl}
(1) & A_{[n]} + 0_{[n]} & = & A_{[n]} \\
(2) & A_{[n]}[n-1:0] & = & A_{[n]} \\
(3) & (A_{[n]} \bmod M_{[n]}) \bmod M_{[n]} & = & A_{[n]} \bmod M_{[n]} \\
(4) & A_{[n]} \& (A_{[n]} | B_{[n]}) & = & A_{[n]} \\
(5) & \text{if } C \text{ then } A_{[n]} \text{ else } A_{[n]} & = & A_{[n]}
\end{array}
$$

Tricky rules

These rules are usually safe to apply, although potentially expensive. Many of them concern modular arithmetic, enabling important simplifications for PKC problems.
E.g.

$$
\begin{array}{lrcll}
(6) & A_{[n]} \leq k_{[n]} & = & \text{true} & \text{if } U(A_{[n]}) \leq k \\
(7) & A_{[n]} \bmod k_{[n]} & = & A_{[n]} & \text{if } U(A_{[n]}) < k \\
(8) & ((A_{[n]} \bmod M_{[n]}) + (B_{[n]} \bmod M_{[n]})) \bmod M_{[n]} & = & (A_{[n]} + B_{[n]}) \bmod M_{[n]} & \text{if } U(A_{[n]} + B_{[n]}) < 2^n \\
(9) & A_{[n]}[j:i] & = & 0_{[j-i+1]} & \text{if } G(U(A_{[n]})) < i
\end{array}
$$

where $U(A_{[n]})$ is an overapproximation of the bit-vector term $A_{[n]}$, and $G(n)$ is the index of the most significant 1-bit of the natural number n.

Handling conditionals

In order to solve PKC problems effectively, we need to perform plenty of simplifications involving *if-then-else* terms. Some of these simplifications, such as *if-lifting*, are potentially expensive and thus their application has to be restricted to certain cases.
E.g.

$$
\begin{array}{lrcll}
(10) & X_{[n]} \otimes (\text{if } C \text{ then } A_{[n]} \text{ else } B_{[n]}) & = & \text{if } C \text{ then } X_{[n]} \otimes A_{[n]} \text{ else } X_{[n]} \otimes B_{[n]} & \text{if } X_{[n]} \text{ contains no conditional} \\
(11) & (\text{if } C \text{ then } A_{[n]} \text{ else } B_{[n]}) \otimes (\text{if } C \text{ then } A'_{[n]} \text{ else } B'_{[n]}) & = & \text{if } C \text{ then } A_{[n]} \otimes A'_{[n]} \text{ else } B_{[n]} \otimes B'_{[n]} & \\
(12) & \text{if } C \text{ then } A_{[n]} \text{ else } B_{[n]} & = & A_{[n]} & \text{if } C \text{ is deduced from the context}
\end{array}
$$

Results

bveq is our prototype rewriting system implemented in Maude, and combined with the Z3 SMT solver (version 3.2) as described above.

Benchmark	bveq	Z3	Yices	CVC3
DES encrypt	0.68	**0.31**	1.62	25.53
SHA-1	**4.93**	OOM	T/O	T/O
SHA-2	T/O	OOM	T/O	OOM
Peasant multiplication 8-bit	**0.00**	24.15	37.97	14.52
Peasant multiplication 32-bit	**0.01**	T/O	T/O	OOM
Interleaving multiplication 8-bit	**2.94**	97.31	292.69	OOM
Interleaving multiplication 32-bit	OOM	T/O	T/O	OOM
Binary exponentiation 8-bit	**0.01**	308.19	1096.4	187.22
Binary exponentiation 128-bit	**0.25**	OOM	OOM	OOM

- Despite the potentially expensive rewrites, preliminary results suggest that the trade-off is worthwhile.
- Maude is great for experimentation but the problem requires a higher degree of fine-tuning.
  - Maximal subterm sharing (hash consing).
  - Efficient handling of logical context.
  - ...
- We are presently working in a framework / DSL to specify customized simplifiers for the UFBV theory.

This work is funded by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project PTDC/EIA-CCO/105034/2008.