Provably Optimal Code Generation using Logic Programming

Tom Crick

Department of Information Systems
University of Wales Institute, Cardiff (UWIC)
`tcrick@uwic.ac.uk`

The Optimisation Problem

WITHIN the field of compilers, the term optimisation is something of a misnomer. During the compilation process, compilers attempt to improve (with respect to both size and performance) the sequences of machine-level instructions it generates by applying a fixed set of transforms, reductions and equivalences. In many modern compilers, this can result in significant improvements, but it is unlikely to produce optimal sequences of instructions; and if it does, it will not be possible to determine that they are indeed optimal.

IN a significant range of applications, this approach to code generation is not sufficient; examples include resource-critical environments such as embedded domains, optimising compilers for the increasingly complex modern machine architectures and high-performance computing.

SUPEROPTIMISATION [4,5] is an approach that views code generation for acyclic code sequences as a combinatorial search problem. Rather than starting with crudely generated code and improving it, a superoptimiser starts with the specification of a function and performs a directed search for a sequence of instructions that meets this specification. Superoptimisation provides a fresh approach to the optimisation problem by aiming for optimality from the outset.

Answer Set Programming

ANSWER Set Programming (ASP) [1] is a declarative programming paradigm that allows reasoning about possible world views in the absence of complete information. It is a powerful and intuitive non-monotonic logic programming language for modelling, reasoning and verification tasks.

ASP describes a problem as a logic program in *AnsProlog*, a set of axioms and a goal statement, under the answer set semantics of logic programming in such a way that solving the problem is reduced to computing the answer sets of the program.

DUE to the increasing efficiency of its heuristic domain tools, known as solvers (such as CLASP, SMODELS, CMODELS and SUP), ASP is particularly suited to difficult (primarily NP-hard) search problems, making a number of problems tractable in the general case.

EXAMPLE applications of ASP to real-world problems include diagnostic reasoning, multi-agent systems, phylogenetics, biological networks, automatic music composition, evolutionary history of languages, cryptography, security engineering, instruction scheduling, program analysis and decision support systems for the NASA Space Shuttle.

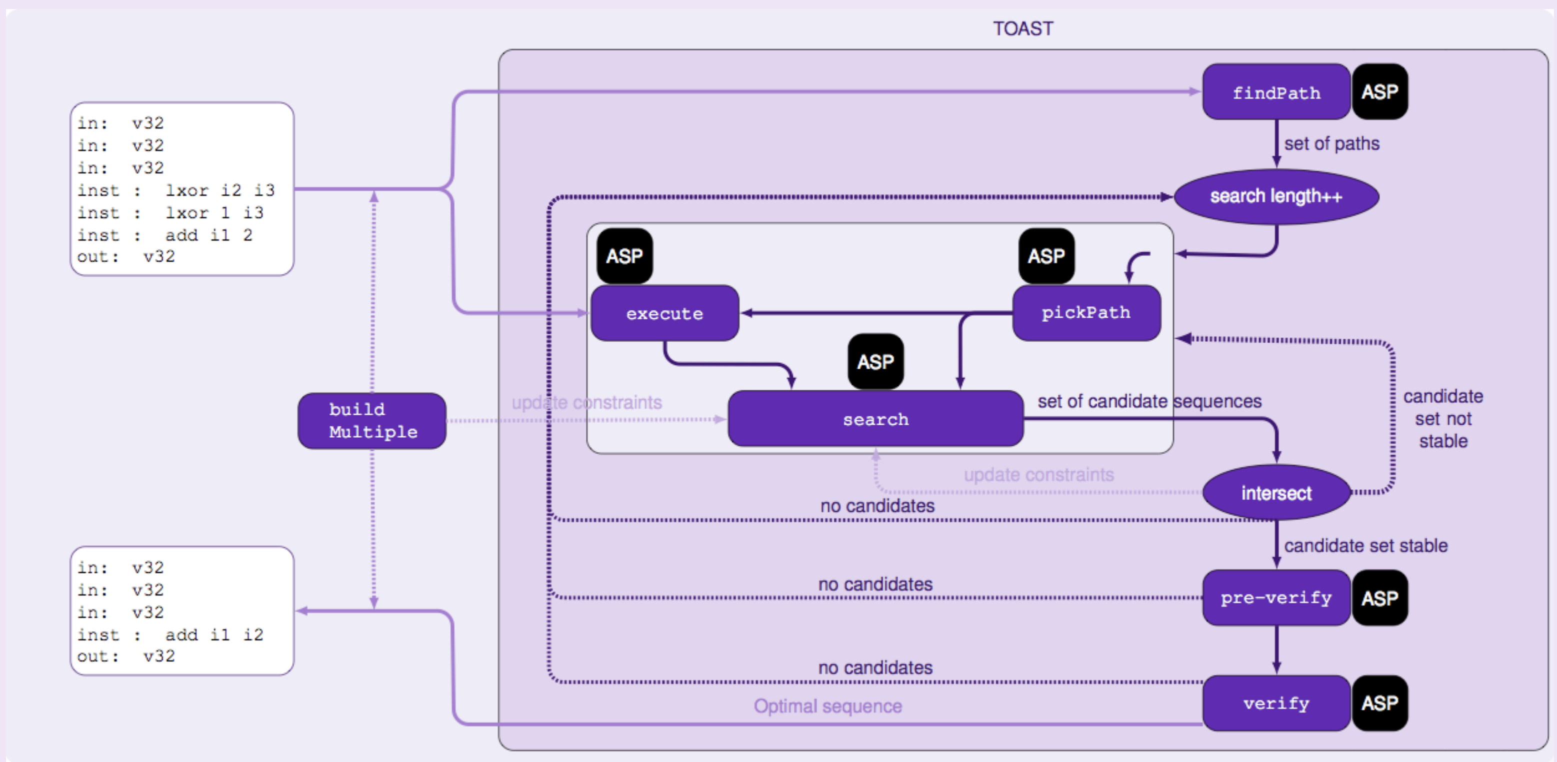
Solution: TOAST

THE *Total Optimisation using Answer Set Technology* (TOAST) [2,3] system uses ASP as the modelling and computational framework to solve the superoptimisation search problem. The motivation for the TOAST system is as follows:

- New structured approach to optimisation
- Lack of proven optimality of existing techniques
- Emergence of new performance-critical domains
- Modelling and computational power of ASP

TOAST consists of modular interacting components that generate *AnsProlog* programs, starting with a model of the microprocessor architecture, its instruction semantics and the original sequence to be optimised. A controlling interface utilises these components to generate a shorter, superoptimised version of the original sequence using off-the-shelf domain solving tools.

TOAST Framework



Experimental Results

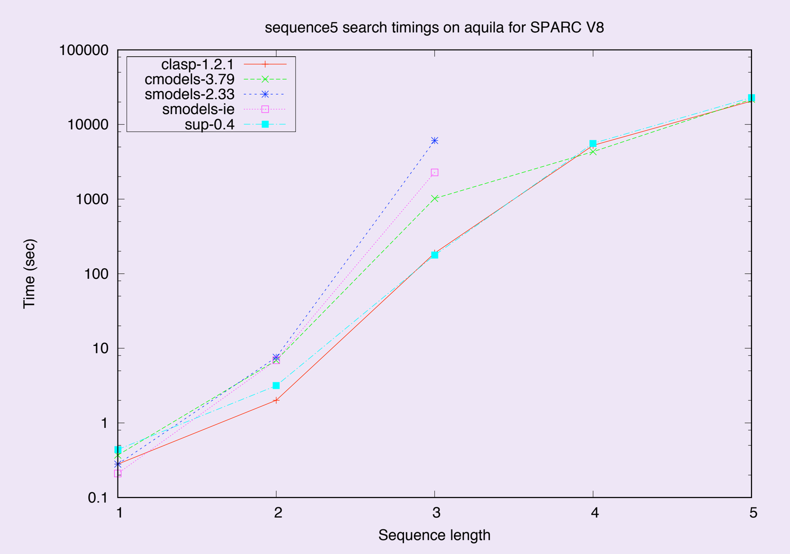
THE `sequence5` test is a sequence that is already optimal, giving an approximate ceiling on the performance of the system in searching over the large instruction space. `verifytest1` tests the (non-trivial) equivalence of two code sequences, while `verifytest2` tests the non-equivalence of two code sequences that only differ on one set of inputs. The table below presents timings for these search and verification tests for the SPARC V8, a popular 32-bit RISC architecture; solver time outs occurred after 200 hours.

	sequence5					verifytest1			verifytest2		
Solver	**1**	**2**	**3**	**4**	**5**	**8 bit**	**16 bit**	**32 bit**	**8 bit**	**16 bit**	**32 bit**
clasp-1.2.1	0.28	2.01	189	5211	20625	0.46	0.48	15.81	0.31	0.37	8.67
cmodels-3.97	0.37	6.89	1019	4314	21699	0.51	0.58	22.19	0.41	0.67	10.22
smodels-2.33	0.28	7.57	6100	t/o	t/o	0.18	11.33	t/o	0.20	4.75	t/o
smodels-ie-1.0.0	0.21	6.91	2279	t/o	t/o	0.20	11.08	t/o	0.21	4.79	t/o
sup-0.4	0.44	3.15	177	5596	23012	0.40	3.38	t/o	0.15	0.14	8.70
Atoms	853	1411	2098	2941	4196	904	2212	6940	1030	1526	2518
Rules	42740	118779	238212	410902	662049	1622	4870	17122	3591	6591	12583

Analysis

SUPEROPTIMISATION naturally decomposes into two sub-problems: *searching* for sequences that meet specific criteria and then *verifying* which of these candidates are functionally equivalent to the original sequence.

THE TOAST system is currently able to superoptimise sequences of five instructions in a practical time with current solving tools; this is a significant result considering empirical evidence for the average size of basic blocks (between 5-6 instructions). This can also be extended to superoptimise superblocks of instructions.



Conclusions and Future Work

- Development of a structured approach and adaptable framework to generating truly optimal code sequences is an important development for the domain.
- Superoptimisation of code is achievable in the general case and can be used to generate provably optimal code sequences for 32-bit architectures (and that doing the same for 64-bit architectures is also tractable).
- ASP is an appropriate paradigm for reasoning about large-scale, real-world problems. The flexibility of *AnsProlog* allows arbitrary constraints to be added to the search with minimal effort, something that is very difficult in the case of procedural superoptimisers.
- With further advances in solver technology and search heuristics, it is hoped that TOAST can be built into a competitive superoptimising system, especially for use as a peephole superoptimiser, via the generation of equivalence classes of code sequences with *buildMultiple*).
- Key future application areas would be in compiler toolchains such as GCC and JIT compilers, along with extensions to the modelling framework to handle multi-threaded and mult-core architectures. Also, focusing on the embedded domain, such as the ARM family of processors.

References

[1] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[2] Martin Brain, Tom Crick, Marina De Vos and John Fitch. *TOAST: Applying Answer Set Programming to Superoptimisation*. In ICLP 2006, volume 4079 of LNCS, pages 270–284. Springer, 2006.

[3] Tom Crick, Martin Brain, Marina De Vos and John Fitch. *Generating Optimal Code using Answer Set Programming*. In LPNMR 2009, volume 5753 of LNCS, pages 554–559. Springer, 2009.

[4] Torbjörn Granlund and Richard Kenner. *Eliminating Branches using a Superoptimizer and the GNU C Compiler*. In Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation (PLDI'92), pages 341–352. ACM Press, 1992.

[5] Henry Massalin. *Superoptimizer: A Look at the Smallest Program*. In Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), pages 122–126. IEEE Computer Society Press, 1987.

Cardiff's metropolitan university
prifysgol metropolitan Caerdydd

EPSRC
Pioneering research and skills

SET for Britain 2010, House of Commons, 8 March 2010