Foundations of a Generative C++ Platform for Generalised Satisfiability Problems

Tony H. Bao
University of Wales, Swansea

The 21st BCTCS, Nottingham 2005
SAT Solvers

• A “theoretical” SAT Solver - OKSolver (1998 - 2002)

• Won four awards in SAT Competition 2002

• Carefully handcrafted (like all existing SAT Solvers) to ensure high efficiency.
Tightly Coupled Components

SAT Solver

Monitoring

Data Structure
(annotated graph)

I/O

Algorithms
(Inference Engine)

Heuristics
(statistics)
Why not use libraries?
Libraries don’t deliver

- Exponential running time results in enormous scope for algorithms and implementations
- Many new algorithms and data structures are developed as well as many variations on existing algorithms and data structures.
Generality and abstraction that are required for a good library are (until now) always achieved at the expense of efficiency!
Generic Programming in C++

- Programming using types as generic parameters.
- Fundamental method that produces highly general and reusable components.
- Parametric static polymorphism.
- Realised in C++ by the powerful template mechanism.
Express software components in the most abstract form possible.

The abstract component can be instantiated back to a concrete implementation which is as efficient as a handcrafted one.

- Compiler performs optimisation on all source code.
- Template Meta-Programming fine tune optimisation and delivering general components (DSL)
Using generic and generative programming: Types include **classes**, classes include interface **and** implementation
Concepts

• Specifies syntactical, semantical and complexity requirements of generic parameters

• Concrete type that fulfil these requirements are said to be a **model** of the concept.

• Extended Abstract Data Type
In an ideal world.
Input a model and check it against a given concept
Output indicate where the problem is if any. Whether it’s syntactically incorrect or the semantics differs from our definition or complexity of the model doesn’t match.
Syntax

• Can be specified at source code level
• Boost Concept Checking Library (BCCL) that enforces these syntax rules at compile time.
Semantics

• How to express this?
• Correctness checker impossible.
Every concept comes with its own testing system
Complexity

- Prediction of resource usage for every operation of the model.

- In general, can only be done by running the operations.
• Stipulated complexity term belongs to the model. e.g.,

\[ t = \alpha \times 2^{\beta \times n} \]

• System does curve fitting to obtain alpha and beta. We then get the estimated resource prediction.
Families of Concepts

• Differentiation in semantics and complexity
• Additionally causes syntax changes
Generative Programming

- Built To Order components i.e., type inference
- Active library
- Configuration knowledge on top of components
- High level user spec input. System automatically works out a solution. Ideally
- In reality, a lot of help are needed for this
OKPlatform - The platform for comprehensive Satisfiability solution

- Generators
- SAT Solvers
- GUI
- Experimental System
- Documentation
- Statistical Analysis
OKPlatform

- Extreme Open Source
- Holistic approach
- Hackable, in a good way