Integration of a SAT Solver Into Maple

Stephen Forrest

ACA 2016, Kassel

Maplesoft Product Line

Maple™

MapleSim™

Maplesoft Engineering Solutions

Maple T.A.™

Möbius

About me

- Maplesoft employee 2001-2005, 2011-present, currently working from Germany
- Contributor to myriad parts of Maple (including `CodeGeneration`, `Logic`, connectivity routines)
- Prior experience with SAT is largely from a theoretical context (computational complexity)
- No prior experience with modern SAT/SMT solvers

Logic in Maple (1)

- Maple's builtin logic is three-valued (true, false, FAIL)
- Maple also has a package for interacting with formulas from (two-valued) propositional logic. Supported operations include:
  - **BooleanSimplify** – return an equivalent but more compact formula if one exists
  - **Dual** – construct the logical dual
  - **Equivalence** – check equivalence of two formulae
  - **Normalize** – convert to a normal form (CNF or DNF)
  - **Satisfy** – generate a satisfying truth assignment
  - **Tautology** – check if formula holds for all truth assignments
- Logic package has no notion of quantification

Logic in Maple (2)

- Logic package wasn't initially designed to handle large problems
- The algorithm of **Satisfy**'s original implementation was essentially:
  - Using de Morgan's law and the distributive law in the naïve way, convert the input formula to an arithmetic formula modulo 2, effectively using the ring $(\mathbb{Z}/2\mathbb{Z}, \oplus, \wedge)$ where $\oplus$, $\wedge$ denote XOR and AND respectively.
  - Traverse the resulting XOR expression and return an assignment corresponding to a (conjunctive) clause with the largest number of literals.
- Problems with exponential blow-up in size introduced by application of the distributive law are well-known
- Many immediate ways to make this faster and more scaleable!

Why add a SAT solver to Maple?

- Several high-performance, portable, and licence-compatible SAT solver implementations were available
- Integration of such a solver could hopefully assist in computations in many different domains:
  - Logical satisfiability (obviously)
  - Graph-theoretic problems, e.g. graph colouring, graph isomorphism
  - NP-hard combinatorial problems
  - Model checking

MiniSat

- **MiniSat** is “*a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT.*” [1]
- Developed and maintained by Niklas Eén & Niklas Sörensson since 2003
- Features include “*conflict-clause recording, conflict-driven backjumping, VSIDS dynamic variable order, two-literal watch scheme, [and] extensions for incremental SAT and for non-clausal constraints over boolean variables.*” [1]
- Written in portable C++
- Used by many SMT solvers
- Available under the MIT licence

DIMACS CNF file format

- *Problem*: MiniSat expects input in the **DIMACS CNF format**
  - Simple line-based text format in which a line contains a sequence of signed integers.
  - Integers denote literals (negated or non-negated terms)
  - The contents of a line together denote a clause
  - Example corresponding to $(x \vee y) \wedge (\neg x \vee y) \wedge (y \vee \neg z)$:

```
p cnf 3 3
  1  2  0
 -1  3  0
 2  -3  0
```

- Maple input may be an arbitrarily deep Boolean formula and include other logical connectives (implies, xor).

Tseitin transformation

- To capitalize on SAT solver speed, we will need to convert our input to CNF efficiently before writing out as DIMACS CNF.
- The *Tseitin transformation* is a well-known technique which transforms an arbitrary Boolean formula to an **equisatisfiable but not equivalent** Boolean in CNF, through the addition of a linearly-bounded number of new variables.
- First step: add a new, pure Maple routine **Tseitin** to **Logic:**

```
> Logic:-Tseitin(a xor b);
```

$$(a \vee b) \wedge (\neg a \vee \neg b)$$

```
> Logic:-Tseitin(a and not b or c);
```

$$(B1) \wedge (B0 \vee b) \wedge (\neg B0 \vee \neg b) \wedge (B1 \vee \neg B) \wedge (B1 \vee \neg c) \wedge (\neg B1 \vee B \vee c) \wedge (\neg B \vee a) \wedge (\neg B \vee B0) \wedge (B \vee \neg a \vee \neg B0)$$

MiniSat integration (1)

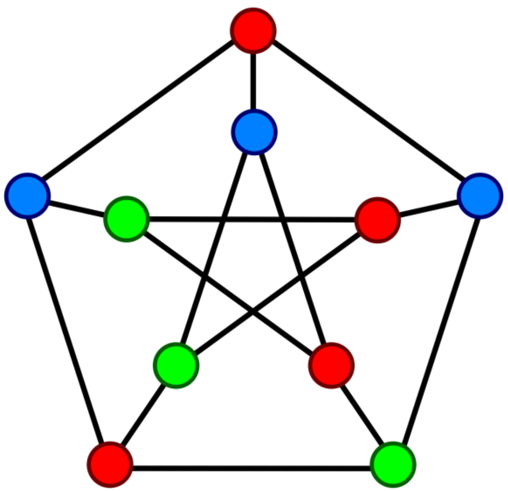
- Next step: add a copy of MiniSat to Maple packaged as a Maple *kernel extension*
- mplMiniSat is a C++ program which accepts a Maple expression sequence which contains:
  - a string (the DIMACS CNF encoding)
  - a boolean indicating whether a boolean result or a satisfying assignment is requested
  - some additional booleans for configurable options

MiniSat integration (2)

Parameter	Description	Default Value	Exposed in Maple?
var-decay	The variable activity decay factor	0.95	Yes
cla-decay	The clause activity decay factor	0.999	Yes
rnd-freq	The frequency with which the decision heuristic tries to choose a random variable	0	Yes
rnd-seed	Used by the random variable selection	1648253	No
ccmin-mode	Controls conflict clause minimization (0=none, 1=basic, 2=deep)	2	No
Rfirst	The base restart interval	100	No
rnd-init	Randomize the initial activity	False	Yes
phase-saving	Controls the level of phase saving (0=none, 1=limited, 2=full)	2	No
luby	Use the Luby restart sequence	True	No
rinc	Restart interval increase factor	2	No
gc-frac	The fraction of wasted memory allowed before a garbage collection is triggered	0.20	Yes

Applications: Graph colouring (1)

First effort: graph colouring



- Input: given graph with vertices $V = \{1, \ldots, n\}$ and edges $E \subseteq V \times V$.
- Define a SAT problem as follows:
  - Let vij represent the condition that vertex i is coloured by colour j.
  - For $1 \leq i \leq n$, assert that i gets some colour:
    $$vi1 \vee vi2 \vee \ldots \vee vik$$
  - For each $\{i, j\} \in E$, ensure i and j are never assigned the same colour:
    $$(\neg vi1 \vee \neg vj1) \wedge \cdots \wedge (\neg vik \vee \neg vjk)$$
  - Take the conjunction of all of these; resulting CNF expression has k ·|V| distinct variables and |V| + k ·|E| clauses
  - Complete graph: n^2 variables and $n + n^2\ (n-1)/2$ clauses

Applications: Graph colouring (2)

Computing a *k*-Colouring for the Complete Graph on *k* Vertices

k	Variables	Clauses	Time Taken (s)
6	36	96	0.007
7	49	154	0.055
8	64	232	0.066
9	81	333	0.284
10	100	460	3.22
11	121	616	241.80 (4.03 min)
12	144	804	2336.40 (38.94 min)

SAT integration: future work

Further continuation of this SAT integration could involve:

- More efficient encoding of hard problems as SAT instances (cf. Albert Heinle's talk)
- Proofs of unsatisfiability (i.e. returning an **unsatifiable core**)
- Other SAT solvers (#SAT or MAXSAT?)
- Further tweaking of MiniSat parameters?

Beyond SAT: SC-Square

- *"The research areas of SMT [SAT-Modulo-Theories] solving and symbolic computation are quite disconnected. On the one hand, SMT solving has its strength in efficient techniques for exploring Boolean structures, learning, combining solving techniques, and developing dedicated heuristics, but its current focus lies on easier theories and it makes use of symbolic computation results only in a rather naive way."*

  – Erica Ábrahám, ISSAC 2015 [2]

First steps: SMTLIB connectivity?

- **SMT-LIB** [4] is a standardized input language with support for:
  - A variety of theories (integers, reals, bit vectors) with and without quantification
  - Arbitrarily-nested functional inputs
  - Returning either a Boolean result or a satisfying witness
  - An assertion stack, to temporarily impose then remove assumptions to examine special cases without redoing the complete analysis

Some SMT Design issues

What issues exist for integration of SMT solvers with a CAS?

- Design issues general to all computer algebra systems (see [3])
  - How to use incremental solutions from SMT solvers?
  - How to guide SMT solvers with knowledge from the CAS?
  - How to use SMT for problems for which we seek more than an existential result?
- Design issues specific to Maple:
  - Inference of domains:
    - Intended domain of variables is usually implicit from the Maple command:

Integer	Floating point	Real	Complex
isolve, ifactor	evalf, fsolve	evalc, RealDomain:-solve	solve, factor

    - Difficult if problem uses mixed domains (e.g. Int/Real)
      - assuming facility can help here though support throughout library is not uniform
  - How best to express quantification?

References

1. Niklas Eén, Niklas Sörensson. **An Extensible SAT-solver.** SAT 2003: 502-518. http://minisat.se/downloads/MiniSat.pdf

2. Erika Ábrahám. **Building Bridges Between Symbolic Computation and Satisfiability Checking.** *Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation.* http://dl.acm.org/citation.cfm?id=2756636

3. E. Ábrahám, J. Abbott, B. Becker, A. M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J.H. Davenport , M. England, P. Fontaine, *et al*. **Satisfiability Checking and Symbolic Computation.** http://arxiv.org/pdf/1607.06945.pdf

4. C. Barrett, A. Stump, C. Tinelli. **The SMT-LIB Standard – Version 2.0**. http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r10.12.21.pdf

Maplesoft™

Mathematics • Modeling • Simulation

A Cybernet Group Company