Automated Theorem Proving
– Foundations –

September 2019

Wolfgang Küchlin

Symbolic Computation Group
Wilhelm-Schickard-Institute of Informatics
Faculty of Mathematics and Sciences

Universität Tübingen

Steinbeis Technology Transfer Centre
Object and Internet Technologies (STZ OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR

Contents (top level)

- Industrial applications of Logic
  - SAT Solving in automotive configuration
- SAT Solving: from Davis & Putnam (1960) to CDCL
- Automotive Examples
- Short outlook: Optimization

SR

Contents

- The AI-Context: The Automation of Reasoning
- Propositional Logic and First Order Proof
  - Proposition Decision Methods Overview
- Application Example: Automotive Configuration
- Preparation for Automated Theorem Proving
  - Conjunctive Normal Form CNF
  - Efficient CNF Conversion: Tseitin Transform
- From DP and DPLL to CDCL SAT-Solving
- More Automotive Configuration Problems and Examples
- From SAT to Optimization: Finding optimal models
  - MaxSAT algorithms

Areas and Methods of Classic AI

- Goal (strong AI): Build artificial Human
  - Weak AI: Machine which performs a limited task like a human
- Classic areas (since late 1950s / 1960s)
  - Vision; Speech understanding, Reasoning, Robotics
  - **Reasoning** (Nilsson. *Problem Solving Methods in AI*, 1971. [3]
    - Idea: not only mathematical thinking, but thinking in general, can be expressed as logical deduction! What is proof? A finite deduction in Predicate Calculus (PC)! [2]
    - Consistency Check in PC via Boolean consistency (Davis&Putnam 1960)
    - Resolution Theorem Proving in Predicate Calculus (J.A. Robinson, 1965)
    - Nilsson: Problem solving methods in AI, 1971. Searching („A*, alpha-beta“), planning („blocks world“), game playing („chess“, „tic-tac-toe“)
    - Algebraic: symbolic integration heuristics (later replaced by algorithm)
  - Infrastructure: Programming languages Lisp, Prolog
    - Symbolic computation, dynamic heap storage, recursion

Short history of AI Reasoning with Mathematical Logic

- Math. reasoning demonstrates intelligence! (really ?)
- Proof procedures (calculi) + search heuristics (intell.?)
  - Propositional Calculus: DPLL SAT-Solving: Davis & Putnam (J.ACM 1960), Davis & Putnam & Logemann & Loveland (C.ACM 1962). No deduction here.
  - Predicate Calculus: „Resolution w. Unification“ : J.A. Robinson (J.ACM 1965)
    - Resolution is deduction, both propositional and 1st order. Idea: deduction = resoning.
  - Largely textbook examples from 1960—1990; PROLOG came and went
- Applicability of SAT-Solving
  - First order inconsistency (Davis&Putnam 1960); hence first order proof!
  - Computer Hardware verification (switching algebra: microelectronic circuits)
  - Computer Software verification: compile program to Boolean circuit
  - Configuration: Product description (product variance), variant parts list (BoM)
- 1990s: algorithmic break-through in SAT, industrial applications
  - Intel floating point bug pushes CDCL SAT-Solving (DPLL + resolution)
  - Conflict Driven Clause Learning SAT: 100.000s clauses and variables

The origins of SAT Solving

- Approach: Enumerate the Herbrand Universe of a first order formula and check each enumeration level for **Boolean consistency**.
- Martin Davis, Hilary Putnam: A Computing Procedure for Quantification Theory. J.ACM 7, 1960.

"The hope that mathematical methods employed in the investigation of formal logic would lead to purely computational methods for obtaining mathematical theorems goes back to Leibniz and has been revived by Peano around the turn of the century and by Hilbert's school in the 1920's. Hilbert, noting that all of classical mathematics could be formalized within quantification theory, declared that the problem of finding an algorithm for determining whether or not a given formula of quantification theory is valid was the central problem of mathematical logic."

  - Method (by hand): Unit propagation + pure literal elimination + resolution
- Gilmore: A proof method for quantification theory, 1960
  - Method (implemented on IBM 704): DNF conversion
- Quine: A proof procedure for quantification theory. 1955.
  - Method: Truth tables

In Defence of Propositional Logic / Boolean Algebra

- The Real World is finite and discrete !
  - In practice, even time is finite and discrete.
  - Boolean Algebra is enough: „the method of thought“
- Higher logics are rather hard to understand and use
  - Even Boolean Algebra is hard with larger formulae
- Boolean Logic is decidable
- Decision procedures are fast, they scale, they terminate
- Fully automatic theorem proving: no human intervention
- Large scale production use in industry
  - semi conductor
  - automotive: Daimler, BMW, VW / AUDI, OPEL, GM, Renault, …

Some Boolean Decision Procedures

- Bad news: SAT is NP-complete (Cook 1972)
- Good News: SAT(F) is decidable and solves all NP-complete problems!
  - Truth Tables → guaranteed exponential (#variables), toy problems only
  - Disjunctive normal form (DNF) → easily exp., small problems ok.
  - Tableaux → similar to DNF, easily exp., small problems ok.
  - Boolean Polynomials → „Stone polynomials“, canonical form, little use.
  - Binary Decision Diagrams (ROBDD) → model checking use, 100s variables ok, O(1) SAT-solving, easy model counting, canonical form.
  - Propositional Resolution → too many deductions, theoretical importance
  - Davis-Putnam-Logemann-Loveland (DPLL) → small problems ok.
  - DPLL based CDCL SAT-Solving → practically efficient for science and industry, 100,000+ variables, method of choice, very robust, much research.

Application Example: *Mercedes Configuration.*

Example: E-Class

- approx. 1.500 codes (options, countries, ..)
- approx. 3.000 rules in product description
- rule based BOM with approx. 35.000 parts


```
B(P09) = 297+540+543;

B(610) = (512 / 527 / 528)+608+ -978;

Z(P09) = (M271 + -M013 / M272) + 830 /
          Z04+ -(M273 + Z27)

Z(682) = 623 / 830 / 513L

B(450) = L+965+ 670+837+ -P34+
((M271/M651)+(953/955)+(100A/200A)+(334/335)+(301/336/337) /
 / M642+(2XXL/557L/571L)+(953/955)+(100A/200A)+(334/335)+ (301/337));
```

Application Example: *Mercedes Configuration.*

Example: E-Class

- approx. 1.500 codes (options, countries, ..)
- approx. 3.000 rules in product description
- rule based BOM with approx. 35.000 parts

```
B(P09) = 297+540+543;

B(610) = (512 / 527 / 528)+608+ -978;

Z(P09) = (M271 + -M013 / M272) + 830 /
         Z04+ -(M273 + Z27)

Z(682) = 623 / 830 / 513L

B(450) = L+965+ 670+837+ -P34+
((M271/M651)+(953/955)+(100A/200A)+(334/3
 / M642+(2XXL/557L/571L)+(953/955)+(100A/2
```

P09: Sonnenschutzpaket
297: Rollos Fond-Türen links+rechts
540: Rollo elektr. Heckfenster
543: Sonnenbl. li+re m. Make-up-Sp.
610: Nachtsichtsystem
512: Comand APS mit DVD-Wechsl.
527: Comand DVD APS mit NAVI
528: Comand DVD+ ohne NAVI
608: Autom. Fernlichtschaltung
978: Spezial-Fahrgestell
682: Feuerlöscher
830: Zusatzteile China
623: Golf-Staaten-Ausführung
513L: Belgien
M271: R4-Ottomotor
M272: V6-Ottomotor
M273: V8-Ottomotor
M013: Motor leistungsreduziert
Z04: B4 Panzerung
Z27: Bodenpanzerung
450: TAXI-International

Application Example: *Checking Configuration Options*

- Car order is a list L of options (option codes)
  - Codes in L are *true*, all others are *false* (for this car)
- Compile all configuration rules into a single formula **C**
  - Solutions $\beta(\mathbf{C})$=true are constructible orders
  - Restriction $\mathbf{C}|_{o=true}$ sets o=true in **C**: all cars with option o
- Checking the configuration rule-set
  - Is option o available? → SAT($\mathbf{C}|_{o=true}$) = true?
  - Is option o forced? → SAT($\mathbf{C}|_{o=false}$) = false?
  - Is option o available in country c? → SAT($\mathbf{C}|_{c=true,o=true}$) = true?
  - Is option o forced in country c? → SAT($\mathbf{C}|_{c=true,o=false}$) = false?
- Configuration step
  - Is option o available after selecting options $o_1,\ldots,o_n$?
    → SAT($\mathbf{C}|_{o1=true,\ldots,on=true,\ o=true}$) = true?

SAT Software Verification (Bounded Model Checking)

CBMC Advanced Example 2

Function: `abs(x)`. Property: `abs(x) >= 0`.

```
short myabs(short a) {
  short result;
  if (a < 0)
    result = -a;
  else
    result = a;
  assert(result >= 0);
  return result;
}
```

CBMC output:

```
  abs::myabs::1::result=-32768 (1000000000000000)

Violated property:
  file test.c line 8 function myabs
  assertion
  result >= 0

VERIFICATION FAILED
```

Result

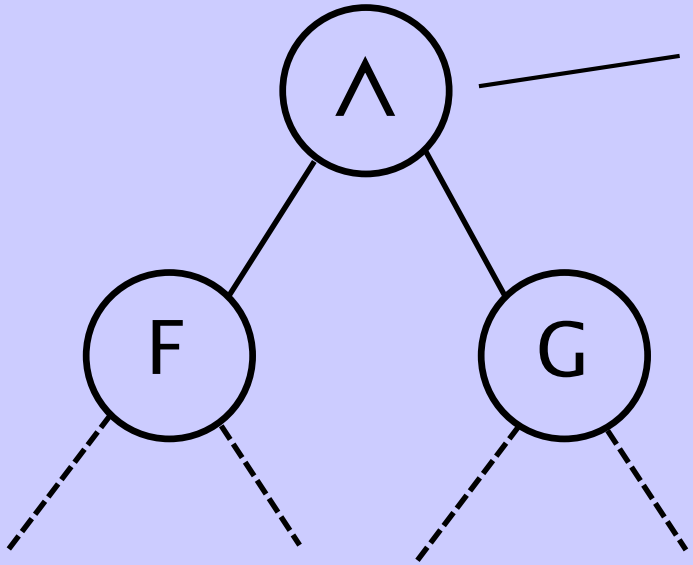
Explanation by counterexample

Conjunctive Normal Form (CNF)

- Natural representation of an axiom / constraint system
- Resolution and DPLL / CDCL SAT-Solving require CNF
- CNF is a conjunction ($\wedge$) of *clauses* ($\ell_1 \vee \ \ldots \ \vee \ \ell_n$)
  - The ℓ_i are *literals* (positive or negative variables)
  - Example: $F = (x \vee \neg y) \wedge (x \vee \neg z) \wedge (z \vee \neg y) \wedge (z \vee \neg x)$
  - Simplified set notation: $F = \{\{x, \neg y\}, \{x, \neg z\}, \{z, \neg y\}, \{z, \neg x\}\}$
  - CNF lists a set of (simple) *constraints*, which must be **simultaneously** satisfied for F=1.
  - CNF enumerates conditions for $F \neq 0$ (negations of F´s roots).
- CNF conversion (theoretically)
  - Push negations ¬ inside → negation normal form NNF
  - Apply distributivity law: $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$

CNF Transformation

- In the worst case, the size of CNF(F) is exponential in F
- Solution: Tseitin´s transformation CNF-T(F) is
  - in CNF
  - linear in F
  - SAT-equivalent to F [i.e. CNF-T(F) is SAT *iff* F is SAT]



- Introduce Tseitin Variable $x_\wedge$ with $x_\wedge \leftrightarrow F \wedge G$. In CNF:
- {{¬x, F}, {¬x, G}, {¬F, ¬G, x}}
- Ind. Hypothesis:
  F and G are Tseitin variables
- Plaisted & Greenbaum:
  {¬F, ¬G, x} not necessary

Literature

1. Thomason, Richmond, "Logic and Artificial Intelligence", *The Stanford Encyclopedia of Philosophy* (Winter 2018 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/win2018/entries/logic-ai/>.
2. Bringsjord, Selmer and Govindarajulu, Naveen Sundar, "Artificial Intelligence", *The Stanford Encyclopedia of Philosophy* (Fall 2018 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/fall2018/entries/artificial-intelligence/>.
3. Nils J. Nilsson. *Problem Solving Methods in Artificial Intelligence*. Mc Graw Hill 1971. (Historical usage of Automated Theorem Proving in AI)
4. Mordechai Ben-Ari. Mathematical Logic for Computer Science. Springer. 3rd ed. 2012. (Excellent textbook with broad coverage of methods.)
5. Michael Huth and Mark Ryan. *Logic in Computer Science*. Cambridge University Press 2004. (Focus on automated verification.)
6. A. Biere, M. Heule, H. van Maaren, T. Walsh (eds.). *Handbok of Satisfiability*. IOS Press 2009. (Comprehensive current account of SAT based methods)
7. W. Küchlin, C. Sinz. *Proving Consistency Assertions for Automotive Product Data Management*. J. Automated Reasoning 24 (1—2), 2000.