Automated Theorem Proving

– Foundations –

July 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

- The AI-Context
- Propositional Algebra
- Satisfiability, Validity, Entailment
- Application Example: Automotive Configuration
- Decision Methods Overview
- Preparation for Automated Theorem Proving
  - Disjunctive Normal Form DNF
  - Conjunctive Normal Form CNF
  - Efficient CNF Conversion Methods
  - Tseitin Transform

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! [2]
    - Resolution Theorem Proving (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

Satisfiability and Validity

- A formula F is *satisfiable* if it has a *model*. A model is a valuation β: Var(F) $\rightarrow$ {1, 0} s.th. β(F)=1. We write $\beta \vDash F$ or $\vDash_{\beta} F$. We write SAT(F)=true if F is satisfiable, else SAT(F)=false if F is unsatisfiable.
  - The interpretation of the logical connectives is fixed, so β only depends on the valuation of the propositional variables. There are no function or predicate symbols (and no quantifiers).
- A formula F is *valid* ($\vDash F$), if β(F)=1 for all β.
- F is valid iff. ¬F is unsatisfiable.
  - Since β(F)=1 iff. β(¬F)=0, and β(F)=0 iff. β(¬F)=1.

Semantic Entailment

- Formula F entails formula G under a valuation β ($F \vDash_\beta G$) (resp. G follows from F), if $\beta(G)=1$ whenever $\beta(F)=1$.

  (β must completely assign all variables in F and G)
- If $F \vDash_\beta G$ for *all* complete valuations β, then G follows from F, and we write $F \vDash G$.
- Proposition: $F \vDash G$ iff $F \wedge \neg G \vDash \bot$
  - Let $F \vDash G$. Then $\beta(\neg G)=0$ whenever $\beta(F)=1$.
  - Let $F \wedge \neg G \vDash \bot$. Hence if $\beta(F)=1$ then $\beta(\neg G)=0$, and so $\beta(G)=1$.
- Corollary: $F \vDash G$ iff SAT($F \wedge \neg G$) = false (and $\vDash \neg F \vee G$)
  - There is no counterexample where $\beta(F)=1$ and $\beta(G)=0$
- → Semantic entailment can be proved by SAT-Solving

Propositional Algebra: Equivalences

We write F ≡ G if $\models (F \leftrightarrow G)$.

- $\beta(F) = \beta(G)$ for all valuations β.
- Attn: ≡ is a meta-symbol, not a propositional connective!
- ≡ denotes an equivalence relation

➢ Distributivity:

$F \lor (G \land H) \equiv (F \lor G) \land (F \lor H)$ $\quad$ $F \land (G \lor H) \equiv (F \land G) \lor (F \land H)$

➢ Absorption:

$F \lor (F \land G) \equiv F \land (F \lor G) \equiv F$

➢ DeMorgan´s Laws:

$\neg(F \land G) \equiv \neg F \lor \neg G$ $\quad$ $\neg (F \lor G) \equiv \neg F \land \neg G$

➢ Commutativity and Associativity of $\land$, $\lor$.

➢ … and many more …

Decision Procedures and SAT-Solving

- Bad news: SAT is NP-complete (Cook 1972)
  - No known algorithm decides SAT (true/false) in polynomial time
- Good News: SAT(F) is decidable!
  - Truth Tables → guaranteed exponential, toy problems only
  - Disjunctive normal form (DNF) → easily exp., small problems ok.
  - Tableaus → similar to DNF, easily exp., small problems ok.
  - Boolean Polynomials → 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) → toy problems
  - DPLL based CDCL SAT-Solving → practically efficient for science and industry, 100,000+ variables, method of choice, very robust, much research.

Method Example: Truth Table and DNF

$F = ((x \wedge \neg(y \to z)) \oplus x)$

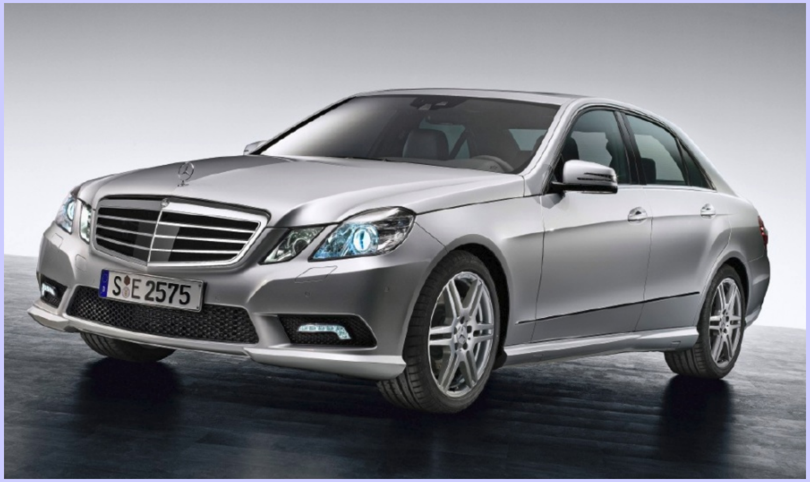
x	y	z	$y \to z$	$\neg(y \to z)$	$x \wedge \neg(y \to z)$	$(x \wedge \neg(y \to z)) \oplus x$
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	0	0	1	0	0	1
1	0	1	1	0	0	1
1	1	0	0	1	1	0
1	1	1	1	0	0	1

F is satisfiable, but not valid. Exponential number of rows.

DNF: enumerate all points =1

$DNF(F) = (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z) = (x \wedge \neg y) \vee (x \wedge y \wedge z)$

Application Example: *Mercedes Configuration.*

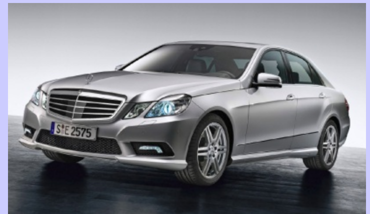


Example: E-Class

- 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 $o_1,\dots,o_n$?

    → $SAT(\mathbf{C}|_{o1=true,\dots,on=true,\ o=true}) = true$?

Conjunctive Normal Form (CNF)

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

Example: CNF from function table

F = ((x∧¬(y→z))⊕x)

x	y	z	y→z	¬(y→z)	x∧¬(y→z)	(x∧¬(y→z)) ⊕x
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	0	0	1	0	0	1
1	0	1	1	0	0	1
1	1	0	0	1	1	0
1	1	1	1	0	0	1

CNF: negate all roots: ¬(¬x) ∧ ¬(x∧y∧¬z) ≡ x ∧ (¬x ∨¬y ∨**z**)
≡ x ∧ (¬y ∨**z**)

Complexity of CNF-Transformation

- Application of Distributivity Law may double the formula size
  - $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
- Worst case: CNF(F) exponentially larger than F
  - Very serious impediment for applications
- Example: $(x_{11} \wedge x_{12}) \vee \ldots \vee (x_{n1} \wedge x_{n2})$.
  - CNF contains 2^n clauses with n variables each (→ Induction)
- Solution in 2 steps
  1. Tseitin transformation
  2. Plaisted-Greenbaum Transformation
- Transformations maintain satisfiability, not equivalence!

CNF: Tseitin-Transformation

- Key idea:
  - introduce auxiliary variables to represent (sub-)formulae
  - introduce additional constraints to link variables to formulae
- General concept, we use it only for CNF
  - In $F \vee (G \wedge H)$, represent $(G \wedge H)$ by new x
  - Add constraint $x \leftrightarrow (G \wedge H)$.
  - Hence $F \vee (G \wedge H)$ is replaced by $(F \vee x) \wedge (x \leftrightarrow (G \wedge H))$
  - Now convert $x \leftrightarrow (G \wedge H)$ to $x \rightarrow (G \wedge H)$ and $(G \wedge H) \rightarrow x$, giving clauses $(\neg x \vee G)$, $(\neg x \vee H)$, $(\neg G \vee \neg H \vee x)$ in addition to $(F \vee x)$.
  - This is the „full“ Tseitin-Transformation
  - Only satisfiability is preserved: $F \vee (G \wedge H) \backsimeq (F \vee x) \wedge (x \leftrightarrow (G \wedge H))$
  - Model count is preserved
- Bad news?: Now G and H are doubled instead of F!?

CNF: Tseitin-Transformation

- Good news!
  - Think recursively: May assume G and H are already represented by variables g and h.
  - Then we get clauses (¬x∨g), (¬x∨h), (¬g∨¬h∨x) in addition to (F∨x) where only 2 variables are doubled
  - and only one set of 3 clauses and 3 variables each for g and h.
- Plaisted & Greenbaum Transform: (F∨x)∧(x → (G∧H))
  - David A. Plaisted, Steven Greenbaum: A Structure Preserving Clause Form Transform. *J. Smbolic Computation* Vol 2(3), 1986.
  - Often also misnamed „Tseitin-Transform“
  - Drops constraint ((G∧H) → x)
  - Model count may (but need not) double with every aux. Variable
  - Theorem: F∨(G∧H) ≅ (F∨x)∧(x → (G∧H))
    - Intuition: if G and H are satisfied, wie don‘t care about the Tseitin variable

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.