Propositional Satisfiability (SAT): Introduction

Tjark Weber

+ slides by Daniel Le Berre

Introduction to SAT

History, Algorithms, Practical considerations

Daniel Le Berre[1]

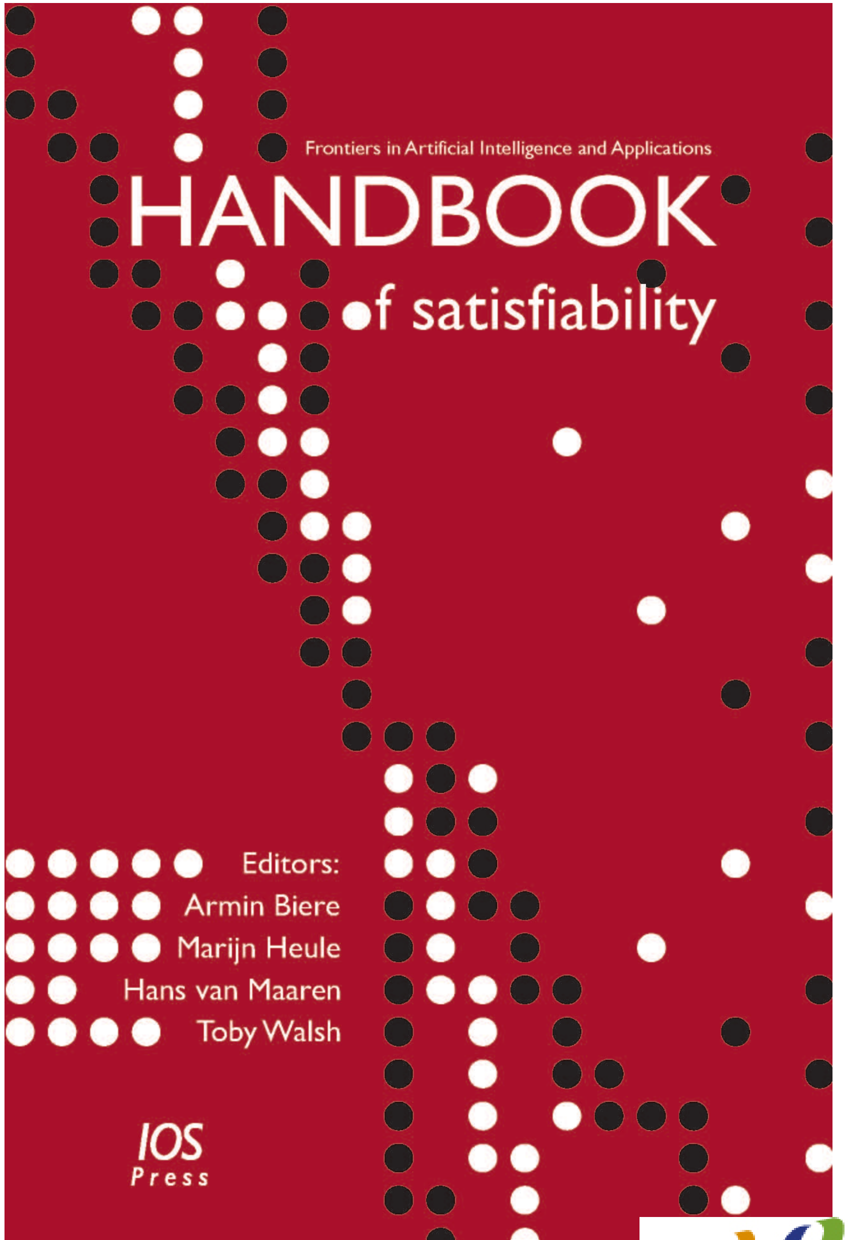
CNRS - Université d'Artois

SAT-SMT summer school
Semmering, Austria, July 10-12, 2014

[1] Contains material provided by Joao Marques Silva, Armin Biere, Takehide Soh

Disclaimer

- Not a complete view of the subject
- Limited to one branch of SAT research (CDCL solvers)
- From an AI background point of view
- From a SAT solver designer
- For a broader picture of the area, see the handbook edited in 2009 by the community



UNIVERSITÉ D'ARTOIS

Disclaimer: continued

- The best solvers for practical SAT solving in the 90's were based on local search or randomized DPLL
- Since then, the best performing solvers are based on the Conflict Driven Clause Learning architecture.
- The current challenge is to create a new kind of solvers targeting parallel architectures ...

UNIVERSITÉ D'ARTOIS

Context: SAT receives much attention since a decade

Why are we all here today?

- Most companies doing software or hardware verification are now using SAT solvers.
- SAT technology indirectly reaches our everyday life:
  - Intel core I7 processor designed with the help of SAT solvers [Kaivola et al, CAV 2009]
  - Windows 7 device drivers verified using SAT related technology (Z3, SMT solver) [De Moura and Bjorner, IJCAR 2010]
  - The Eclipse open platform uses SAT technology for solving dependencies between components [Le Berre and Rapicault, IWOCE 2009]
- Many SAT solvers are available from academia or the industry.
- SAT solvers can be used as a black box with a simple input/ouput language (DIMACS).
- The consequence of a new kind of SAT solver designed in 2001 (Chaff)

The SAT problem: theoretical point of view

Definition

Input: A set of clauses C built from a propositional language with n variables.
Output: Is there an assignment of the n variables that satisfies all those clauses?

The SAT problem: theoretical point of view

Definition

Input: A set of clauses C built from a propositional language with n variables.
Output: Is there an assignment of the n variables that satisfies all those clauses?

Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For C_1, the answer is yes, for C_2 the answer is no

$$C_1 \models \neg(a \wedge \neg c) = \neg a \vee c$$

The SAT ~~problem~~ solver: practical point of view

Definition

Input: A set of clauses C built from a propositional language with n variables.
Output: If there is an assignment of the n variables that satisfies all those clauses, provide such assignment, else provide a subset of C which cannot be satisfied.

The SAT ~~problem~~ solver: practical point of view

Definition

Input: A set of clauses C built from a propositional language with n variables.
Output: If there is an assignment of the n variables that satisfies all those clauses, provide such assignment, else provide a subset of C which cannot be satisfied.

Example

$$C_1 = \{\neg a \vee b, \neg b \vee c\} = (\neg a \vee b) \wedge (\neg b \vee c) = (a' + b).(b' + c)$$

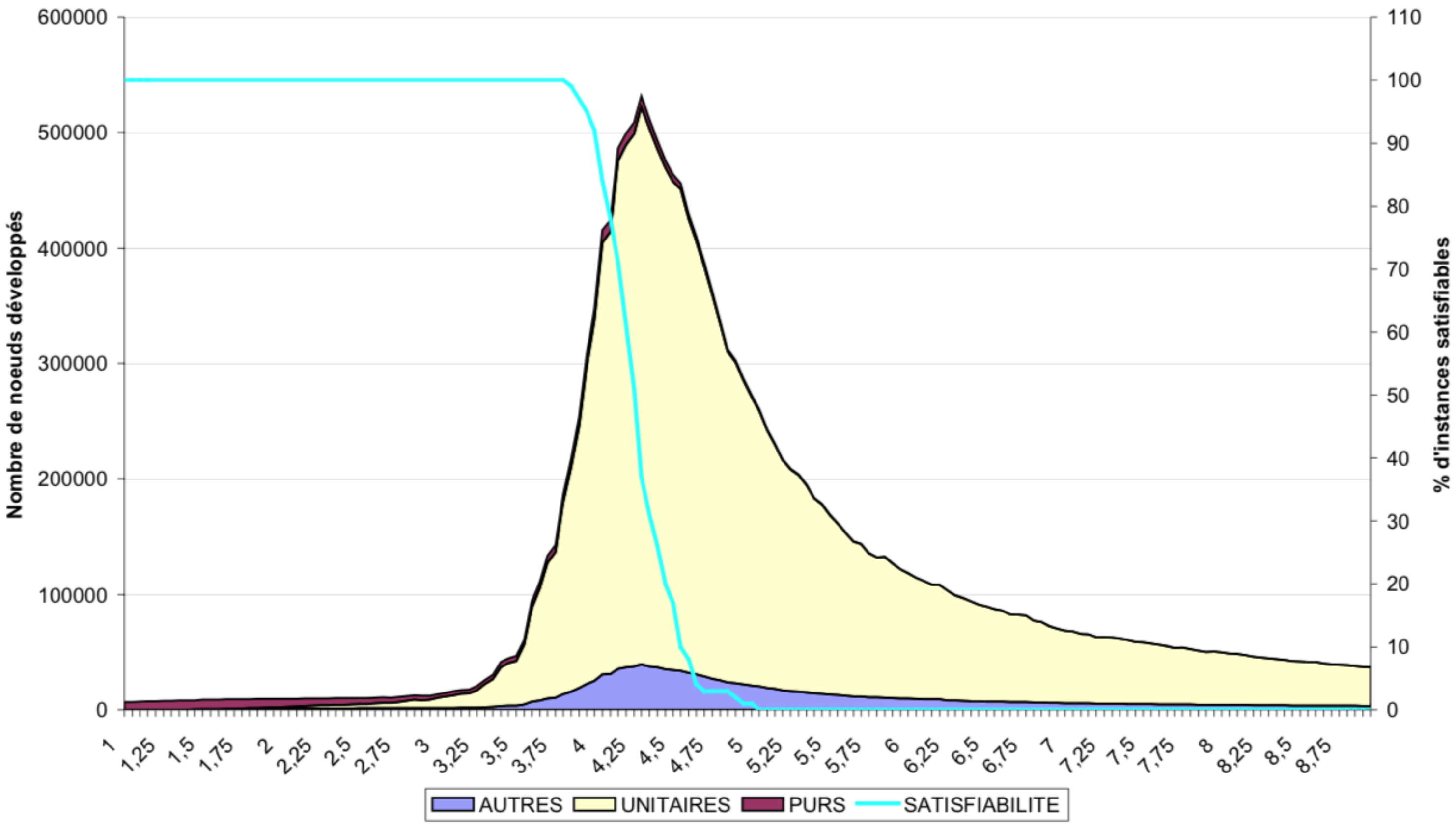
$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \wedge a \wedge \neg c$$

For C_1, one answer is $\{a, b, c\}$, for C_2 the answer is C_2

SAT is important in theory ...

- Canonical NP-Complete problem [Cook, 1971]
- Threshold phenomenon on randomly generated k-SAT instances [Mitchell,Selman,Levesque, 1992]

Proportion des différentes propagations pour un DP MOMS



Example: 1 to 9 ratio $\frac{\#clauses}{\#variables}$ for $k = 3$

UNIVERSITÉ D'ARTOIS

… in practice: Computer Aided Verification Award 2009

awarded to

Conor F. Madigan
Sharad Malik
Joao Marques-Silva
Matthew Moskewicz
Karem Sakallah
Lintao Zhang
Ying Zhao

for

fundamental contributions to the development of high-performance Boolean satisfiability solvers.

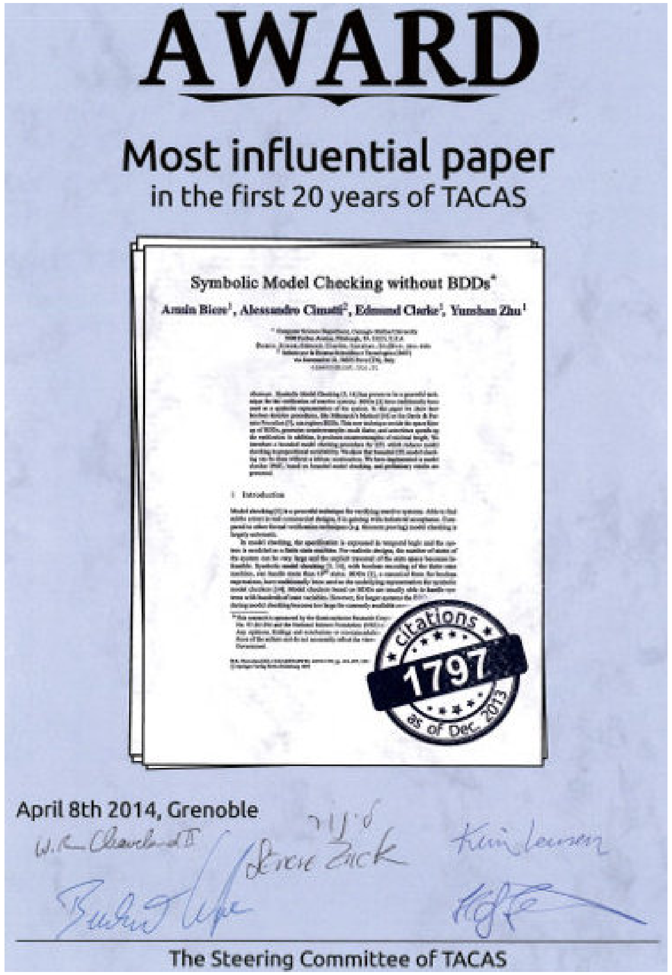
Authors of GRASP SAT solver
Authors of CHAFF SAT solver

awarded to

A. Biere
A. Cimatti
E. Clarke
Y. Zhu

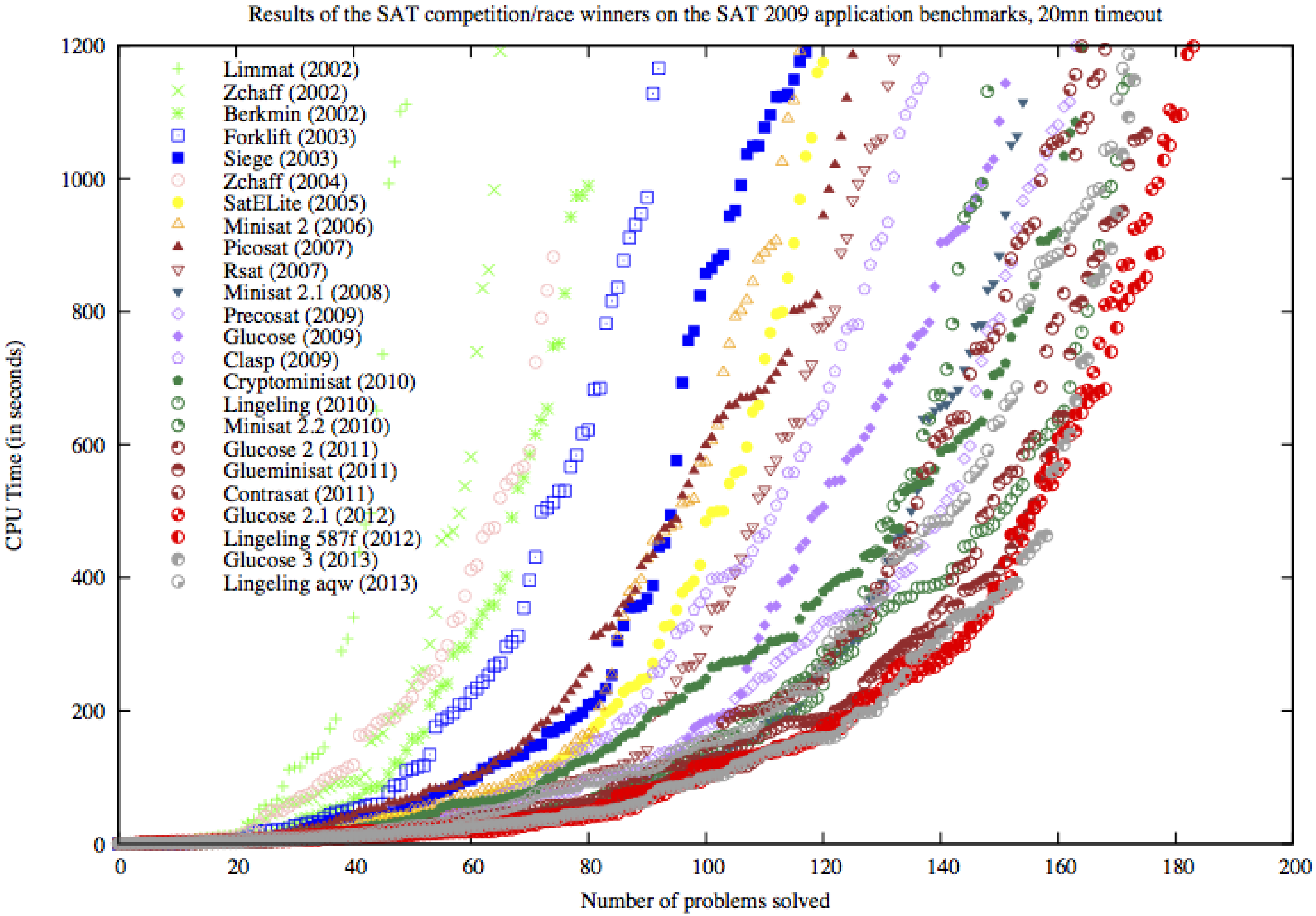
for

Symbolic Model Checking without BDDs



UNIVERSITÉ D'ARTOIS

Evolution of the performance of some SAT solvers



UNIVERSITÉ D'ARTOIS

Where can we find SAT technology today?

- Formal methods:
  - Hardware model checking; Software model checking; Termination analysis of term-rewrite systems; Test pattern generation (testing of software & hardware); etc.
- Artificial intelligence:
  - Planning; Knowledge representation; Games (n-queens, sudoku, social golfers, etc.)
- Bioinformatics:
  - Haplotype inference; Pedigree checking; Analysis of Genetic Regulatory Networks ; etc.
- Design automation:
  - Equivalence checking; Delay computation; Fault diagnosis; Noise analysis; etc.
- Security:
  - Cryptanalysis; Inversion attacks on hash functions; etc.

Where can we find SAT technology today? II

- Computationally hard problems:
  - Graph coloring; Traveling salesperson; etc.
- Mathematical problems:
  - van der Waerden numbers; Quasigroup open problems; etc.

- Core engine for other solvers: 0-1 ILP/Pseudo Boolean; QBF; #SAT; SMT; MAXSAT; ...
- Integrated into theorem provers: HOL; Isabelle; ...
- Integrated into widely used software:
  - Suse 10.1 dependency manager based on a custom SAT solver.
  - Eclipse provisioning system based on a Pseudo Boolean solver.
  - Eiffel language uses Z3 to check contracts.