RESEARCH @ ECLIPSE FOUNDATION

Open Research Webinars

OW2

Sat4j, from the lab to OW2 with and for Eclipse

Daniel Le Berre, Université d'Artois and CNRS

UNIVERSITÉ D'ARTOIS

cnrs

Agenda

- What does SAT stand for?
- What is SAT4J?
- Why OW2 is important for SAT4J?
- How is SAT4J related to Eclipse?
- Some remarks about Academic Software in the Economic world

What does SAT stand for?

Academic and practical considerations

The SATisfiability Problem

- A combinatorial problem on Boolean variables
- Simple to express
- Simple to check that the provided solution is correct
- First problem shown to be NP-complete in 1971: *they are instances of that problem that we cannot solve in reasonable time with a computer*

The simplest hard problem in computer science

Academic view of SAT

- Boolean variables
- Clauses (disjunction of literals)
- Formula (conjunction of clauses)
- Problem: can I assign the variables in such a way that all clauses are satisfied?
- Decision problem: answer is yes or no
  Can also ask for the assignment (model)

$$v_1, v_2, v_3, v_4, v_5, v_6$$

$$\neg v_1 \vee v_2 \vee \neg v_4$$

$$v_2 \vee v_4$$

$$\neg v_1 \vee v_4 \vee v_5$$

$$\neg v_5 \vee \neg v_6$$

Around SAT: Many related problems

- SAT: is it possible to satisfy all the clauses? Find any solution.
- PBO (Pseudo Boolean Optimization)
  - Cardinality constraints, Pseudo-Boolean Constraints
  - Objective function to optimize
- MAXSAT: satisfy as many clauses as possible
- #SAT: count the number of solutions

Academic practitioner view of SAT: SAT solvers

- File based input (Dimacs format)
- Provides an assignment if satisfiable
- Else
  May provide a proof if unsatisfiable
  May also provide an unsatisfiable subset of the formula if unsatisfiable
- The answer of the solver can be checked

```
p cnf 6 6
1 0
-2 3 0
-1 2 -4 0
2 4 0
-1 4 5 0
-5 -6 0
```

Name	APPLICATIONS/c32sat/ post-cbmc-zfcp-2.8-u2-noholes.cnf
MD5SUM	c4aa2ddc60eee766fbdf95a32eed5b43
Bench Category	APPLICATION (applications instances)
Best result obtained on this benchmark	SAT
Best CPU time to get the best result obtained on this benchmark	51.5012
Satisfiable	
(Un)Satisfiability was proved	
Number of variables	10950109
Number of clauses	32697150
Sum of the clauses size	76320846
Maximum clause length	65
Minimum clause length	1
Number of clauses of size 1	2415
Number of clauses of size 2	21783823
Number of clauses of size 3	10907882
Number of clauses of size 4	1592
Number of clauses of size 5	131
Number of clauses of size over 5	1307

Results of the different solvers on this benchmark

Solver Name	TraceID	Answer	CPU time	Wall clock time
SApperIoT *base* (complete)	1563400	SAT	51.5012	204.435
picosat *913* (complete)	1563397	SAT	116.704	120.088

General information on the benchmark

Name	CRAFTED/sgen/unsat/ sgen1-unsat-121-100.cnf
MD5SUM	24865fea2f97ba63d115c240cb3cb69f
Bench Category	CRAFTED (crafted instances)
Best result obtained on this benchmark	
Best CPU time to get the best result obtained on this benchmark	
Satisfiable	
(Un)Satisfiability was proved	
Number of variables	121
Number of clauses	252
Sum of the clauses size	756
Maximum clause length	3
Minimum clause length	3
Number of clauses of size 1	0
Number of clauses of size 2	0
Number of clauses of size 3	252
Number of clauses of size 4	0
Number of clauses of size 5	0
Number of clauses of size over 5	0

Results of the different solvers on this benchmark

Solver Name	TraceID	Answer	CPU time	Wall clock time
SAT07 reference solver: SATzilla *CRAFTED* (complete)	1785603	? (exit code)	4998.65	5001.1
SATzilla2009_C *2009-03-22* (complete)	1825787	? (exit code)	4998.65	5000.32
VARSAT-industrial *2009-03-22* (complete)	1785604	? (TO)	5000.04	5001.91
glucose *1.0* (complete)	1784160	? (TO)	5000.04	5002.51

Why a SAT solver is extraordinary?

- It finds a needle in a Haystack!
- For n Boolean variables, the search space is 2^n
- $2^{10\,950\,109}$ correspond roughly to $10^{3\,000\,000}$
- 2^{121} correspond roughly to 10^{36}
- Number of atoms in visible universe: about 10^{80}

Any practical application?

SAT vs Software Dependency Management

• Boolean variables	packages installed or not
• Clauses (disjunction of literals)	dependencies conflicts
• Formula (conjunction of clauses)	dependency graph
• Solution (model)	Installation profile

SAT as a dependency management problem

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

```
package: a
version: 1
conflicts: a = 2

package: a
version: 2
conflicts: a = 1
```

```
package: b
version: 1
conflicts: b = 2

package: b
version: 2
conflicts: b = 1
```

```
package: c
version: 1
conflicts: c = 2

package: c
version: 2
conflicts: c = 1
```

```
package: clause
version: 1
depends: a = 2 | b = 1 | c = 1

package: clause
version: 2
depends: a = 2 | b = 2 | c = 1

package: clause
version: 3
depends: a = 1

package: clause
version: 4
depends: c = 2

package: formula
version: 1
depends: clause = 1, clause = 2,
         clause = 3, clause = 4

request: satisfiability
install: formula
```

Dependency management as a SAT problem

Dependencies can easily be translated into clauses

```
package: a
version: 1
depends: b = 2 | b = 1, c = 1
```

$$a_1 \rightarrow (b_2 \vee b_1) \wedge c_1$$

$$\neg a_1 \vee b_2 \vee b_1, \neg a_1 \vee c_1$$

Conflict can easily be translated into binary clauses

```
package: a
version: 1
conflicts: b = 2, d = 1
```

$$\neg a_1 \vee \neg b_2, \neg a_1 \vee \neg d_1$$

Both problems are NP-complete!

SAT4J, OW2 and Eclipse

SAT4J: www.sat4j.org

- Implementation in Java of Minisat specification
- Clauses, Cardinality and pseudo-Boolean constraints
- Solves SAT, MAXSAT, PBO and related problems
- Provides also end user utilities (Boolean encodings)
- Written using Java Open Source Conventions and Practices
- Distributed under the dual GNU LGPL/EPL licenses

SAT4J and OW2

- SAT4J supported since 2005 by OW2 (ObjectWeb)
  - CVS/SVN/git
  - Mailing lists/forums
  - Issue tracker
  - Releases

Trust!

- Visibility outside academia (in open source ecosystem)

- First users were academics (MIT, UTEXAS, INRIA)

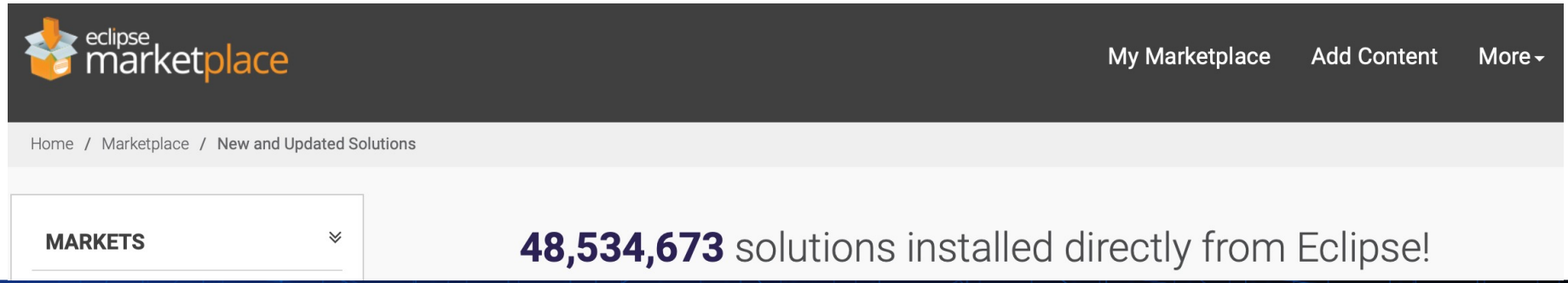
SAT4J and Eclipse: origin

- Pascal Rapicault working on p2 knew:
  - NP-completeness of dependency management
  - Early work on dependency solving using constraints solvers (EDOS, Linspire)

- Was looking for a solver to include in p2
  - Open Source
  - In Java
  - Reliable enough for inclusion in production software

SAT4J and Eclipse: integration

- 2008: Sat4j used as external component (file based)
- 2009: elected Eclipse committer to include Sat4j code inside p2 and provide explanation support and better objective function
- 2010: Eclipse marketplace opens, based on p2



Open Source Research Software Outside Academia

Why does it happen?

- Sat4j in OW2
  - First step to exist outside the lab
  - Only way for Eclipse to check for the reliability/support around the tool
- Sat4j in Eclipse
  - Sat4j built using Eclipse
  - Give back to the community
  - First answer was « no NP-complete solver in my IDE »
- Do my research around SAT but teach software engineering (using Java)

Summary

- Have a combinatorial problem? Check if a SAT solver can help
- Research software benefits from OW2 or research@Eclipse for reaching the economic market
  - Provides visibility outside the research community
  - Enforces market practices (QA, distribution, etc.)
  - Premium Open Source catalogs
- Requires the mindset to *build a product, not just a software*