# On SAT technologies

## for dependency management and beyond

Daniel Le Berre  Anne Parrain

CRIL-CNRS UMR 8188, Université d'Artois, Lens, FRANCE
{leberre,parrain}@cril.univ-artois.fr

ASPL'08, Limerick - September 12, 2008


# What can SAT solvers do for you today ?

## Many problems can be solved using a reduction into SAT

- 1996- Planning (SATPLAN,Blackbox)
- 1998- Software Specification (Nitpick, Alloy)
- 1999- Bounded Model Checking, Equivalence checking, Formal Verification, etc.
- 2005- Checking the consistency of a Linux distribution (EDOS)
- 2005- Design and debug feature models.
- 2006- Haplotyping inference (Bioinformatics)
- 2007- Cleaning up databases ?
- 2008- Resolve Eclipse or Maven dependencies ?
- 2008- Improve Linux distributions update/upgrade (Mancoosi)

# Pure SAT is not sufficient to solve many problems

## Things are getting harder

- SAT is a decision problem : answer is yes/no.
- For very constrained problems, a proof of satisfiability is usually a sufficient answer for users : SAT solvers provide them.
  - Bounded Model Checking : the proof is a bug
  - Planning : the proof is a plan
- For underconstrained problems, there are many solutions. Users look for the best solution : optimization problem.

# Binate Covering Problem

## We really need this !

- Append to the set of clauses an objective function.

$$min/max : \sum_{i=1}^{n} a_i.x_i$$

- $a_i$ integral coefficient
- boolean variables $x_i$ interpreted with truth value $\in \{0, 1\}$.
- $\overline{x_i} = 1 - x_i$.
- A note about complexity
  - SAT is NP-Complete
  - Binate Covering Problem is NP-Hard
- In practice, it is usually faster to answer SAT than UNSAT.
- Can be solved by adding new constraints of the form $\sum_{i=1}^{n} a_i.x_i < k$ with k the current best know value for the objective function (Linear search, SAT, SAT, ...SAT, UNSAT pattern)

# Preferences for managing dependencies

## Examples of Eclipse, Maven, Linux needs

- "I prefer to have solutions including the latest versions of the plugin"
- "I prefer to install the packages closer to the root"
- "I prefer to install the minimum number of packages"
- "I prefer to install the smaller set of packages in size"

In some cases, this cannot be easily mapped into a single objective function : we enter multi-objective optimization, a whole field of research in Artificial Intelligence and Operation Research

# What users want : explanations !

- If the answer is an unexpected no, most users would like to know what went wrong !
- Some SAT solvers can reduce the source of inconsistency to a subset of the clauses : UNSAT cores, Minimal Unsatisfied Subformula, ...
- Not as easy to use as the SAT solver : usually scripts on top of a solver (zCore, AMUS).
- Specific code (Minisat 1.13_p, no 2.0 release)
- Unsat core on CNF is sometimes different from unsat core in the original problem : Alloy 4 can detect which parts of the specification are inconsistent thanks to Minisat proof logging.

# What developers want : nice SAT packages !

## Are current SAT solvers real tools or research toys ?

- Full featured : unsat core, proof logging, optimization support, custom constraints support, etc.
- Supported software : tutorial, forum, mailing lists, etc.
- Easy to use : good API support, easier input language, verbose ouput, translation to CNF helpers.
- In their language

# What SAT researchers want : nice problems !

- theoretically well defined structure
- challenging benchmarks
- your problems expressed in our words :)

# Thanks for your attention


Questions ?

# The SAT4J project http://www.sat4j.org/

*"Java users deserve SAT technology too"*

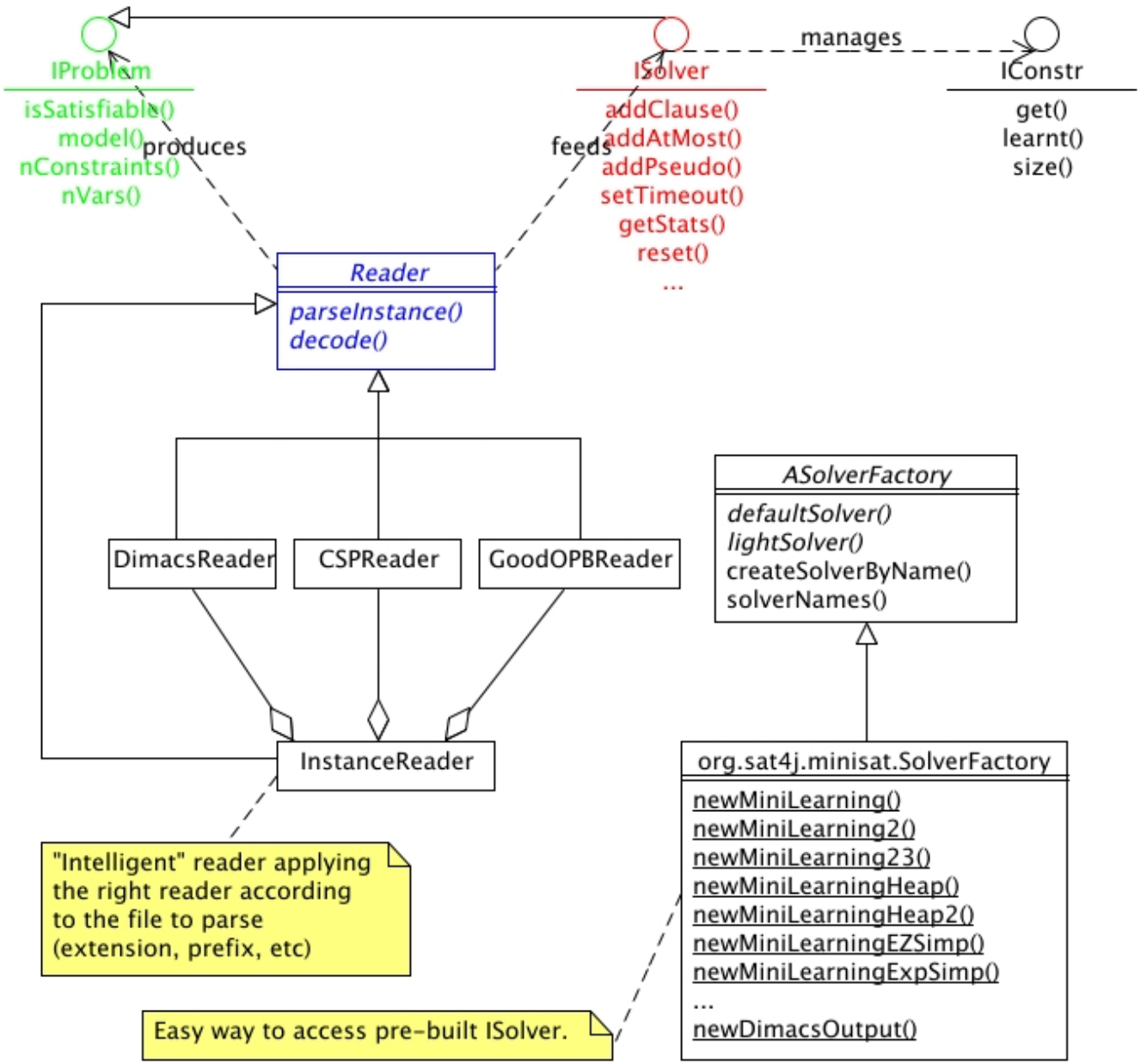
- An open source (EPL/LGPL) library of SAT solvers in Java
- Project started late 2003 as an implementation in Java of the MiniSAT specification.
- Library updated continuously with latest SAT improvements
- See it in action in the SAT competitions (2004, 2005, 2007) and the SAT Race (2006, 2008).
- Can handle several kind of constraints :

  clauses $a \vee b \vee \neg c$

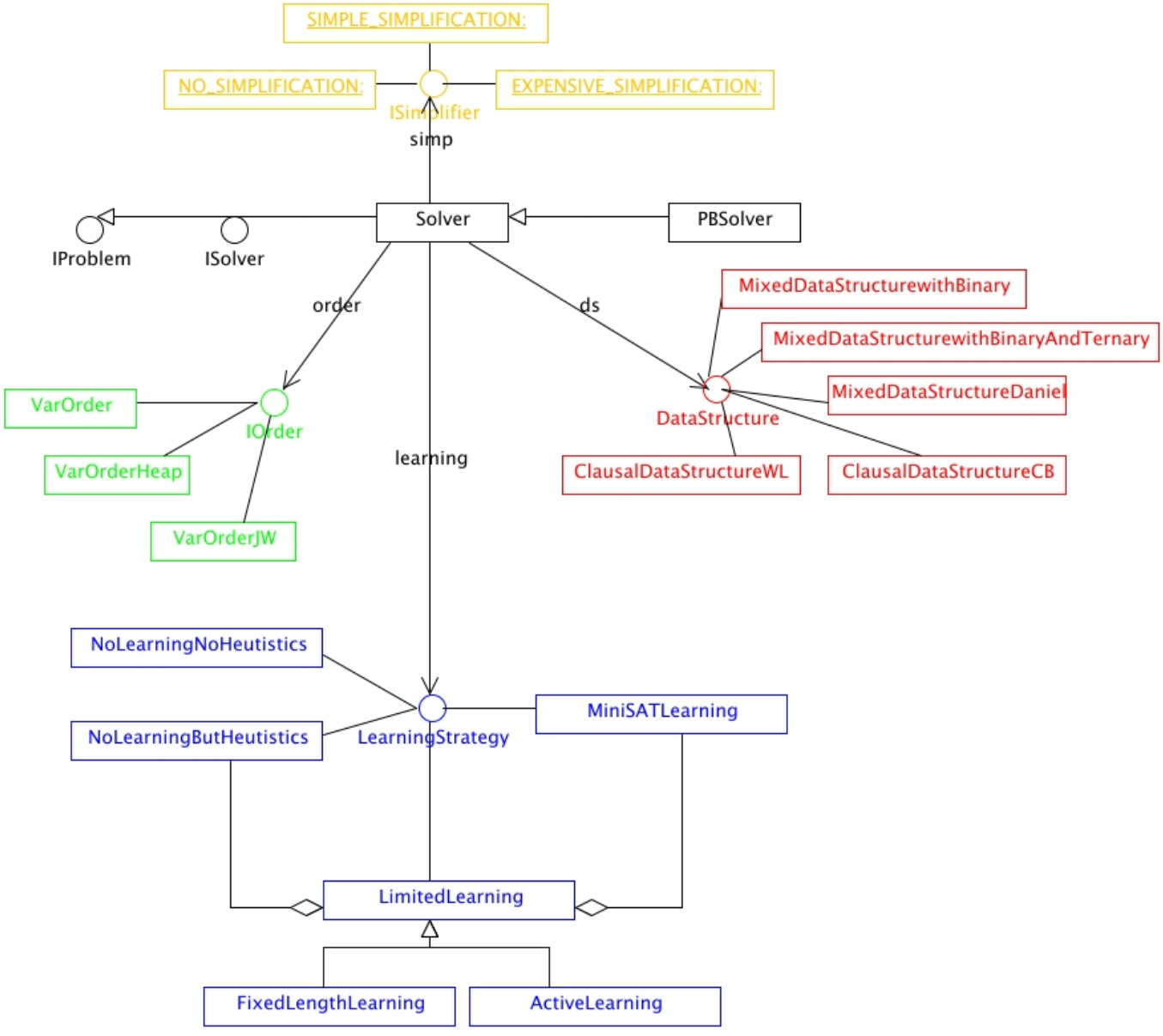
  cardinality $a + b + c + \neg d \geq 3$

  pseudo boolean $3 * a + 2 * b + 2 * c + \neg d \geq 3$
- Constraint Satisfaction Problem (CSP) to SAT support.
- Optimization problems ([Weighted] [Partial] MAXSAT).
- Target easy integration in any Java software !
- Community support provided by OW2 consortium

# SAT4J Design Overview



# SAT4J Solver Overview : a SAT solver as a SPL



# Context/Vocabulary

See `http://wiki.eclipse.org/Equinox_P2_Resolution` for details

**Artifact** Something to install (a jar file for instance).

**Installable Unit** Metadata describing an artifact, with version number, capabilities, dependencies, etc.

**Capability** feature provided by an installable unit

**Requirement** feature needed by an installable unit

**Enablement filter** an expression allowing to express conditional/platform dependent requirements.

# Equinox p2 bundle dependencies

```
IU: org.eclipse.swt v 3.2.0
 Capabilities:
     {namespace=package, name=a, version=1.0.0}
     {namespace=foo, name=b, version=1.3.0}
     {namespace=package, name=c, version=4.1.0}
 Requirement expressions
     (true) ->
     {namespace=package, name=r1, range=[1.0.0, 2.0.0)} and
     {namespace=foo, name=r1, range=[3.2.0, 4.0.0)}
     (& (os=linux) (ws=gtk)) ->
     {namespace=package, name=r2, range=[1.0.0, 2.0.0)} or
     {namespace=foo, name=bar, range=[3.2.0, 4.0.0)}
```

# Requirements encoding

- Each requirement of an installable unit $IU_i$ is represented by a simple clause

$$IU_i \rightarrow IU_{j_1} \vee IU_{j_2} \vee ... \vee IU_{j_n} \quad (1)$$

  where $IU_{j_x}$ represents all the Installable Units (IUs) that provided a capability satisfying this requirement.
- Additional Unit clauses do represent the IU that the user wants to install (or already installed ones).
- p2 keeps tracks of user installed IU (vs the ones installed to solve dependencies) to avoid uninstalling them later.

# Incompatible cases

Some installable units cannot be installed together (e.g. singleton attribute). This is modeled either with binary negative clauses

$$\neg IU_{v_i} \vee \neg IU_{v_j} \tag{2}$$

or with a cardinality constraint

$$IU_{v_1} + IU_{v_2} + ... + IU_{v_n} \leq 1 \tag{3}$$

where all the $IU_{v_i}$ do represent the Installable Units providing different versions of the same artifact for instance.

# Managing multiple IU versions or sources

If an $IU_{j_1}$ is available in multiple versions or from different sources, we abstract it from the requirements encoding :

$$IU_i \rightarrow IU_{j_1} \vee IU_{j_2} \vee ... \vee IU_{j_n} \quad (4)$$

becomes

$$IU_i \rightarrow AIU_{j_1} \vee IU_{j_2} \vee ... \vee IU_{j_n} \quad (5)$$

$$AIU_{j_1} \rightarrow IU_{j_{v1}} \vee IU_{j_{v2}} \vee ... \vee IU_{j_{vm}} \quad (6)$$

Suppose $vm$ to be the latest version, and $v1$ the oldest, we add penalties for installing older versions in the objective function : $\sum_{i=m-1}^{1} 2^{m-i}.IU_{j_{vi}}$

# Encoding optionality

- All optional requirements of an installable unit $IU_i$ are represented by the following set of clauses :

$$IU_i \rightarrow IU_{j_1} \vee IU_{j_2} \vee ... \vee IU_{j_n} \vee Noop_{IU_i}$$
$$Noop_{IU_i} \rightarrow \neg IU_{j_1}$$
$$Noop_{IU_i} \rightarrow \neg IU_{j_2}$$
$$...$$
$$Noop_{IU_i} \rightarrow \neg IU_{j_n}$$

  where $IU_{j_x}$ represents all the Installable Units (IUs) that provided a capability satisfying this optional requirement.
- In the objective function (to minimize), we add $\sum_{i=1}^{n} -1.IU_{j_i}$.
- We also apply IU abstraction if multiple versions are available.

# Encoding summary

- constraints are expressed by clauses (or cardinality constraints)
- preferences on versions and optionalities are expressed by a single objective function
- Two phase approach : only applicable requirements are encoded.
- Eclipse provisioning is an instance of the Binate Covering problem
- How to solve such problem ?
  - Pure SAT : Minisat+
  - Pseudo Boolean : Pueblo (OPIUM choice)
  - ILP : GNU LPTK
- Additional constraints for Eclipse : should better be in Java (else the solver need to be recompiled and tested on all platforms), with a license compatible with EPL.