# Using SAT for solving package dependencies

Michael Schröder
Novell, Inc.

openSUSE

Novell

# What was wrong with the old solver

Much too slow

- with bug repositories solving could take several minutes

Complex code, many special cases, still some bugs

- solver could get stuck

Bad backtracking

- recommended packages treated as required

Bad diagnostics and suggestions if unsolvable

- "libfoo is required by package barbaz"

# The SAT Problem

SAT: Boolean satisfiability problem

- find a True/False assignment to all variables of a boolean expression (AND/OR/NOT) so that it is True.
- NP complete

Normalization:

- (a | b | c) & (d | e | f) ... = TRUE
  The (...) terms are called *Rules* consisting of *literals*
- a, b, c can also be negated: -a

Example:

- (a | b | c) & (-c) & (-a | c) = TRUE
- Solution: a = FALSE, b = TRUE, c = FALSE

# Advantages of SAT

Well researched problem

- many example solvers available (chaff, minisat...)

Very fast

- package solving complexity is very low compared to other areas where SAT solvers are used

No complex algorithms

- solving just needs a couple of hundreds lines of code

Understandable suggestions

- solver calculates proof why a problem is unsolvable

# From dependencies to rules

"Requires: package" dependencies

- A requires B provided by B1, B2, B3
- Rule: (-A | B1 | B2 | B3)
  "either A is not installed or one of B1, B2, B3 is installed"

"Conflicts: package" dependencies

- A conflicts with B provided by B1, B2, B3
- 3 Rules: (-A | -B1), (-A | -B2), (-A | -B3)
  "either A is not installed or B1 is not installed"

"Obsoletes: package" dependencies

- treated as conflicts




Novell.

# Making rules (cont.)

Unary rules:

- (-A) Package A cannot be installed
  nothing provides a requirement, wrong arch, ...
  erase request (*job rule*)
- (A) Package A must be installed
  install request (*job rule*)

TRUE/FALSE values:

- TRUE: package will installed
- FALSE: package will not be installed/will be uninstalled



Novell

# Solver algorithms

## Unit propagation

- A Rule is called *unit*, if all literals but one are FALSE
- If a Rule is unit, the remaining literal can be set to TRUE
- Example: (a | b | c) & (-c) & (-a | c) = TRUE
  c is FALSE (unary rule)
  (-a | c) is unit → -a is TRUE, a is FALSE
  (a | b | c) is unit → b is TRUE

## Algorithm:

- free choice: find some undecided variable, assign TRUE or FALSE
- propagate all unit rules
- repeat until all variables are decided

# Unit propagation & dependencies

Requires rule (-A | B1 | B2 | B3)

- A, B1, B2 is FALSE → B3 must be TRUE
  "If A is installed and all but one of the providers of a requires dependency cannot be installed, the remaining one must be installed"
  → adds packages to the install set
- B1, B2, B3 is FALSE → A must be FALSE
  "If none of the provides of a required dependency can be installed, the requiring package cannot be installed"
  → adds packages to the conflicts/erase set

Conflicts rule (-A | -B1)

- A is TRUE → B1 must be FALSE and vice versa

# Contradictions

Unit propagation can lead to a contradiction

- This means that a literal must be both TRUE and FALSE
- Example (-a | b) & (-a | c) & (-b | -c)
  if solver sets a to TRUE → b, c is TRUE, c is FALSE!
- learn new rule from rules involved in contradiction
  → learned rule is (-a)
- undo last free assignment and continue solving
- if nothing to undo, problem was unsolvable

First implemented in 1996 in the GRASP solver.

# Dealing with free choices

Here is where you influence the quality of the solution:

- try to keep packages installed
- minimize number of packages to install

Algorithm

- if a package was installed before and is not in the conflicts set, install it
- if a rule is not TRUE, but all of the negative literals are FALSE, choose best of the undecided positive literals and install the corresponding package
  (-A | B1 | B2)    A TRUE → choose B1 or B2
- do not install any other package (i.e. set all undecided variables to FALSE)

# System policies

A *policy rule* defines what to do with installed packages

- must not be deinstalled or downgraded
- must not change architecture
- must not change vendor

Rule format:

- (A | A2 | A3 | A4)

A2/A3/A4 are the allowed update candidates (same name and newer version or package with matching Obsoletes: dependency)

# Reporting conflicts

If a problem turns out to be unsolvable, the solver algorithm will return a set of rules that led to the conflict

- As a system with no rpms installed is conflict free, the returned set of rules must contain at least one *job rule* or *policy rule*
- A possible solution is to remove one of those rules, i.e. remove a job (do not try to install package 'foo') or a policy rule (allow deinstallation of package 'bar')
- Advantage: users understand those rules!

# Conclusion

Using SAT solver algorithms solve many of the problems the old solver had

- speed: magnitudes faster
- reliable results
- extendibility: implementation of complex dependencies is easy
- sensible error reports

We're also working on a new repository format that can be processed much faster

- new dictionary based SOLV format

Novell®

openSUSE

Novell