Introducing the OKlibrary

Oliver Kullmann
Computer Science Department
Swansea University

SAT 2008
(Guangzhou, 15.5.2008)

An "integrated research platform" (IRE)
for (generalised) SAT

Introduction

Overview on the current state of the `OKlibrary`:

`http://www.ok-sat-library.org`

- “pre-alpha”
- an attempt to build a research environment for all aspects related to theoretical and practical research on generalised SAT
- currently focusing on a convenient

interactive environment.

Overview

An example session: “Autarky finding à la LS”

As an example, we consider the translation of the problem of autarky finding into a SAT problem according to [Mark Liffiton, Karem Sakallah, *Searching for Autarkies to Trim Unsatisfiable Clause Sets*] ([1]), using the example given by Mark in his talk at this conference.

1. First we show the example `ls_exmp_talk_guangzhou`.
2. Then we compute the translation `t_ls` via the translation function `aut_sat_ls`.
3. And we show the set of variables of the “formal clause-set” `t_ls` (a pair, consisting of the set of variables and the clause-set).

The example session: The translation

```
OKplatform> oklib --maxima
(%i1) oklib_load_all();
(%i2) ls_exmp_talk_guangzhou;
[{1,2,3,4,5},
  {{-5,-4},{-2,-1},{-2,3},{-1,2},{1},{4,5}}]
(%i3) t_ls : aut_sat_ls(
  ls_exmp_talk_guangzhou)$
(%i4) t_ls[1];
  {1,2,3,4,5,lsavv(1),lsavl(1,-1),lsavl(1,1),
  lsavv(2),lsavl(2,-1),lsavl(2,1),lsavv(3),
  lsavl(3,-1),lsavl(3,1),lsavv(4),lsavl(4,-1),
  lsavl(4,1),lsavv(5),lsavl(5,-1),lsavl(5,1),
  lsavc({-5,-4}),lsavc({-2,-1}),lsavc({-2,3}),
  lsavc({-1,2}),lsavc({1}),lsavc({4,5})}
```

Example session continued

```
(%i5) t_ls[2];
  {{-5,-lsavv(5),lsavl(5,1)},
   {-5,-lsavl(5,-1)},
   {-4,-lsavv(4),lsavl(4,1)},
   {-4,-lsavl(4,-1)},
   ...
(%i6) dispfun(lean_usat_ls);
lean_usat_ls(FF) := block(
 [T : aut_sat_ls(FF)],
  [T[1], adjoin(map(lsavv,FF[1]),T[2])]
)
(%i7) dll_simplest_trivial2(t_ls);
(%o7) true
```

Explanations

1. We showed the clauses in `t_ls`.
2. The function `lean_usat_ls` uses the ls-translation to translate the problem whether a clause-set is lean (has no non-trivial autarky) to the unsatisfiability problem.
3. Via `dispfun` the (simple) definition of the function is shown, which just adds a clause stating that one of the variables must be used in the autarky.
4. Finally we run a SAT solver on the translated clause-set, which confirms satisfiability of it (since the example `ls_exmp_talk_guangzhou` is not lean).

Example session: the test system

```
(%i8) dispfun(okltest_lean_usat_ls);
okltest_lean_usat_ls(f):=
 block([solver:dll_simplest_trivial2],
  assert(nvar_f(f(ls_exmp_talk_guangzhou)) =
    5 + 5 + 6 + 2*5),
  okltest_lean(buildq([f,solver],
    lambda([FF],not solver(f(FF))))),
  true
)
```

Explanations

1. For every function *f* we have a (generic) test function `okltest_f` which tests the functionality *f* shall provide.
2. Note that *f* is a parameter, so we have a higher-order test system, where tests are not hard-coded, and can be re-used.
3. We show only a very simple test, which uses our example, checks whether the translation has the correct number of variables, and that the translation is satisfiable.
4. As SAT solver we use `dll_simplest_trivial2` which is pure backtracking without reduction, using a branching literal the first literal in the first clause.

Example session: the clause-variable matrix

```
(%i10) M : cl_var_com_fcs(
          ls_exmp_talk_guangzhou);
(%o10) [{{-5,-4},{-2,-1},{-2,3},{-1,2},
         {1},{4,5}},{1,2,3,4,5},
         lambda([C,v],
           if elementp(v,C) then +1
         else
           (if elementp(-v,C) then -1 else 0))]
(%i15) display2d : true;
(%i16) com2m(M);
         [  0    0   0  - 1  - 1 ]
         [ - 1  - 1  0   0    0  ]
         [  0   - 1  1   0    0  ]
         [ - 1   1   0   0    0  ]
         [  1    0   0   0    0  ]
         [  0    0   0   1    1  ]
```

Explanations

1. `M` is the clause-variable matrix of the example, with the clauses constituting the rows.
2. More precisely, it is a “combinatorial matrix”, without an order on the rows or columns.
3. Using the conversion function `com2m`, which translates combinatorial matrices into Maxima matrices, we can view the matrix.

Example session: all autarkies

```
(%i6) T_t_ls : dll_simplest_st(
        t_ls,johnson_heuristic)$
(%i17) nnds_l(T_t_ls);
  30061
(%i9) dT_t_ls : condense_st(T_t_ls)$
(%i11) nnds_l(dT_t_ls);
(%o11) 465
(%i18) count_sat_st(T_t_ls,t_ls[1]);
(%o18) 72
(%i19) count_sat_st(dT_t_ls,t_ls[1]);
(%o19) 72
(%i16) 2^5 + 2^4 + 2 * 2^3 + 2 * 2^2;
(%o16) 72
```

Explanations

1. `T_t_ls` is a (complete) splitting tree for the example clause-set, using a plain backtracking algorithms (no reduction etc.) with the Johnson-heuristics (we want to minimise the size of the *whole* search tree).
2. This tree has 30061 nodes: The translation already had 26 variables, and this is the full search tree.
3. Condensing the tree, that is collapsing nodes with two satisfiable or two unsatisfiable children, we get a reasonable tree `dT_t_ls` representing the boolean function of the translated example.
4. Counting the number of satisfying assignments (of the translated clause-set) we get 72.
5. The little computation shows how this number can be computed from the autarkies of the original clause-set — details left as an exercise!

Basic ideas of the Maxima/Lisp level

- “pseudo-code which runs”
- “procedural specification”
- exploring the whole world of generalised SAT, NP and beyond
- also for teaching
- admittedly it’s not fast but “that is a feature, not a bug”.

3 levels

Three programming language layers:

1. the Maxima/Lisp level — the “set-theoretic level”
2. the Axiom/Aldor level: adding
   1. polymorphism, generic programming
   2. abstract data types
   3. also fundamental data structures
3. the C++ level: the full level, adding
   1. full control over algorithmic details (including construction and destruction of objects)
   2. distinction between compile-time and run-time.

The currently usable parts

- the Maxima/Lisp level
- ExternalSources
- the documentation system
- the test system
- source control.

An **integrated research environment** for
generalised SAT,
built together from powerful components in the Unix/Linux tradition,
combined through the `OKplatform`,
with the `OKlibrary` as core.

On the C++ level

The `OKlibrary` started as a

generative library for generalised SAT solving.

Meanwhile the scope has broadened, but the C++ level still is of great importance (for the future):

1. many plans
2. lots of code
3. but yet postponed due to **C++ 09**.

What is available

- C/C++ libraries
- various compiler
- programming support
- buildsystem components
- databases
- statistics and computer algebra
- proof assistants and automated theorem proving
- last, but not least, SAT.

Let's have a look ...
[you may visit `http://cs.swan.ac.uk/~csoliver/ok-sat-library/internet_html/doc/local_html/ExternalSources.html`]

Higher-order unit testing

Already at the Maxima/Lisp level we have seen a unit test system

which is not, as usual, left to the back office, but is integral part of the library.

Generic test functionality is offered, which can be reused.

A similar (stronger) system is supplied at the C++ level.

Integration and application testing

An application test system is under construction.

Specialised to SAT solving, this will be also applicable for "external SAT solvers":

1. specify a test level: "basic", "full" or "extensive"
2. watch whether your solver survives.

A “holistic” library

I do not mean here the quasi-religious implications of some interpretations of “holism” (“The whole is more than the sum of its parts.”), but that

the library is “complete in itself”,

or, in other words, it is **completely clonable**.

- We have already seen one example with the higher-order unit test system.
- Another aspect is that nearly all planning is “internalised” (available via the html documentation).

A strong source control system is needed. We use the most powerful open-source system, `Git` ...

Summary

I. I hope you got an impression of the parts of the library which are currently usable by the uninitiated user.

II. For the years to come, I hope a strong system can be created, spanning the whole of the “generalised SAT universe”.

End

Mark Liffiton and Karem Sakallah.
Searching for autarkies to trim unsatisfiable clause sets.
In *Theory and Applications of Satisfiability Testing - SAT 2008*, pages 182–195.