The new release, BerkMin561, is an optimized version of BerkMin56. New features of BerkMin561 are:

- better optimized code;
- better tuned heuristics;
- faster input of CNF formulas;
- possibility of choosing between several strategies of decision making and restart policies;
- possibility of tuning the program to the size of the formula

We believe that for many "real-life" instances, BerkMin561 will be the most efficient SAT-solver of the public domain. Currently, for large instances from the formal verification domain with more than 20,000-25,000 variables, we use a different SAT-solver (BerkMin62 and its descendents), which we cannot make publicly available yet. Nevertheless, BerkMin561 has been successfully applied to solving very large CNF formulas (up to a million variables).

Here is the list of available decision making and restart **strategies**.

Strategy 0 (default)

This is a general-purpose strategy. It is an improved version of the decision-making and restart policy described in our paper [DATE-2002].

Strategy 1.

This strategy is very promising for particular applications like checking the equivalence of combinational circuits. The efficiency of this strategy substantially improves if the user provides some additional information about the structure of the underlying circuit , namely the topological levels of the variables where the level of a variable *z* is the length of the longest path from a primary input of the circuit to point *z*. (By the underlying circuit we mean a so-called miter, a circuit that is built out of the two circuits to be checked for equivalence.) The format in which this information should be provided is described below. Using this strategy one can check the equivalence of multipliers (an optimized circuit versus the initial one) or even check for equivalence circuits that do not have functionally equivalent internal points at all. We are going to publish these results soon. Besides, strategy 1 has been also successfully used (even without extra information about topological levels) for solving instances of other classes (different from equivalence checking).

Strategy 2.

This strategy is a static choice of branching variables. Namely, the next variable to be assigned is the free variable having the smallest number. Sometimes, using static ordering gives very good results (The reason is, probably, that in many cases, the variable numbering used in the initial CNF formula, carries some information about the structure of the problem specified by this formula.)

Strategy 3.

This strategy may be useful for structured CNF formulas for which Strategy 1 does not work well. Strategy 3 ignores the fact that the conflict clauses are organized as a chronologically ordered stack. The most active variable is selected as the next variable to be branched on. This is exactly the choice made by BerkMin56 [DATE2002] when all the conflict clauses are satisfied and only some clauses of the original formula are left unsatisfied. (Strategy 3 makes such a choice of the branching variable even when the stack of unsatisfied conflict clauses is not empty.)

We will be very thankful for any kind of feedback with regard to the effectiveness of the strategies listed above.

Tuning to the size of the formula and its probable complexity can be done by using parameters *backtrack_limit* and *time_limit*. The program terminates as soon as it exceeds the number of backtracks specified by the parameter *backtrack_limit* or the runtime of the program exceeds the value of *time_limit* specified in seconds.

Remark 1. The parameter *backtrack_limit* is also used for determining the size of the memory the program will need for the formula to be tested. (The greater *backtrack_limit* is, the more time the program takes to allocate and initialize its data structures.) So if, on a particular instance, the program terminates reporting that it does not have enough memory, one can try to decrease the number of backtracks.

Remark 2. The way BerkMin is written now, it is possible that, for a hard instance, the program will terminate because the database size has grown very large while, in fact, the computer still has some unused memory. So there is a small probability that when using all the available computer resources the program could solve this instance. Nevertheless, in our experience, if a formula grows too large, there is little chance that it can be successfully solved. (Recall that BerkMin periodically clears up the clause database trying to get rid of very large clauses and/or clauses that are not used in conflicts any more. So typically the size of the clause database at its peak is only a few times greater than that of the initial formula.) Anyway, if you are not able to solve a formula with BerkMin561 please let us know. We are interested in collecting hard CNF formulas appearing in industrial setting. So sending us "real-life" formulas that you fail to solve with BerkMin561 is *most welcome*!! Besides, we can try to solve this formula with other SAT-solvers that are at our disposal.

Synopsis

BerkMin561 *formula_name* [b *backtrack_limit*] [s *strategy_number*] [t *time_limit*]

The parameters are shown in italic. The value of *backtrack_limit* and *time_limit* can be any positive integer while *strategy_number* $\in \{0,1,2,3\}$. The *formula_name* parameter is the file name of a CNF formula specified in the DIMACS format.

Remark. All the parameters but *formula_name* can be omitted. Besides, all the optional parameters can be put in arbitrary order (after *formula_name*).

The default values:

backtrack_limit = 100000000
strategy_number = 0
time_limit = 360000

For instance,

BerkMin561 large.cnf

BerkMin561 easy.cnf b 1000

BerkMin561 equivalence_check.cnf b 100000 s 1

BerkMin561 try_static_order.cnf b 100000 s 2

BerkMin561 your_default_strategy_stinks.cnf s 3

BerkMin561 you_got_only_one_hour_pal.cnf t 3600

Remark. For a few classes of formulas the program chooses the best strategy automatically ignoring the value of *strategy_number* specified by the user. For instance, for random 3-CNF formulas from the hard region, (that is the number of clauses is 4.25 times the number of variables) BerkMin561 always uses Strategy 3. (Don't raise your hopes though. BerkMin's performance on random CNF formulas is not very impressive.)

Extra information to be added to a CNF formula when using Strategy 1

(We repeat once again that you can use Strategy 1 even for formulas that do not contain any information about topological levels.)

The section with extra data is added at the end of the formula and starts with the keyword *cvar_levels*. This keyword is followed by the list of topological levels of the variables given according to the variable numbering of the formula. For instance, the line "cvar_levels 0 1 2 3" means that the variable 1 of the formula has level 0, the variable 2 – level 0, the variable 3 – level 1, the variable 4 – level 2, the variable 5 – level 3. The end-of-line symbol '\n' can be put anywhere after the key word *cvar_levels* but should not break a number specifying a variable's level.

The formula below describes an instance of self-verification.

```
c Checking the equivalence of C17.iscas with itself
c The outputs of the first copy are (22GAT(10) 23GAT(9))
c The outputs of the second copy are (22GAT(10)'23GAT(9)')
c Variable numbering:
c input variables : (1GAT(0) 1) (2GAT(1) 2) (3GAT(2) 3)
c (6GAT(3)4)(7GAT(4) 5)
c
c intermediate and output variables of the first copy:
c (11GAT(5) 6) (10GAT(6) 7)(19GAT(7) 10)
c (16GAT(8) 11) (23GAT(9) 14) (22GAT(10) 15)
c
c intermediate and output variables of the second copy:
c (11GAT(5)' 8) (10GAT(6)' 9) (19GAT(7)' 12)
c (16GAT(8)' 13) (23GAT(9)' 16) (22GAT(10)' 17)
c
c the variable specifying the equivalence of outputs
c 22GAT(10) and 22GAT(10)':
c (eq1 18)

c the variable specifying the equivalence of outputs
c 23GAT(9) and 23GAT(9)':
c (eq2 19)
c
p cnf 19 45
-3 -6 0
-4 -6 0
3 4 6 0
-1 -7 0
-3 -7 0
1 3 7 0
-6 -10 0
-5 -10 0
6 5 10 0
-2 -11 0
-6 -11 0
2 6 11 0
-11 -14 0
-10 -14 0
11 10 14 0
-7 -15 0
-11 -15 0
7 11 15 0
-3 -8 0
-4 -8 0
3 4 8 0
-1 -9 0
-3 -9 0
1 3 9 0
-8 -12 0
-5 -12 0
8 5 12 0
-2 -13 0
-8 -13 0
```

```
2 8 13 0
-13 -16 0
-12 -16 0
13 12 16 0
-9 -17 0
-13 -17 0
9 13 17 0
15 17 -18 0
-15 -17 -18 0
-15 17 18 0
15 -17 18 0
14 16 -19 0
-14 -16 -19 0
-14 16 19 0
14 -16 19 0
18 19 0
cvar_levels 0 0 0 0 0 1 1 1 1
2 2 2 2 3 3 3 3 4 4
```