DPLL+ROBDD Derivation Applied to Inversion of Some Cryptographic Functions

Alexey Ignatiev and Alexander Semenov

Laboratory of discrete analysis and applied logic
Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia

Fourteenth International Conference on
Theory and Applications of Satisfiability Testing

Ann Arbor, USA
June 19, 2011

Inverting discrete functions

Consider polynomial time computable functions that form a family of type

$$f_n : \{0,1\}^n \to \{0,1\}^*,$$

where $\{0,1\}^n$ is the set of all possible binary sequences of the length n, $n \in N_1$, $\{0,1\}^* = \bigcup_{n \in N_1} \{0,1\}^n$.

Inverting discrete functions

Consider polynomial time computable functions that form a family of type

$$f_n : \{0,1\}^n \to \{0,1\}^*,$$

where $\{0,1\}^n$ is the set of all possible binary sequences of the length n, $n \in N_1$, $\{0,1\}^* = \bigcup_{n \in N_1} \{0,1\}^n$.

- **Definition:**

  The problem of inverting a function f_n at point $y \in range\ f_n$ is the problem of finding such (an arbitrary) $x \in \{0,1\}^n$ that $f_n(x) = y$.

Inverting discrete functions

Consider polynomial time computable functions that form a family of type

$$f_n : \{0,1\}^n \to \{0,1\}^*,$$

where $\{0,1\}^n$ is the set of all possible binary sequences of the length n, $n \in N_1$, $\{0,1\}^* = \bigcup_{n \in N_1} \{0,1\}^n$.

- **Definition:**

  The problem of inverting a function f_n at point $y \in range\ f_n$ is the problem of finding such (an arbitrary) $x \in \{0,1\}^n$ that $f_n(x) = y$.

- Example (cryptanalysis of a keystream generator):

  $$\xrightarrow{x=(x_1,\ldots,x_n)} \boxed{f_n} \xrightarrow{y=(y_1,\ldots,y_m)}$$

  Cryptanalysis problem: given a generator algorithm and a keystream $y = (y_1, \ldots, y_m)$ one should find an initial sequence $x = (x_1, \ldots, x_n)$.

Encoding to SAT

Let f_n be an arbitrary discrete function, $S(f_n)$ be a Boolean circuit representing f_n (e. g. over $\{\&, \neg\}$).

$$\begin{array}{c}\text{circuit}\\ S(f_n)\end{array} \xrightarrow[\text{introducing auxiliary variables}]{\text{Tseitin transformations}} \begin{array}{c}\text{CNF}\\ \underset{G \in S(f_n)}{\&} C(G)\end{array}$$

Encoding to SAT

Let f_n be an arbitrary discrete function, $S(f_n)$ be a Boolean circuit representing f_n (e. g. over $\{\&, \neg\}$).

$$\begin{array}{c}\text{circuit}\\ S(f_n)\end{array} \xrightarrow[\text{introducing auxiliary variables}]{\text{Tseitin transformations}} \begin{array}{c}\text{CNF}\\ \mathop{\&}\limits_{G \in S(f_n)} C(G)\end{array}$$

Then

$$C_y(f_n) = \left(\mathop{\&}\limits_{G \in S(f_n)} C(G) \right) \cdot y_1^{\sigma_1} \cdot \ldots \cdot y_m^{\sigma_m},$$

is a CNF encoding the invertion problem of the function f_n at point $y = (\sigma_1, \ldots, \sigma_m)$. Here CNFs $C(G)$ encode gates G of $S(f_n)$ and

$$z^\sigma = \begin{cases} \bar{z}, \text{ if } \sigma = 0 \\ z, \text{ if } \sigma = 1 \end{cases}$$

A core-DPLL approach to inversion of discrete functions

- Approach: to restrict DPLL derivation to a set of variables denoting an input for $S(f_n)$.
- core-DPLL cannot polynomially simulate DPLL (even without clause learning and restarts)[1].
- The aim: to show that the use of core-DPLL provides a number of additional (or rather useful) technical capabilities while inverting discrete functions.

[1] Järvisalo, M., Junttila, T.: Limitations of restricted branching in clause learning. Constraints 14(3), 325–356 (2009).

The basic idea is...

The basic idea is...

- to represent arrays of conflict clauses accumulated by core-DPLL in the ROBDD form;

The basic idea is...

- to represent arrays of conflict clauses accumulated by core-DPLL in the ROBDD form;
- to manage hybrid DPLL+ROBDD derivation.

The basic idea is...

- to represent arrays of conflict clauses accumulated by core-DPLL in the ROBDD form;
- to manage hybrid DPLL+ROBDD derivation.

Some other attempts to combine DPLL with BDDs:

- Chatalic, P., Simon, L.: Zres: The old Davis-Putnam procedure meets ZBDDs. In: McAllester, D. (ed.) CADE 2000. LNCS(LNAI), vol. 1831, pp. 449–454. Springer, Heidelberg (2000).
- Aloul, F. A., Mneimneh, M. N., Sakallah, K. A.: ZBDD-Based Backtrack Search SAT Solver. In: Proceedings of International Workshop on Logic and Synthesis (IWLS), pp. 131–136 (2002).
- Gopalakrishnan, S., Durairaj, V., Kalla, P.: Integrating CNF and BDD based SAT solvers. In: IEEE International High-Level Design, Validation, and Test Workshop, pp. 51–56 (2003).
- Damiano, R. F., Kukula, J. H.: Checking satisfiability of a conjunction of BDDs. In: 40th Design Automation Conference, DAC 2003, pp. 818–823 (2003).
- Ganai, M., Gupta, A.: SAT-Based Scalable Formal Verification Solutions. Series on Integrated Circuits and Systems. Springer-Verlag New York, Inc., Secaucus (2007).

The basic idea is...

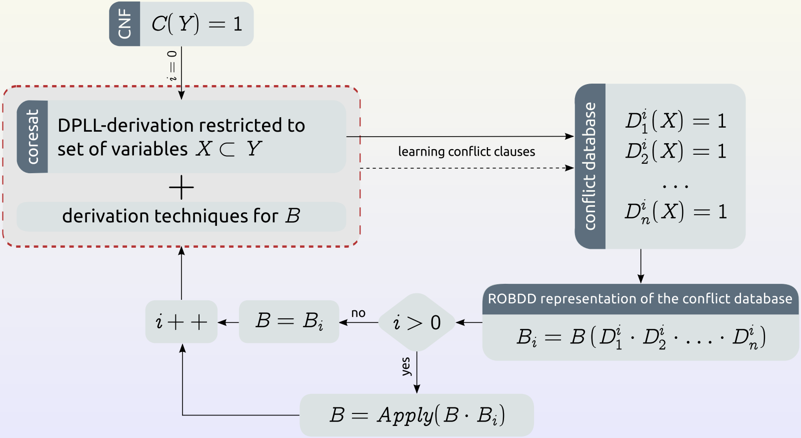
- to represent arrays of conflict clauses accumulated by core-DPLL in the ROBDD form;
- to manage hybrid DPLL+ROBDD derivation.

Some other attempts to combine DPLL with BDDs:

- Chatalic, P., Simon, L.: Zres: The old Davis-Putnam procedure meets ZBDDs. In: McAllester, D. (ed.) CADE 2000. LNCS(LNAI), vol. 1831, pp. 449–454. Springer, Heidelberg (2000).
- Aloul, F. A., Mneimneh, M. N., Sakallah, K. A.: ZBDD-Based Backtrack Search SAT Solver. In: Proceedings of International Workshop on Logic and Synthesis (IWLS), pp. 131–136 (2002).
- Gopalakrishnan, S., Durairaj, V., Kalla, P.: Integrating CNF and BDD based SAT solvers. In: IEEE International High-Level Design, Validation, and Test Workshop, pp. 51–56 (2003).
- Damiano, R. F., Kukula, J. H.: Checking satisfiability of a conjunction of BDDs. In: 40th Design Automation Conference, DAC 2003, pp. 818–823 (2003).
- Ganai, M., Gupta, A.: SAT-Based Scalable Formal Verification Solutions. Series on Integrated Circuits and Systems. Springer-Verlag New York, Inc., Secaucus (2007).

- *No examples representing conflict clauses as ROBDDs and extending basic derivation algorithms to the ROBDD structure.*

A flowchart showing operation of the hybrid solver



ROBDD-based consequence

- Let $\Delta^1(x_i)$ be a set containing truth values of x_i, $i \in \{1, \ldots, n\}$, defined by all the paths in ROBDD $B(f)$ from the root to terminal vertex “1”.

ROBDD-based consequence

- Let $\Delta^1(x_i)$ be a set containing truth values of x_i, $i \in \{1, \ldots, n\}$, defined by all the paths in ROBDD $B(f)$ from the root to terminal vertex "1".
- Suppose, that in $B(f)$ the following conditions for a variable $x_k \in X$, $X = \{x_1, \ldots, x_n\}$, hold:
  1. Every path π in $B(f)$ from the root to "1" passes through a vertex marked by x_k.
  2. $|\Delta^1(x_k)| = 1$.

ROBDD-based consequence

- Let $\Delta^1(x_i)$ be a set containing truth values of x_i, $i \in \{1, \ldots, n\}$, defined by all the paths in ROBDD $B(f)$ from the root to terminal vertex "1".
- Suppose, that in $B(f)$ the following conditions for a variable $x_k \in X$, $X = \{x_1, \ldots, x_n\}$, hold:
  1. Every path π in $B(f)$ from the root to "1" passes through a vertex marked by x_k.
  2. $|\Delta^1(x_k)| = 1$.
- Then variable x_k may take on exactly one value (the value of $\Delta^1(x_k)$) in any truth assignment over X that makes f assign true.

ROBDD-based consequence

- Let $\Delta^1(x_i)$ be a set containing truth values of x_i, $i \in \{1, \ldots, n\}$, defined by all the paths in ROBDD $B(f)$ from the root to terminal vertex "1".
- Suppose, that in $B(f)$ the following conditions for a variable $x_k \in X$, $X = \{x_1, \ldots, x_n\}$, hold:
  1. Every path π in $B(f)$ from the root to "1" passes through a vertex marked by x_k.
  2. $|\Delta^1(x_k)| = 1$.
- Then variable x_k may take on exactly one value (the value of $\Delta^1(x_k)$) in any truth assignment over X that makes f assign true.

Definition:

The situation defined by conditions 1–2 is called a ROBDD-based consequence of a value of variable x_k.

ROBDD-based consequence

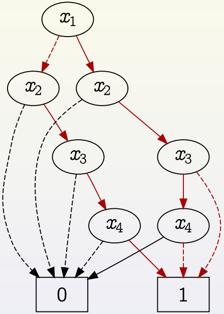


Figure: ROBDD representation of a function $x_2 \cdot (x_1 \oplus x_3 \cdot x_4)$ using the variable ordering $x_1 \prec x_2 \prec x_3 \prec x_4$. We have a ROBDD-based consequence of variable x_2 ($x_2 = 1$) here because each path from the root to "1" passes through a vertex marked by x_2 and $|\Delta^1(x_2)| = 1$.

ROBDD-based consequence

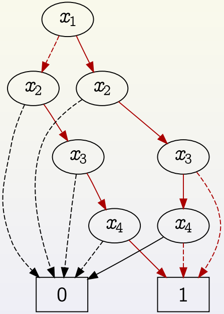


Figure: ROBDD representation of a function $x_2 \cdot (x_1 \oplus x_3 \cdot x_4)$ using the variable ordering $x_1 \prec x_2 \prec x_3 \prec x_4$. We have a ROBDD-based consequence of variable x_2 ($x_2 = 1$) here because each path from the root to "1" passes through a vertex marked by x_2 and $|\Delta^1(x_2)| = 1$.

Lemma. A ROBDD-based consequence of some variable x_k in $B(f)$ results in exactly one of the following:

- each vertex marked by x_k has "0" as the high-child;
- each vertex marked by x_k has "0" as the low-child.

ROBDD-based consequence

Theorem. For a ROBDD $B(f)$ and the values $x_{i_1} = \alpha_{i_1}, \ldots, x_{i_m} = \alpha_{i_m}$, $m \leqslant n$, $\alpha_{i_j} \in \{0, 1\}$, $j \in \{1, \ldots, m\}$, time complexity of the procedure[2] which substitutes given values into $B(f)$ and checks for ROBDD-based consequences of other variables is $O\left(|B(f)|\right)$.

[2] Similar mechanisms were described in [Damiano, R. F., Kukula, J. H.: Checking satisfiability of a conjunction of BDDs. In: 40th Design Automation Conference, DAC 2003, pp. 818–823 (2003)].

Avoiding BCP

Corollary. If some substitution into $B(f)$ implies a ROBDD-based consequence of $x_k = \alpha_k$, $\alpha_k \in \{0, 1\}$ for some $x_k \in X$, then substitution of $x_k = \alpha_k$ in $B(f)$ cannot imply another ROBDD-based consequence.

Avoiding BCP

Corollary. If some substitution into $B(f)$ implies a ROBDD-based consequence of $x_k = \alpha_k$, $\alpha_k \in \{0, 1\}$ for some $x_k \in X$, then substitution of $x_k = \alpha_k$ in $B(f)$ cannot imply another ROBDD-based consequence.

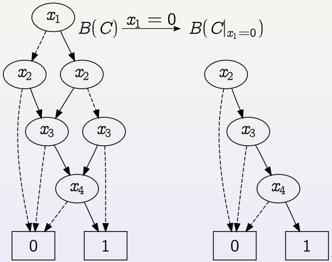
$$C(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \cdot (\overline{x}_2 \vee x_3) \cdot (\overline{x}_3 \vee x_4)$$

$$\xrightarrow{x_1 = 0} \quad C|_{x_1=0} = x_2 \cdot (\overline{x}_2 \vee x_3) \cdot (\overline{x}_3 \vee x_4) \Longrightarrow x_2 = 1$$

$$\xrightarrow{x_2 = 1} \quad C|_{x_1=0,x_2=1} = x_3 \cdot (\overline{x}_3 \vee x_4) \Longrightarrow x_3 = 1$$

$$\xrightarrow{x_3 = 1} \quad C|_{x_1=0,x_2=1,x_3=1} = x_4 \Longrightarrow x_4 = 1$$

(a) BCP



(b) ROBDD-case

Figure: The left part shows the BCP process in CNF $(x_1 \vee x_2) \cdot (\overline{x}_2 \vee x_3) \cdot (\overline{x}_3 \vee x_4)$ started by assigning $x_1 = 0$; the right part demonstrates the result of substituting $x_1 = 0$ into ROBDD representation of the considered CNF — here a single pass through the ROBDD is required.

Lazy computations

Consider the following conditions:

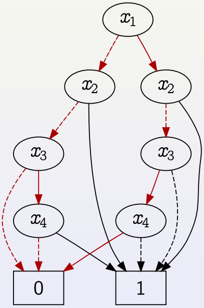
3. For a variable $x_q \in X$, $X = \{x_1, \ldots, x_n\}$, every path π in $B(f)$ from the root to “0” passes through a vertex marked by x_q.
4. $|\Delta^0(x_q)| = 1$.

Lazy computations

Consider the following conditions:

3. For a variable $x_q \in X$, $X = \{x_1, \ldots, x_n\}$, every path π in $B(f)$ from the root to "0" passes through a vertex marked by x_q.
4. $|\Delta^0(x_q)| = 1$.

Figure: ROBDD representation of a function $x_2 \vee (x_1 \oplus x_3 \cdot x_4)$ using the variable ordering $x_1 \prec x_2 \prec x_3 \prec x_4$. Conditions 3–4 hold for x_2 because each path from the root to "0" passes through a vertex marked by x_2 and $|\Delta^0(x_2)| = 1$.

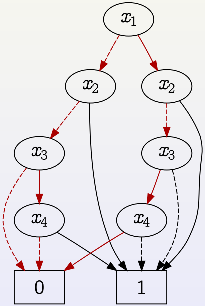


Lazy computations

Consider the following conditions:

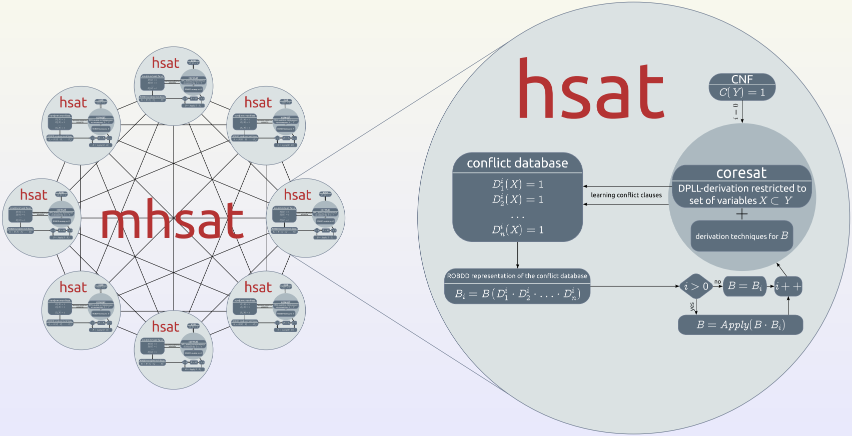
3. For a variable $x_q \in X$, $X = \{x_1, \ldots, x_n\}$, every path π in $B(f)$ from the root to "0" passes through a vertex marked by x_q.
4. $|\Delta^0(x_q)| = 1$.

Figure: ROBDD representation of a function $x_2 \vee (x_1 \oplus x_3 \cdot x_4)$ using the variable ordering $x_1 \prec x_2 \prec x_3 \prec x_4$. Conditions 3–4 hold for x_2 because each path from the root to "0" passes through a vertex marked by x_2 and $|\Delta^0(x_2)| = 1$.

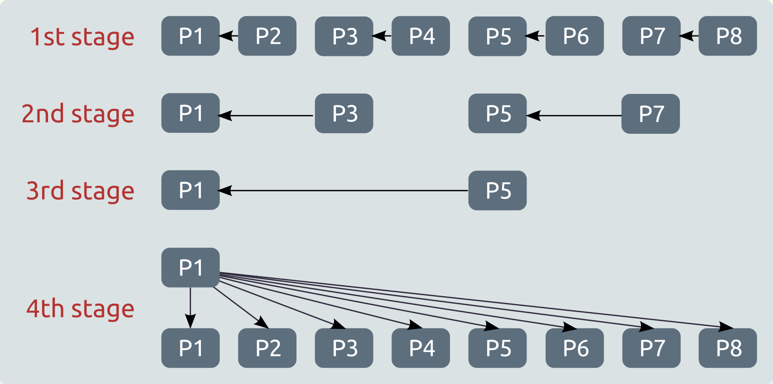


Theorem. Let $B(f)$ be an arbitrary ROBDD and there be such a variable x_q in $B(f)$ so that conditions 3–4 hold for x_q. Then there are no possible ROBDD-based consequences of any variable from $X \setminus \{x_q\}$ in $B(f)$. Time complexity of procedure which checks whether conditions 3–4 hold is $O(|B(f)|)$.

Parallel version of the hybrid solver (mhsat)



Simple exchange of conflict clauses



Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of Collective Communication Operations in MPICH. Int'l Journal of High Performance Computing Applications 19(1), 49–66 (2005).

From MiniSat to mhsat

- MiniSat-C v1.14.1 (http://minisat.se/MiniSat.html)

[3] See [Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC]].

[4] The core-DPLL implementation is based on the use of characteristic vectors similar to the technique proposed in [Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Transactions on Computer-Aided Design 21(12), 1377–1394 (2002)].

From MiniSat to mhsat

- MiniSat-C v1.14.1 (`http://minisat.se/MiniSat.html`)
- dminisat[3]

[3]See [Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC]].

[4]The core-DPLL implementation is based on the use of characteristic vectors similar to the technique proposed in [Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Transactions on Computer-Aided Design 21(12), 1377–1394 (2002)].

From MiniSat to mhsat

- MiniSat-C v1.14.1 (`http://minisat.se/MiniSat.html`)
- dminisat[3]
- coresat[4]

[3] See [Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC]].

[4] The core-DPLL implementation is based on the use of characteristic vectors similar to the technique proposed in [Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Transactions on Computer-Aided Design 21(12), 1377–1394 (2002)].

From MiniSat to mhsat

- MiniSat-C v1.14.1 (`http://minisat.se/MiniSat.html`)
- dminisat[3]
- coresat[4]
- hsat (hybrid sat) = coresat + robdd-related procedures

[3] See [Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC]].

[4] The core-DPLL implementation is based on the use of characteristic vectors similar to the technique proposed in [Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Transactions on Computer-Aided Design 21(12), 1377–1394 (2002)].

From MiniSat to mhsat

- MiniSat-C v1.14.1 (`http://minisat.se/MiniSat.html`)
- dminisat[3]
- coresat[4]
- hsat (hybrid sat) = coresat + robdd-related procedures
- mhsat = n·hsat + sharing accumulated conflict clauses through the MPI interface.

---

[3] See [Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC]].

[4] The core-DPLL implementation is based on the use of characteristic vectors similar to the technique proposed in [Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Transactions on Computer-Aided Design 21(12), 1377–1394 (2002)].

A5/1 description

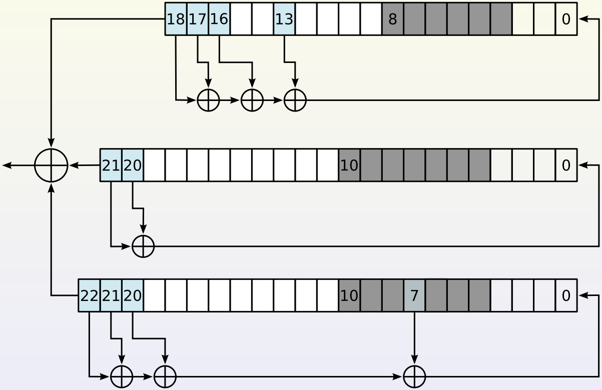


Figure: A5/1 keystream generator. Decomposition set X_{20} of 20 variables

The length of initial sequence is 64 bits.

LFSR 1: $X^{19} + X^{18} + X^{17} + X^{14} + 1$
LFSR 2: $X^{22} + X^{21} + 1$
LFSR 3: $X^{23} + X^{22} + X^{8} + 1$

b_s^1, b_s^2, b_s^3 — values of the *clocking bits*

$$\chi_m(b_s^1, b_s^2, b_s^3) = \begin{cases} 1, b_s^m = majority(b_s^1, b_s^2, b_s^3) \\ 0, b_s^m \neq majority(b_s^1, b_s^2, b_s^3) \end{cases}$$

$$majority(x, y, z) = x \cdot y \vee x \cdot z \vee y \cdot z$$

Experiment description

- Test material: 50 CNFs chosen randomly from the decomposition family generated by decomposition set X_{20}.

[5] For each of 50 CNFs there constructed 4 simpler CNFs obtained by substituting all the possible values of two variables x_{23} and x_{45} into the original one.

Experiment description

- Test material: 50 CNFs chosen randomly from the decomposition family generated by decomposition set X_{20}.
- Approaches:
  - Coarse-grained parallelization without sharing clauses (hsat, dminisat and MiniSat 2.2.0)[5].
  - Solvers with parallel architecture (MiraXT 1.1, ManySAT 1.1 and mhsat).
  - Sequential solving (hsat).

[5] For each of 50 CNFs there constructed 4 simpler CNFs obtained by substituting all the possible values of two variables x_{23} and x_{45} into the original one.

Experiment description

- Test material: 50 CNFs chosen randomly from the decomposition family generated by decomposition set X_{20}.
- Approaches:
  - Coarse-grained parallelization without sharing clauses (hsat, dminisat and MiniSat 2.2.0)[5].
  - Solvers with parallel architecture (MiraXT 1.1, ManySAT 1.1 and mhsat).
  - Sequential solving (hsat).
- Experimental platform: Intel Xeon Processor E5345 (4 cores, 2.33 GHz), 8GB RAM.

[5] For each of 50 CNFs there constructed 4 simpler CNFs obtained by substituting all the possible values of two variables x_{23} and x_{45} into the original one.

Experimental results

Table: Average solving time for each of the solvers.

place	solver	mode of operating	number of cores	avg. time (seconds)
1	mhsat[6]	parallel	4	569.016
2	hsat	coarse-grained	4	644.254
3	MiraXT (mod)	parallel	4	1639.192
4	hsat	sequential	1	2385.578
5	dminisat	coarse-grained	4	2750.486
6	MiraXT (orig)	parallel	4	3214.178
7	ManySAT (mod)[7]	parallel	4	3378.078
8	MiniSat (mod)	coarse-grained	4	5836.782

[6] mhsat taking 4 CPU cores is more than 4 times faster than its sequential version (hsat).

[7] Original versions of ManySAT and MiniSat cannot cope with the tests. However, one can improve them by assigning nonzero values to initial activity for those variables which correspond to initial contents of A5/1's registers.

Conclusions and future work

- The work (including software implementation) is not completed yet.

Conclusions and future work

- The work (including software implementation) is not completed yet.
- Too few experiments.

Conclusions and future work

- The work (including software implementation) is not completed yet.
- Too few experiments.

- The core-DPLL+ROBDD derivation may be useful in inverting some discrete functions.

Conclusions and future work

- The work (including software implementation) is not completed yet.
- Too few experiments.

- The core-DPLL+ROBDD derivation may be useful in inverting some discrete functions.
- Improvements of the approach are expected (both sequential and parallel versions).

Thank you for your attention!