PBLib { A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF

Peter Steinke
peter.steinke@tu-dresden.de

February 23, 2015

Abstract

Many different encodings for PB constraints into conjunctive normal form (CNF) have been proposed in the past. The PBLib project starts to collect and implement these encodings to be able to encode PB constraints in a very simple, but effective way. The aim is not only to generate as few clauses as possible, but also using encodings that maintain generalized arc consistency by unit propagation, to speedup the run time of the SAT solver, solving the formula.

A major issue of the implementation is a high flexibility for the user. Consequently it is not required to bring a PB constraint into a certain normal form. The PBLib automatically normalizes the constraints and decides which encoder provides the most effective translation.

The user can also define constraints with two comparators (less equal and greater equal) and each PB constraint can be encoded in an incremental way: After an initial encoding it is possible to add a tighter bound with only a few additional clauses. This mechanism allows the user to develop SAT-based solvers for optimization problems with incremental strengthening and to keep the learned clauses for incremental SAT solver calls.

1 Implemented Encodings

Table 1 shows the encodings are currently implemented in the PBLib. The label *todo* denotes encodings that are planned for the (near) future.

2 Overview

The overview in Figure 1 shows the sequence of encoding a PB constraint. A PB constraint is given to the PB2CNF class, where the constraint is simpli ed and normalized by the PreEncoder. Next PB2CNF selects a suitable encoder for the simpli ed constraint. The clauses generated by the encoder are added to a clauses database and auxiliary variables are provided by an instance of the AuxVarManager class.

Figure 2 shows the encoding process of an incremental constraint. After this initial encoding the user can encode a tighter bound with the incremental data (without calling PB2CNF again), using the information that the initial bound is already in the clause database.

At most one	At most K	PB
sequential[11]*	BDD[7, 4]**	BDD
bimander[6]	cardinality networks[1]	adder networks
commander[8]	adder networks[4]	watchdog[10]
k-product[3]	*todo: perfect hashing*[12]	sorting networks[4]
binary[2]		binary merge[9]
pairwaise		sequential weight counter[5]
nested		

* equivalent to BDD, latter and regular encoding
** equivalent to sequential counter

Table 1: Implemented encodings

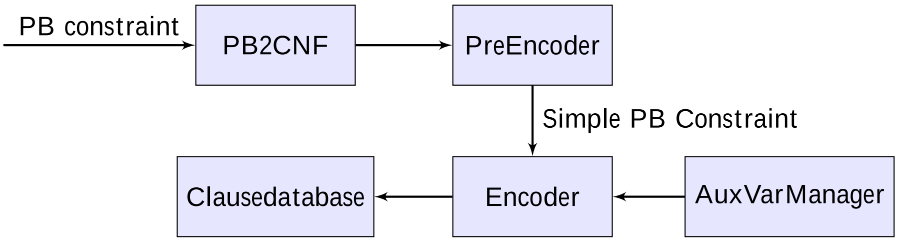

Figure 1: Encoding a PB constraint to CNF

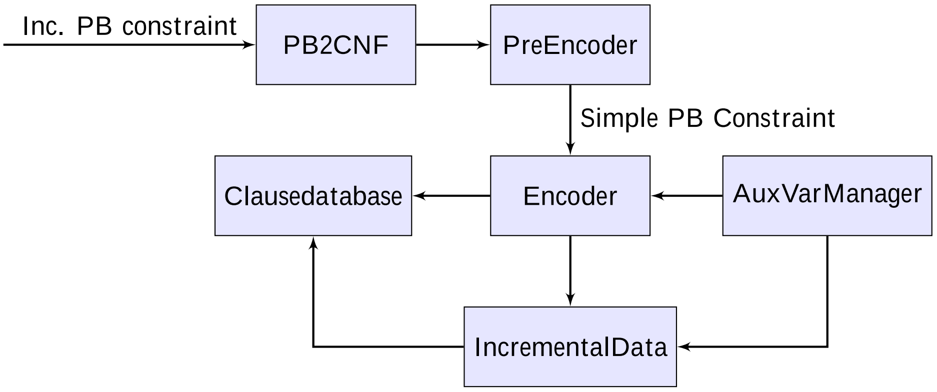

Figure 2: Encoding an incremental PB constraint to CNF

3 Using the PBLib

3.1 Including the PBLib

The following includes are needed for the PBLib:

```
#include "pb2cnf.h"       // encoding interface
```

Optional includes are:

```
#include "VectorClauseDatabase.h" // basic clause database
#include "PBParser.h"     // parser for opb files
#include "SATSolverClauseDatabase.h"
// a clause database that adds all clauses
// directly to a SAT solver (minisat)
```

The project has to be linked against the PBLib (libpblib.a or libpblib.so). Note that PBLib uses the C++11 standard .

3.2 PB Constraints

For an instance of a PB constraint a vector of weighted literals is used:

WeightedLit (int32_t literal, int64_t weight);

In addition to the weighed literals a comparator is needed. This could be less equal, greater equal or a combination of both.

enum Comparator *f* LEQ, GEQ, BOTH *g*;

Depending on the comparator one or two bounds are needed:

PBConstraint (vector< WeightedLit > const & literals, Comparator comparator, int64_t bound);

PBConstraint (vector< WeightedLit > const & literals, Comparator comparator, int64_t less_eq, int64_t greater_eq);
```
// in case comparator == BOTH and less\_eq != greater\_eq
```

3.3 Incremental PB Constraints

In addition to the PB constraints presented above, the incremental version allows an incremental encoding of a sequence of tighter bounds. Note that an incremental PB constraint is not a subclass of PB constraint (in the sense of object-oriented programming).

After the initial encoding of an incremental PB constraint *incPbConstraint*, it is possible to encode new (tighter) bounds with the methods:

incPbConstraint.encodeNewGeq (int64_t newGeq, ClauseDatabase & formula, AuxVarManager & auxVars);

incPbConstraint.encodeNewLeq (int64_t newLeq, ClauseDatabase & formula, AuxVarManager & auxVars);

The clause database *formula* has to contain the initial encoding of *incPbConstraint*.

3.7 Configuration

An instance of the *PBConfig* class contains the configuration for all options in the PBLib. Since PBConfig is a shared pointer you have to initialize it in the following way:

```
PBConfig config = make_shared<PBConfigClass>();
```

The following options (with the given default values) are currently available:

```
PB2CNF_PB_Encoder pb_encoder = PB_ENCODER::BEST;
PB2CNF_AMK_Encoder amk_encoder = AMK_ENCODER::BEST;
PB2CNF_AMO_Encoder amo_encoder = AMO_ENCODER::BEST;
BIMANDER_M_IS bimander_m_is = BIMANDER_M_IS::N_HALF;
int bimander_m = 3;
int k_product_minimum_lit_count_for_splitting = 10;
int k_product_k = 2;
int commander_encoding_k = 3;
int64_t MAX_CLAUSES_PER_CONSTRAINT = 1000000;
bool use_formula_cache = false;
bool print_used_encodings = false;
bool check_for_dup_literals = false;
bool use_real_robdds = true;
bool use_watch_dog_encoding_in_binary_merger = false;

enum PB2CNF_AMO_Encoder {BEST, NESTED, BDD,
                         BIMANDER, COMMANDER,
                         KPRODUCT, BINARY, PAIRWISE};
enum PB2CNF_AMK_Encoder {BEST, BDD, CARD};
enum PB2CNF_PB_Encoder  {BEST, BDD, SWC, SORTINGNETWORKS,
 ADDER, BINARY_MERGE};
enum BIMANDER_M_IS      {N_HALF, N_SQRT, FIXED};
```

Note that if the maximum number of clauses per constraint is (approximately) exceeded, the adder encoding is used as a fallback.

4 PB encoder

The PBLib also includes a PBEncoder which takes an opb input file and translate it into CNF using the PBLib:

```
 usage ./pbencoder inputfile
```

5 Example PB Solver

The PBLib source code contains also a folder *BasicPBSolver* with the implementations of an example PB Solver. It uses minisat 2.2 as a back-end SAT solver.

```
usage ./pbsolver inputfile
```

6 PB Fuzzer

Included is also a PB fuzzer. The program generates a random PB constraint that is solves it with different configuration within the PBLib.

This program helps to find bugs in new or customized implementations of the PBLib.

```
usage: ./fuzzer
```

References

[1] Ignasi Abio, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodriguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 80–96. Springer Berlin Heidelberg, 2013.

[2] Anthony J. Doggett Alan M. Frisch, Timothy J. Peugniez and Peter W. Nightingale. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings. *Journal of Automated Reasoning*, 35, 2005.

[3] Jingchao Chen. A new sat encoding of the at-most-one constraint. In *The Twelfth International Workshop on Constraint Modelling and Reformulation*, 2010.

[4] Niklas Een and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2, 2006.

[5] Steffen Hölldobler, Norbert Manthey, and Peter Steinke. A Compact Encoding of Pseudo-Boolean Constraints into SAT. Technical report, 2012.

[6] Steffen Hölldobler and Van-Hau Nguyen. On SAT-Encodings of the At-Most-One Constraint. In *The Twelfth International Workshop on Constraint Modelling and Reformulation*, 2013.

[7] Albert Oliveras Ignasi Abio, Robert Nieuwenhuis and Enric Rodriguez-Carbonell. BDDs for Pseudo-Boolean Constraints - Revisited. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011.

[8] Will Klieber and Gihwon Kwon. Efficient CNF Encoding for Selecting 1 from N Objects. In *The fourth Workshop on Constraints in Formal Verification*, 2007.

[9] Norbert Manthey, Tobias Philipp, and Peter Steinke. A more compact translation of pseudo-Boolean constraints into CNF such that generalized arc consistency is maintained. In Carsten Lutz and Michael Tielscher, editors, *KI 2014: Advances in Artificial Intelligence*, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer Berlin Heidelberg, 2014.

[10] Yacine Boufkhad Olivier Bailleux and Olivier Roussel. New Encodings of Pseudo-Boolean Constraints into CNF. In Oliver Kullmann, editor, *Theory and Applications of Satis ability Testing - SAT 2009*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

[11] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming { CP 2005*, 2005.

[12] Oded Margalit Yael Ben-Haim, Alexander Ivrii and Arie Matsliah. Perfect Hashing and CNF Encodings of Cardinality Constraints. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satis- ability Testing - SAT 2012*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.