CoPAn User-Guide

Stephan Kottler, Christian Zielke, Paul Seitz, Michael Kaufmann

University of Tübingen, Germany

1 Working with CoPAn

CoPAn is a Java tool to analyse CDCL SAT solvers. It allows the user for an in-depth analysis of conflicts and the process of creating learnt clauses. Particularly we focus on isomorphic patterns within the resolution operation for different conflicts. Common proof logging output of any CDCL solver can be adapted to configure the analysis of CoPAn in multiple ways.

1.1 The Input

CoPAn requires a special input, containing the following files for a solver run of one instance:

- A *.clses file containing all clauses of the original formula in addition to all learnt clauses. This file may also contain information about the (possibly) deleted learnt clauses. Additional key-value pairs can be used to add user-defined properties to clauses as shown in Figure 1.
- A *.confl file containing all learnt conflicts with the clauses that cause the specific reasoning. An example is given in Figure 2.

Both files can be produced by common CDCL solvers with only small changes in the source code and should be named identically (excluding the file ending).

```
4: -84 112 116 b 1 l 1
7: -45 108 190 b 1
3: -54 -131 158 l 2 b 7
1: -118 -165 -175 196
GC<7> 4 3
```

Figure 1. Example for a *.clses input file.

A line within the file (Figure 1) either states a single clause or a run of the garbage collector that deleted a subset of clauses. If the line represents a clause, it starts with an integer as identifier, followed by the list of literals defining the clause. After the last literal of the clause, additional properties of the clause can

be added using a pair of a single character and a numerical value. The single character is the identifier (key) of the additional value and has to be defined in the .config file used in CoPAn.
If the line represents a run of the garbage collector it starts with 'GC', followed by the ID of the last clause (written in angle brackets) that was learnt before the call to the garbage collector. The remaining line lists all deleted clauses, identified by their IDs (separated by space characters).
In this example the clause with identifier '3' contains three literals, caused a backjump by seven levels (b 7) and has an LBD value of 2 (l 2). The last line of the shown excerpt of a *.clses file denotes that after clause '7' has been learnt, the garbage collector deleted the clauses '4' and '3' from the data base.

```
90 553 792 176 858 :859
545 710 :860
145 219 :861
700 295 861 495 403 155 :862
```

Figure 2. Example for a *.confl input file.

Each line within the file (Figure 2) represents one conflict. It starts with the list of clause identifiers (as specified in the .clses file). This list contains all clauses that cause a conflict via a sequence of resolutions. The integer after the colon states the ID of the clause which is learnt by this conflict.
In this example the clause with ID '861' is learnt due to a resolution of clauses '145' and '219'.

If a user chooses to log and analyze additional properties for clauses the properties have to be defined in a *.config file for CoPAn. An example for such a file is shown in Figure 3. The purpose of an extra *.config file is to keep log files as small as possible by simultaneously having understandable property names for the GUI.

```
;this is an example for a .config file
b 0 backjump
l 0 LBD
```

Figure 3. Example for a *.config file.

A line within the file (Figure 3) represents a key, that can be added for any defined clause in the *.clses file, as anticipated in Figure 1. Any line starts with a single character that identifies the key, followed by the default value and an understandable name for the property. All three information are separated by a single space character, therefore the name is prohibited to contain any other space characters. The default value is used for every clause, that does not specify a value for this particular key.

1.2 How to install CoPAn

After downloading CoPAn from `http://algo.inf.uni-tuebingen.de/?site=forschung/sat/CoPAn`, the package has to be extracted to any directory.

1.3 Preprocessing the input

Before CoPAn can be used to analyse resolution patterns, a preprocessing of the input files (generated by preceding solver runs) is required to create the external data structures. The bulk of computation is done at this stage to enable quick computations when working with the main tool.
The preprocessing of two files (`inputfilename.clses` and `inputfilename.confl`) can be started by using the command

```
java -jar CoPAn-preprocess.jar inputfilename.confl
```

It is also possible to process all pairs of `*.clses` and `*.confl` files within a directory `inputdir` at once by the command:

```
java -jar CoPAn-preprocess.jar inputdir
```

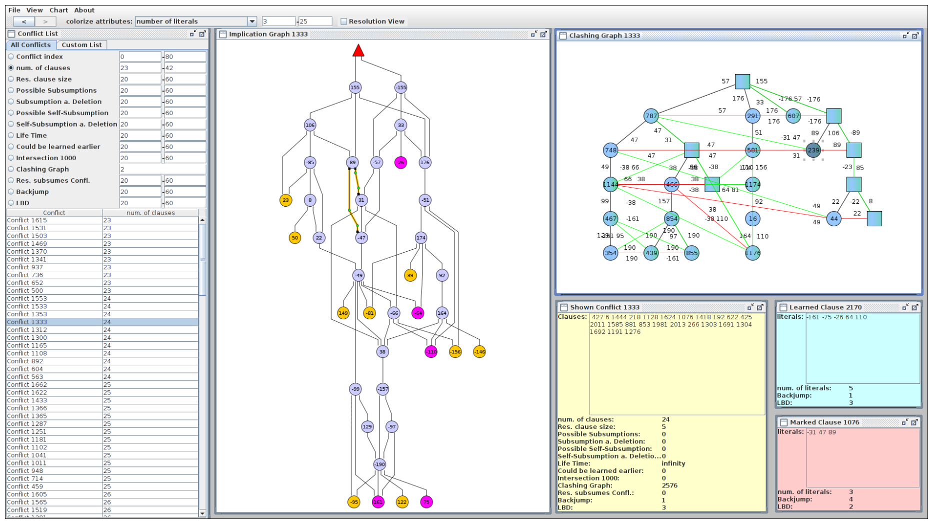
By default the resulting files, an `inputfilename.db` and an `inputfilename.lg` will be created in the same directory as the input files.

1.4 Using the GUI tool

With the following command the GUI application of CoPAn can be started:

```
java -jar CoPAn-GUI.jar
```

After having started the program a user may open any computed `*.db` file (File - Open). All information about one solver run for the chosen instance is kept in this preprocessed file. The CoPAn GUI only allows for a single `*.db` file to be loaded at any time.
Once a `*.db` file is loaded a user may explore the whole set of conflicts that occurred during the execution of the SAT solver for this instance (Figure 4). The left hand side of the main frame shows the **Conflict List** where all conflicts matching a particular filter are listed. Right after loading a `*.db` file no filter is active. By activating one filter and setting the corresponding ranges to a minimum and maximum value the query hits can be tailored to one's needs.
Here is a brief explanation of the filters and their meanings:



colorize attributes: number of literals

Conflict List

All Conflicts | Custom List

○ Conflict index	0	-80
● num. of clauses	23	-42
○ Res. clause size	20	-60
○ Possible Subsumptions	20	-60
○ Subsumption a. Deletion	20	-60
○ Possible Self-Subsumption	20	-60
○ Self-Subsumption a. Deletion	20	-60
○ Life Time	20	-60
○ Could be learned earlier	20	-60
○ Intersection 1000	20	-60
○ Clashing Graph	2	
○ Res. subsumes Confl.	20	-60
○ Backjump	20	-60
○ LBD	20	-60

Conflict	num. of clauses
Conflict 1615	23
Conflict 1531	23
Conflict 1503	23
Conflict 1469	23
Conflict 1370	23
Conflict 1341	23
Conflict 937	23
Conflict 736	23
Conflict 652	23
Conflict 500	23
Conflict 1553	24
Conflict 1533	24
Conflict 1353	24
Conflict 1333	24
Conflict 1312	24
Conflict 1300	24
Conflict 1165	24
Conflict 1108	24
Conflict 892	24
Conflict 604	24
Conflict 563	24
Conflict 1662	25
Conflict 1622	25
Conflict 1433	25
Conflict 1366	25
Conflict 1365	25
Conflict 1287	25
Conflict 1251	25
Conflict 1181	25
Conflict 1102	25
Conflict 1041	25
Conflict 1011	25
Conflict 948	25
Conflict 714	25
Conflict 459	25
Conflict 1605	26
Conflict 1565	26
Conflict 1519	26

Shown Conflict 1333

Clauses: 427 6 1444 218 1128 1624 1076 1418 192 622 425 2011 1585 881 853 1981 2013 266 1303 1691 1304 1692 1191 1276

num. of clauses:	24
Res. clause size:	5
Possible Subsumptions:	0
Subsumption a. Deletion:	0
Possible Self-Subsumption:	0
Self-Subsumption a. Deletio...	0
Life Time:	infinity
Could be learned earlier:	0
Intersection 1000:	0
Clashing Graph:	2576
Res. subsumes Confl.:	0
Backjump:	1
LBD:	3

Learned Clause 2170

literals: -161 -75 -26 64 110

num. of literals:	5
Backjump:	1
LBD:	3

Marked Clause 1076

literals: -31 47 89

num. of literals:	3
Backjump:	4
LBD:	2

Figure 4. Main view of the CoPAn GUI. A conflict with 24 clauses is selected.

- **Conflict index** Conflicts in the `*.confl` file were indexed according to their appearance in the file.
- **Num. of clauses** Number of clauses used for resolution to create the learnt clause.
- **Res. clause size** Number of literals in the learnt clause.
- **Possible Subsumptions** Number of possible subsumptions of all clauses (learnt and original ones) done by the learnt clause.
- **Subsumption a. Deletion** Number of possible subsumptions that could have been done by the learnt clause after it was deleted by the garbage collector. Therefore only learnt clauses that were added to the formula after the deletion time are candidates for a subsumption.
- **Possible Self-Subsumption** Number of possible self-subsumptions of all clauses (learnt and original ones) done by the learnt clause.
- **Self-Subsumptions a. Deletion** Analogous to **Subsumption a. Deletion**
- **Life Time** The life time of a learnt clause C is defined by the amount of learnt clauses that were learnt in between creation and destruction of C.
- **Could be learnt earlier** If a learnt clause C_i subsumes a clause C_j that was learnt earlier, this filter checks whether or not the conflict of C_i contains clauses that were learnt in between the creation of of C_i and C_j. If the conflict of C_i uses no new information, the solver executed a bad behavior, since it could have learnt a better clause than C_j (because C_i subsumes C_j) at the same time. Values are 0 or 1.

– **Intersection 1000** Counts the number of learnt clauses C_i that occur 1000 conflicts before and behind the current one C_x iff both learnt clauses C_i and C_x have at least $0.8 *$ size of C_i literals in common.
– **Clashing Graph** Shows all conflicts with an isomorphic clashing graph. The clashing graph identifier is set by the isomorphism algorithm.
– **Res. subsumes Confl.** Checks whether the learnt clause subsumes the conflicting clause which caused the conflict in the solver run (1 or 0).
– **Backjump / LBD** User-defined filters as configured in the `*.db` file.

The Conflict List can be used to focus the analysis on a subset of clauses. It also provides the possibility to copy the filtered conflicts (or a subset of selected conflicts) to the so-called **Custom List**. Conflicts in the **Custom List** can further be limited by sequences of filter applications that can be invoked via a pop-up menu that appears after a right click within the **Custom List**.
Activating one conflict from the **Custom List** or **Conflict List** causes the visualization of the corresponding *clashing graph*, *implication graph* or both, depending on the settings in View - Layout.
Three more panels in the main frame show statistical information about the selected conflict, its learnt clause, and the currently marked clause in the *clashing graph*. The values should be self-explanatory.

1.5 Working without a GUI

The batch version of CoPAn ('CoPAn-analyser') may be more convenient if the same analysis has to me made for several files or if queries are permanently adapted within a longer in-depth analysis. Any sequence of filters on a particular set of precomputed `*.db` files can be processed by the analyser tool. Since the bulk of computation is done during preprocessing queries are handled quickly. The analyser tool handles the following parameters at invocation:

```
java -jar CoPAn-analyser.jar <path> <regex> <howmany> <attribute
                        constraints...>
```

The parameters have the following meaning:

– `<path>` Directory with preprocessed solver runs (contains `*.db` files).
– `<regex>` Java regular expression to restrict the analysis to those files in the directory that match the expression. The wildcard '.*' selects all files.
– `<howmany>` Upper bound for the number of matches to be written out.
– `<attribute>` Attribute to be written out after the analysis. The complete list is given in Figure 5.
– `<constraints>` List of constraints to filter the conflicts. If more than one constraint is specified only those conflicts are considered that fulfill the intersection of all constraints.

The following line would analyse and print the clashing graph pattern of all conflicts that contain more than 4 clauses for all instances in 'yourDir'. Only the top most 500 patterns will be printed:

```
java -jar CoPAn-analyser.jar "yourDir" ".*" "500" "ClashingGraph
                            NumberOfClauses>4"
```

The list of possible attributes is shown in Figure 5. It is analogous to the list of filters that can be used in the GUI version.

ClashingGraph	Pattern of clashing graph
NumberOfClauses	Number of clauses in the conflict
ResolutionClauseSize	Number of literals in learnt clause
ResSubsumesConflict	The learnt clause subsumes the conflict clause
PossibleSubsumptions	Number of possible subsumptions by the learnt clause
SubsumptionDeletion	Number of possible subsumptions after the deletion of the learnt clause
PossibleSelf-Subsumption	Analogous to PossibleSubsumption
Self-SubsumtionDeletion	Analogous to SubsumptionDeletion
LifeTime	Number of conflicts between creation and deletion of a clause
CouldBeLearnedEarlier	Consideres a learnt clause C_i that subsumes an older clause C_j. Checks whether the creation of C_i uses clauses that were learnt in between the creation of C_j and C_i (analogous to the GUI filter). If not the clause could have been learnt earlier.

Figure 5. List of possible attributes to be used in constraints.

1.6 Final Remarks

The downloadable package contains the binaries of CoPAn. The source is not available for public download due to licence restrictions of the obfuscated graph drawing library[1]. For first examinations the package also contains the solver output of our solver SApperloT[2] for two small SAT instances. Moreover, an additional package that contains the solver output of twenty more instances can also be downloaded from `http://algo.inf.uni-tuebingen.de/?site=forschung/sat/CoPAn`.

[1] http://docs.yworks.com/yfiles/doc/api/index.html
[2] http://algo.inf.uni-tuebingen.de/?site=forschung/sat/SApperloT