PCASSO – a Parallel CooperAtive Sat SOlver

Ahmed Irfan and Davide Lanti and Norbert Manthey
Knowledge Representation and Reasoning,
TU Dresden, Germany

***Abstract*—The SAT solver PCASSO is a parallel SAT solver based on partitioning the search space iteratively.**

I. Introduction

PCASSO uses two main methods: creating partitions and solving partitions. Partitions are created through *partition functions*, where a partition function is a function ϕ such that, given a formula F and a natural number $n \in \mathbb{N}^+$, $\phi(F, n) := (F_1, \ldots, F_n)$, where $F \equiv F_1 \vee \ldots \vee F_n$ and each pair of partitions is disjoint: $i \neq j \in [1, n],\ F_i \wedge F_j \models \bot$. Without loss of generality we assume that partitions $F_1, \ldots, F_n$ are always of the form $F \wedge K_1, \ldots, F \wedge K_n$, where $K_1, \ldots, K_n$ are sets of clauses, called *partitioning constraints*. By iteratively applying the partition function to a formula F, a *partition tree* is produced. Nodes in the partition tree are tagged with their *positions*: the root node F is tagged with the empty position ϵ; the i-th successor (from left to right) of a node F^p at position p is the node F^{pi} (see Figure 1). Please notice that, as positions are strings, the standard *prefix* relation among strings ($<$) is defined for positions as well.

II. Main Techniques

The partition function used in PCASSO is *tabu scattering*, which is an extension of *scattering* [1]. The idea of scattering is to define each partitioning constraint as conjunctions of *cubes* [2], where a cube is a formula $Q := \{C_1, \ldots, C_n\}$ such that $|C_i| = 1$, for each $1 \leq i \leq n$. Observe that the negation of a cube $Q := \{\{l_1\}, \ldots, \{l_n\}\}$ is the clause $\{\overline{l_1}, \ldots, \overline{l_n}\}$. More precisely, given a formula F_0 and an integer n, the n partitions $F_1, \ldots, F_n$ are created by using $n-1$ cubes $Q_1, \ldots, Q_{n-1}$ and applying them according to the following schema: $F_1 := F_0 \wedge Q_1$; $F_{m+1} := F_0 \wedge (\bigwedge_{i=1}^{m} \overline{Q_i}) \wedge Q_{m+1} (1 \leq m < n-1)$; and finally $F_n := F_0 \wedge \bigwedge_{i=1}^{n-1} \overline{Q_i}$. *Tabu scattering* adds the restriction to scattering that a variable used in one cube must not be used in the cubes for creating the remaining partitions. Using tabu scattering, we diversify the search more. PCASSO uses lookahead techniques [3] for choosing the literals (in cubes). In particular it chooses variables with the maximum *mixdiff* score [3]. The score *mixdiff* of a variable is the product of the *diff* score of each polarity of the variable. We calculate the *diff* score of the polarity of a variable by applying lookahead, and use the following weighted sum: 0.3 times the number of propagated literals plus 0.7 times the number of newly created binary clauses. After choosing the variable with the maximum *mixdiff* score, we choose the polarity of the variable that has the lowest *diff* score for creating cubes. We also use the following reasoning techniques: failed literals, necessary assignments, pure literals, and add learned clauses to the partition constraints. Techniques like constraint resolvent, double lookahead, and adaptive pre-selection heuristics are also used as proposed in the literature [3].

To describe the *node-state* of a node F^p at a certain point of execution we use a triple (F^p, s, r) where $s \in \{\top, \bot, ?\}$ ($\top$ indicates that an incarnation found a model for the node, whereas $\bot$ indicates that an incarnation proved unsatisfiability of F^p; finally, $?$ indicates that the node has not been solved yet) and $r \in \{\blacktriangleright, \blacksquare\}$ (indicating whether an incarnation is *running* on F^p or not, respectively). Given the notion of node-state, PCASSO exploits the *overlapped solving* strategy if two incarnations are allowed to run at the same time on nodes F^p, F^q such that $p \leq q$. In order to solve an unsatisfiable node F^p, either F^p has to be directly solved by some incarnation or each child node F^{pi} has to be solved. There is no limit on the solving time for each node.

Per variable, VSIDS activity and progress saving are shared from parent to child nodes. When PCASSO starts solving, the root node and the nodes at the partition tree level one start at almost the same time. The nodes at partition tree level greater than one are usually created after some time, so we initialize their search process with the VSIDS and progress saving information of their parent, because the child node searches in the sub-search space of its parent and whatever is learned by the parent search can help the solving child node as well.

Learned clauses are shared between incarnations to intensify the search. A learned clause is considered *unsafe* if it belongs to partitioning constraints, or it is obtained by a resolution derivation involving one or more unsafe clauses. A clause that is not unsafe is called *safe* clause, and only safe clauses are shared. A learned clause is shared in a sub-partition tree if it is safe in that sub-partition tree. This information can be calculated by tagging each clause with the position of the sub-tree where the clause is valid (*position-based* tagging [4]). We propose a dynamic learned clause sharing scheme, that is based on LBD scores [5]. A learned clause is eligible for sharing by an incarnation if the LBD score of this clause is lower than a fraction δ of the global LBD average of the incarnation. In PCASSO, we use $\delta = 0.5$.

PCASSO uses different restart policy and different clause cleaning policies for the nodes, depending whether the node is root, leaf or middle (not root and not leaf).

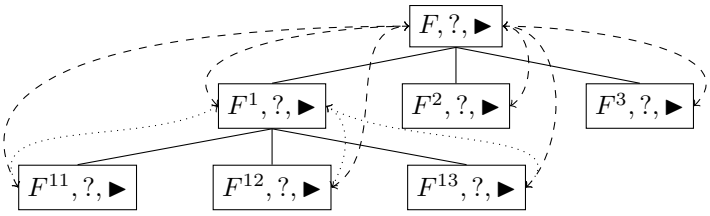

Fig. 1. Visualization of a partition tree with clause sharing and overlapped solving: The dashed lines represent the possible communication when flag based sharing is used, whereas the dotted and dashed lines together visualize the possible sharing with position based tagging.

PCASSO can have a scenario that there is only one unsolved node at some partition level. We call this scenario the *only child scenario*. Consider that if the only child scenario happens at some level of the partition tree, then there are two cases: i) the parent node is looking into the search space which has been solved by one of its children already, ii) the parent node is looking into the same search space where its unsolved children are looking. In either case, we have the risk of doing redundant work. We propose an approach to get out of this scenario by reintroducing the solving limit in a node that has only one unsolved child (AVOID). To be on safe side, we do not apply this limit for the root node. The introduced limit grows with the level of the node (level $*$ 4096 conflicts). Since in the only child scenario all learned clauses can be shared among the two participating nodes, we can also EXPLOIT this situation, by enabling this sharing. In the extreme case, this configuration is very similar to portfolio solvers, since then all clauses can be shared without restrictions. When clauses are tagged by position-based tagging [4], additional information can be obtained by performing a conflict analysis on solved unsatisfiable nodes. Consider a node $(F^p, \bot, \blacksquare)$, and let $\{\}^q$ be the empty clause derived by the incarnation that solved F^p. Then, from the main theorem in [4], we conclude that $\{\}^q$ is the semantic consequence of the node of position q in the partition tree. Observe that q is a prefix p: $q \leq p$. Consequently, not only the node at position p can be marked as unsatisfiable, but also the node F^q as well as all its child nodes. As a result, more incarnations can be terminated and start solving different partitions. We call this kind of technique *conflict driven node killing*. A similar approach is reported in [6].

III. MAIN PARAMETERS

The major parameters of the solver influence the number of threads that should be used, the number of partitions that should be created for each node, and how sharing should be performed. For the competition, we use 8 threads, and produce 8 partitions. Furthermore, we share learned clauses according to their LBD value. Finally, the treatment of the only-child scenario can be influenced.

For each of these big parts of the solver, many small parameters are provided, that control the special behavior of the system. There are only minor magic constants that control the run time of the look-ahead procedures during partitioning, which are chosen according to the literature [3].

IV. SPECIAL ALGORITHMS, DATA STRUCTURES, AND OTHER FEATURES

Each node in the partition tree is associated a *pool of shared clauses*, where a pool is implemented as a vector of clauses. This permits to decouple the life of a shared clause from the life of the incarnation where the shared clause has been learned. Instead of tagging each clause with a position, clauses are tagged with integers representing a *level* in the partition tree (root node has level zero). Each incarnation working over a node F^p can only access the pools placed at nodes of positions $q \leq p$. Concurrent access to pools is regulated by standard `POSIX` Read-Write locks.

COPROCESSOR is used as preprocessor [7].

V. IMPLEMENTATION DETAILS

PCASSO is built on top of GLUCOSE 2.2.

VI. SAT COMPETITION 2013 SPECIFICS

PCASSO has been submitted to both the application and the crafted parallel track.

VII. AVAILABILITY

The source code of PCASSO is available at tools.computational-logic.org under the GPL license.

REFERENCES

[1] A. E. J. Hyvärinen, T. Junttila, and I. Niemelä, "A distribution method for solving SAT in grids," in *Proc. of the 9th international conference on Theory and Applications of Satisfiability Testing*, ser. SAT'06. Springer-Verlag, 2006, pp. 430–435.

[2] M. J. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and conquer: Guiding CDCL SAT solvers by lookaheads," in *Accepted for HVC 2011*, 2012.

[3] M. J. H. Heule and H. van Maaren, *Look-Ahead Based SAT Solvers*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, February 2009, vol. 185, ch. 5, pp. 155–184.

[4] D. Lanti and N. Manthey, "Sharing information in parallel search with search space partitioning," in *Proc. of the 7th International Conference on Learning and Intelligent Optimization (LION 7)*, ser. LNCS. Springer, 2013.

[5] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proc. of the 21st international jont conference on Artifical intelligence*, ser. IJCAI'09. Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[6] A. E. J. Hyvärinen, T. Junttila, and I. Niemelä, "Grid-based SAT solving with iterative partitioning and clause learning," in *Proc. of the 17th international conference on Principles and practice of constraint programming*, ser. CP'11, 2011, pp. 385–399.

[7] N. Manthey, "Coprocessor 2.0: a flexible cnf simplifier," in *Proceedings of the 15th international conference on Theory and Applications of Satisfiability Testing*, ser. SAT'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 436–441. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31612-8_34