Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

SATPin

Axiom Pinpointing for Lightweight Description Logics through Incremental SAT

Norbert Manthey · Rafael Peñaloza · Sebastian Rudolph

Received: date / Accepted: date

Abstract One approach to axiom pinpointing (AP) in description logics is its reduction to the enumeration of minimal unsatisfiable subformulas, allowing for the deployment of highly optimized methods from SAT solving. Exploiting the properties of AP, we further optimize incremental SAT solving, resulting in speedups of several orders of magnitude: through *persistent incremental solving* the solver state is updated lazily when adding clauses or assumptions. This adaptation consistently improves the runtime of the tool by an average factor of 3.8, and a maximum of 38. SATPin, our system, was tested over large biomedical ontologies and performed competitively.

Keywords Pinpointing · SAT · Description Logics

1 Introduction

Axiom pinpointing (AP) is the task of identifying ontology axioms responsible for a logical consequence. It is used to understand and correct modeling errors in very large ontologies. For example, the 2007 version of Snomed incorrectly implied that *amputations of finger* were *amputations of arm*. Automated AP tools helped identify the 6 axioms (from $\sim 300,000$) causing this error (Baader and Suntisrivaraporn, 2008) and change the modelling strategy followed by the developers of Snomed, to avoid causing it again (Baader et al, 2009). AP has numerous applications, e.g. in context-based, error-tolerant reasoning (Baader et al, 2012; Ludwig and Peñaloza, 2014), and reasoning with probabilities, preferences, and provenance (Ceylan and Peñaloza, 2014; Riguzzi et al, 2015; Schenk et al, 2009).

For $\mathcal{EL}^+$ (Baader et al, 2005), Sebastiani and Vescovi reduce AP to propositional minimal unsatisfiable subformula enumeration, exploiting SAT developments (e.g., clause learning, two-watched-literal data structure) to build efficient AP systems. We build on this idea and identify new enumeration optimizations based on the specific shape of the propositional formula constructed. Incremental SAT solving, partial restarts, and an improved search space pruning strategy can considerably increase the efficiency of AP. While search-space pruning via clause learning is known in SAT, it has never been used for AP. Partial restarts are also used in SAT, but not for consecutive calls to a SAT solver in incremental SAT solving. We introduce *persistent incremental solving*, a *lazy* approach preserving relevant information between runs, which has applications way beyond AP. We also use *assumption prefetching*, which modifies the testing order of the assumption literals aiming detecting conflicts earlier. Our Minisat-based SATPin system uses these optimizations. Experiments show that SATPin is efficient for AP over large inputs. We compare SATPin and its modified Minisat to other state-of-the-art SAT solvers via the IPASIR interface observing that IPASIR solvers cannot keep up with the partial restart modification from SATPin.

N. Manthey · S. Rudolph
Technische Universität Dresden, Germany
E-mail: nmanthey@conp-solutions.com
E-mail: sebastian.rudolph@tu-dresden.de

R. Peñaloza
University of Milano-Bicocca, Italy
E-mail: rafael.penaloza@unimib.it

2 Enumerating MinAs with SAT Technology

We assume familiarity with description logics (DLs) and SAT. A DL ontology is a set of axioms in a given syntax, and a consequence is a statement which can be logically

derived from this ontology. $\mathcal{EL}^+$ axioms are $C \sqsubseteq D$ or $r \sqsubseteq s$ with C, D $\mathcal{EL}$ concepts, and s, t role names. For the ontology $\mathcal{O}$ and consequence c, a *MinA* is a minimal (w.r.t. set inclusion) subset of $\mathcal{O}$ from which c still follows. *Axiom pinpointing* is the task of finding all MinAs. A similar problem exists in SAT. A CNF formula is a set of clauses. A *MUS* for an unsatisfiable formula is a minimal unsatisfiable subformula. Partitioning the formula into sets of clauses, a *group-MUS* is a minimal union of partitions which is unsatisfiable. One is interested in enumerating all group-MUSes (Liffiton and Malik, 2013; Previti and Marques-Silva, 2013).

AP on $\mathcal{EL}^+$ is reducible to all-group-MUS enumeration (Sebastiani and Vescovi, 2009). The approach constructs a Horn formula whose variables correspond to axioms or consequences from an ontology, and whose clauses represent derivation steps in a reasoning algorithm. This formula together with the negated consequence is used as one partition, while new clauses containing variables for the original axioms are added as singleton partitions called assumption variables. Hence, the results from *all-group-MUS* enumeration correspond to the MinAs from the ontology. For example, for the ontology $A \sqsubseteq B, A \sqsubseteq C, B \sqsubseteq D, B \sqcap C \sqsubseteq D, C \sqsubseteq E$ and the consequence $A \sqsubseteq D$, the approach builds the formula F and query literal:

$$\begin{gathered} \neg x_{A\sqsubseteq B} \vee \neg x_{B\sqsubseteq D} \vee x_{A\sqsubseteq D}, \\ \neg x_{A\sqsubseteq B} \vee \neg x_{A\sqsubseteq C} \vee \neg x_{B\sqcap C\sqsubseteq D} \vee x_{A\sqsubseteq D}, \\ \neg x_{A\sqsubseteq C} \vee \neg x_{C\sqsubseteq E} \vee x_{A\sqsubseteq E}, \qquad \neg x_{A\sqsubseteq D} \end{gathered}$$

and assumptions $x_{A\sqsubseteq B}$, $x_{A\sqsubseteq C}$, $x_{B\sqsubseteq D}$, $x_{B\sqcap C\sqsubseteq D}$, $x_{C\sqsubseteq E}$. A group-MUS is $F \cup \{\neg x_{A\sqsubseteq D}, x_{A\sqsubseteq B}, x_{B\sqsubseteq D}\}$ corresponding to the MinA $\{A \sqsubseteq B, B \sqsubseteq D\}$.

Henceforth, $\overline{\phi}$ is the negation of ϕ. SATPin uses a novel enumerator as a CDCL-based SAT solver. An inner solver finds models (conjunctions of literals) of a formula using the assumption variables, starting from `true`. If a model I is found, $\overline{I}$ is added to the formula, and a new model is searched. Each model is a MinA candidate which is sent to an outer solver to verify that $F \cup I$ falsifies the query literal. For compactness, we add only the decision literals of I; all others are implicit. Repetitions are avoided through the clause $\overline{R}$ for each MinA R. The enumeration incrementally adds the variables of the last MinA to the solver as a clause. Previous clauses remain valid: candidates are not enumerated twice and known MinAs are not repeated. Fig. 1 shows the basic design: an inner enumerator based on MiniSat[1] or an IPASIR solver (Balyo et al, 2016), which implements strategies for finding new MinA candidates, and an outer solver for verifying candidates. The details are explained next.

[1] `https://github.com/niklasso/minisat`

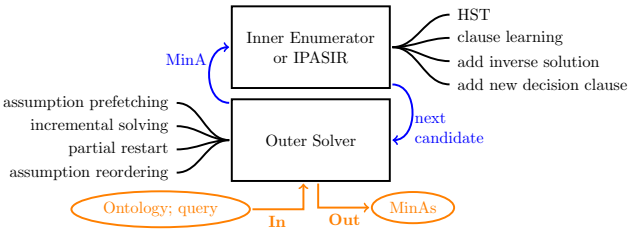


Fig. 1: The basic architecture of SATPin.

Combining hitting-set-tree (HST) enumeration with clause learning is not trivial, as propagation and learned clauses may *fast-forward* the HST enumeration, skipping necessary combinations and resulting in unseen MinAs. An HST-like behaviour is kept via a new decision heuristic. A stack keeps found solutions and the decision heuristic systematically picks decision literals not contained in previous MinAs. As this becomes costly for many and large MinAs, we add a fallback option to the standard variable state independent decaying sum (VSIDS) (Moskewicz et al, 2001) decision heuristic from MiniSat for candidate enumeration; triggered after a given number of MinAs have been found. To ensure that all future violating sets are different, a clause requires at least one new decision variable. This also ensures that a literal is removed from all new candidates.

Candidate Enumeration MiniSat enumerates models of a formula "densely," requiring the variables in the formula to be numbered by the first n natural numbers. We map the variables appearing at some MinA to a smaller set. MiniSat generates models over this reduced set of variables only, which are turned into candidates through the reverse mapping.

Incremental Solving The incremental solving process from MiniSat was modified so that restarts do not jump to decision level 0 of the outer solver as commonly done, but only to the level where the last assumption was used as a decision variable. This keeps many decision levels and limits the propagation needed to re-generate the trail of the solver at the next call.

Assumption Prefetching An *early refined cores* strategy (Manthey, 2015) aborts a search if unit propagation causes a conflict on an assumption-only decision level. As the only assumption literal which can be falsified is the query literal q, we test if q is falsified before each search decision. Our solver always places this literal as the last assumption. *Assumption prefetching* is applied when a decision literal is chosen based on assumptions. If an assumption is the next decision literal, we check whether q is falsified. If so, we alter the order of the assumption literals and move q to become the next assumption literal. Thus, many assumption literals can be skipped. As a side effect, found conflicts (MinA can-

didates) are smaller. As it is hard to predict which assumption literal should be checked before each decision, this is usually not used by generic SAT solvers.

IPASIR Different solvers can be used for the candidate validation via the *reentrant incremental SAT solver API* (IPASIR): an IPASIR solver is initialized with the formula; next, MINISAT calls are replaced by forwarding assumptions to the solver, calling its search routine, and copying the model and conflict information back into the SATPIN data structures. The integration of IPASIR solvers allows for an improved performance, by the use of state-of-the-art solvers. However, it introduces a communication overhead, as all assumptions must be sent at each call. When the internal MINISAT, which does not support generic calls, is used, only local changes to internal data structures are needed, saving several computation steps.

3 Persistent Incremental SAT Solving

During the entailment check, all literals $x \in X$ are applied. Depending on the ontology, X can be very large but the relevant literals V may be much fewer; the encoding of NCI contains 46,800 literals (Table 1), but V contains only 0.09% of the literals. For each imply check, the remaining 99.91% literals could be dismissed immediately. Instead of initializing a consistent set of assumptions M as the empty set, we initialize $M = T$ to the set of (currently) irrelevant literals, as explained later. This is sound, since $(F \wedge T) \not\rightarrow q$, where F is the formula. Theoretically, this optimization improves the algorithm by several orders of magnitude making the difference between solvability and infeasibility.

Incremental SAT solvers reset the internal interpretation of the solver after each call. We implemented *persistent incremental solving* (PIS) in MINISAT's SOLVE method as part of the model search. The closest related work is *partial restarts* with *matching trails* (van der Tak et al, 2011). SATPIN updates the set M. The incremental SAT call finds either a conflict w.r.t. the current assumptions, or a model. A conflict implies the presence of a new MinA, which updates the relevant literals V and adds the conflict clause to the solver to avoid repetitions. When keeping the current partial interpretation in the solver, the new clause is still falsified; it is a conflict. An invariant on the solver state is that at least one literal of each clause added to a watch list is falsified. To avoid unit propagation triggering from single literal satisfaction, we jump back to free at least two literals. This decision takes into account that multiple clauses may be added during incremental SAT solving, and unit propagation and adding clauses might take turns.

If a model is found, all assumptions can be set without a conflict after unit propagation, and a new set of assumptions is given to the solver. To minimize the change, the new assumptions maximize the common prefix; that is, as many of the previous assumptions as possible are preserved in the same order. SATPIN keeps all stable literals in the prefix, preserving their order as well as possible: we store all relevant literals V that change at the back.

As the truth value assignment may change with each call, regardless of the satisfiability of the previous result, assumptions are stored in two lists. Literals appearing in some MinA are in the list of mutating literals V, and the remaining literals form the list T. Literals can only move from T to V. To quickly extract the literals of T, for each $w \in T$ we store the position p_w of w. When removing the literal w, we erase w from T, and move the very last literal w' still appearing in T to the position p_w, updating the position information for w': $p_{w'} = p_w$. For all updates, we save the smallest changing position. Before the next call to the SAT solver, we reset this decision.

For MINISAT-based solvers this idea works because each assumption literal creates a new decision level.[2] After jumping back, new assumptions replace the current ones without extra care; the algorithm ensures that the prefix of the two lists of literals remains the same. After a satisfiable call, only the literals in V change, and the full saving applies. For unsatisfiable calls, our algorithm might jump back fully, as the first literal in V might have been part of the current MinA and no prefix can be preserved.

The idea of persistent incremental calls can be easily adopted by any incremental solver which: (i) does not clear the trail after the search ends; (ii) integrates new clauses after ensuring there are two literals that are not falsified; and (iii) when being called with a new set of assumptions, reuses the common prefix. In SATPIN, the integration is tightly coupled with model enumeration. For use cases of incremental SAT solving like hardware model checking via IC3, this may have a great effect as well: many, mostly cheap, calls to SAT solvers are performed. Software model checking can also rely on incremental search with many assumption variables. The same is true for other applications such as Satisfiability Modulo Theory (SMT) solving or MUS extraction. Their final effectiveness depends the number of calls to the solver (the higher, the better), the size of the common assumption prefix among calls (the higher, the better), and the simplicity of each call to the SAT solver (the simpler or fewer conflicts per call, the better).

[2] Even if the assumption literal is already satisfied.

Table 1: Structure of the translation of ontologies

	GO	NCI	FullGalen	NotGalen
Axioms ($\|X\|$)	20,466	46,800	36,544	4,379
Variables	237,389	338,380	2,729,734	125,193
Clauses	294,782	342,825	3,844,125	148,103

We implemented these modifications into SATPin and Riss (Manthey, 2014), to compare the performance with and without PIS. Our SATPin implementation[3] uses Minisat 2.2. The data used for the analysis is at `https://doi.org/10.5281/zenodo.3739095`.

4 Experimental Evaluation

We ran SATPin on four well-known $\mathcal{EL}^+$ biomedical ontologies, typically used as benchmarks for DL reasoners in the context of AP: the Gene Ontology (GO), NCI, the $\mathcal{EL}^+$ version of FullGalen, and NotGalen. These benchmarks, originally designed by Sebastiani and Vescovi, have since been used for testing AP in $\mathcal{EL}^+$. Computations ran with a 3 hour timeout on an Intel Xeon CPU at 2.6GHz and a memory limit of 6.5GB.

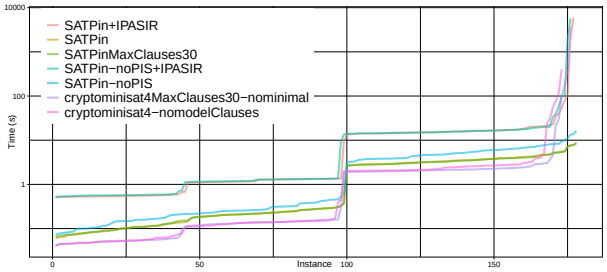
We compare the performance of SATPin with different back-ends and optimizations, to identify which components improve efficiency, and test if recent SAT reasoners outperform Minisat. Cryptominisat is the default IPASIR solver. The back-ends are either the internal, fully accessible Minisat, other solvers used in the incremental track of the most recent SAT Competition, or the modified Riss. From the optimizations we enabled and disabled PIS and conflict minimization in all four possible combinations. We added a switch from HST-tree based model candidate enumeration to the usual search based candidate enumeration after a given amount of violating sets—in our case 30—was found, to check whether the complexity constraint of the HST-based implementation is a bottle neck. All other enumeration techniques from Section 2 are always enabled.

Ontologies were transformed to a Horn formula by *el2sat_all* (Sebastiani and Vescovi, 2009). Table 1 shows the properties of these ontologies and their translations; the number of axioms in the original ontology is the number of selection variables used by SATPin. For each ontology, we computed all MinAs for 100 different consequences as designed by Sebastiani and Vescovi (2015): 50 randomly chosen, and the 50 whose query variable appears most often in the translated formula, which is a proxy for having the most MinAs.[4]

[3] `https://github.com/conp-solutions/satpin`

[4] The original design included the much larger Snomed. We were unable to match the version and the test cases used by Sebastiani and Vescovi and hence do not consider it in our tests. Still, our results are robust across ontologies.

Fig. 2: Cactus plot comparing the overall behaviour of seven different solvers on instances with only one MinA.



Previous work (Manthey et al, 2016) sets SATPin as a competitive AP tool with performance differences noticeable at very large and hard instances. Since then, BEACON (Arif et al, 2016) and PULi (Kazakov and Skocovský, 2018) emerged. A comparison between all tools is beyond the scope of this paper, which focuses on the optimizations of SATPin. Our experiments are designed to evaluate the need of these optimisations, and the overall effect of PIS.

Hypothesis 1 *For the class of simple problems with only one MinA, runtime does not differ significantly.*

All solvers need warm-up time for loading the formula and initialization, specially for large formulas. With only one MinA, the overall runtime is small, and the number of calls to the solver, along their savings, are limited by the size of that MinA; improving a solver by a constant factor per call does not pay off in these cases. Only 3 of 13 variants tested solved all the instances with one MinA, and even within those solved, most solvers show a few outliers taking over 3,000s. Ignoring these, all systems behave almost identically, suggesting that the increased runtime is caused by external factors. Fig. 2 shows a cactus plot for the results by five variants of SATPin with different optimizations, and the two best IPASIR variants; the vertical axis uses a logarithmic scale. The behaviour of other variants tested was similar to the latter two. The three solvers that found all answers (SATPin, SATPin with maximal model clauses—SATPinMaxClauses30, and SATPin without PIS—SATPin-noPIS) behave similarly overall, but the latter is slightly slower in all instances, suggesting that PIS is useful even for simple cases.

Considering all instances, the variant with maximal model clauses shows an improvement, answering an open question by Manthey et al (2016): after 30 MinAs, the candidate enumeration algorithm switches from HST based enumeration to plain SAT search. The

Fig. 3: Runtime comparison of configurations of SATPin with Riss

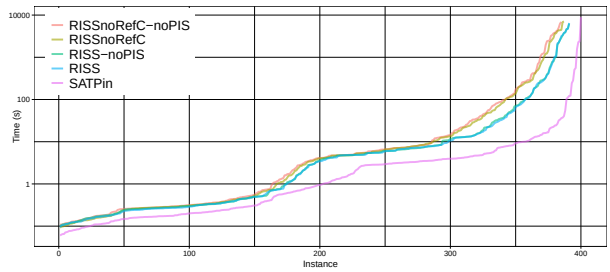
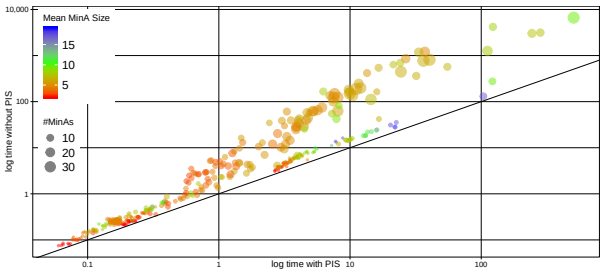


Fig. 4: Runtime of SATPin with and without PIS, w.r.t. MinA number (dot size) and average size (color)



latter is much quicker (at the cost of a few more SAT solver calls) saving considerable runtime. Since most benchmarks have 30 MinAs or less, they are not affected by this option, and both configurations behave identically. The cases with a few MinAs are relevant; in fact, 178 of the 400 instances have one MinA. Still, for larger ontologies, limiting the HST algorithm may have a more significant impact, and should be considered.

Another question is whether using PIS on a recent SAT solver via IPASIR improves performance. We linked Riss to SATPin via IPASIR implementing PIS as a generic method to test whether our improvement for incremental solving meets the expectations. Fig. 3 shows the runtimes of SATPin with Riss, and plain SATPin, where the vertical axis uses a logarithmic scale. Besides toggling PIS in Riss, we considered the *reverse conflict refinement* (RCR) from Riss, which is applied to conflict clauses generated by incremental solving.

Runtime differences for PIS are minimal, but RCR significantly improves it. A huge gap between SATPin and Riss is seen, which increases with the runtime spent for a problem. A part of the explanation why PIS is not effective with IPASIR is that the number of calls to the solver increases and problems with larger formulas tend to be harder. More calls with more assumption literals introduce communication overhead to IPASIR, forwarding all assumptions to Riss, while SATPin just modifies a small subset. With RCR enabled, the number of violating sets of Riss equals the number of MinAs. Other violating sets contain redundant variables which are taken into account during the candidate enumeration phase of SATPin, explaining the improved solver runtime.

Hypothesis 2 *PIS speedup increases with the number of calls to the solver, the number of assumption literals per call, the ratio of variables in MinAs to all assumption variables, and the number and size of MinAs.*

PIS saves work at each call, as only a small part of all variable assignments is considered for finding an answer. This avoids different kinds of steps like: generating decision literals from assumptions, traversing their watch lists to check for further propagation, and moving watched clauses to other watch lists, where a clause may be checked multiple times. Hence multiple instructions and memory reads and writes are saved. As more calls are performed, more steps are saved more often. When the number of assumptions is small, e.g. in the order of 100 literals, saving even 99% of the work is not big enough to make the saving per call significant, even when multiplied with a high number of calls. However, when the number of assumption literals grows (e.g. comparing $100,000$ vs. $1,000$ steps) the effect becomes noticeable. Similarly, the percentage of steps saved per call should be high for the effect to be meaningful; if e.g., only 5% of the steps is saved per call, the benefits could be potentially countered by a better data structure, but saving 95% of the work makes each call to the solver significantly faster, resulting in a noticeable effect overall. The percentage of savings is directly related to the set of relevant literals V in our algorithm, as PIS keeps the prefix of the assumptions stable. The number of calls to the solver is proportional to the number of MinAs. As multiple MinAs lead to more variables involved in them,[5] the relevant set of literals also increases in the presence of new results. This should decrease the speedup, as it reduces the number of saved steps in successive calls. Indeed, the possible combinations of truth assignments to the literals in V increases exponentially, and so does the number of calls to the solver. Similarly, if MinAs are larger, the number of possible combinations to be checked with the solver grows faster than with fewer variables.

Our experiments show that the differences between using PIS or not are more pronounced as more restarts are made, and depend on the number of MinAs and their average size (Fig. 4). For points closer to the diagonal (with moderate speedup) the size of the point is also small, whereas the data points in the top half of the diagram become bigger signalling a correlation.

[5] New MinAs may contain previously unseen variables.

5 Conclusions

SATPin exploits highly optimized SAT algorithms and data structures for efficient AP in $\mathcal{EL}^+$. Our approach is based on the construction of a Horn formula encoding the completion-based procedure for atomic subsumption. The methods proposed generalize to any ontology language with consequence-based reasoning algorithms (Peñaloza et al, 2017). Once the propositional formula is obtained, all optimizations apply.

We present *persistent incremental SAT solving* (PIS), which is as lazy as possible during two consecutive SAT calls. The main goal of PIS is to preserve the state of an incremental solver by: not resetting all decisions on each call; integrating added clauses into the watch data structures by minimal jump backs where two literals can be watched without propagation; and resetting to the matching prefix of the previous and current assumption list. As SATPin relies on a special purpose SAT solver, we added *assumption prefetching*, to check the truth value of a dedicated assumption literal before considering other assumptions.

Empirically, not using PIS significantly degraded performance, even when exchanging Minisat with recent IPASIR-based solvers. This performance degradation may be due to SATPin making many simple SAT calls, with many assumption literals, where most of the assumptions stay static between calls. We observed empirically that the performance of SATPin degrades as the number and size of the MinAs increases. Other solvers suffer from a harder slowdown under these conditions. To avoid this degradation, we change the enumeration strategy after many MinAs have been found.

Acknowledgements We thank ZIH at TU Dresden for the computational resources for the evaluation. S. Rudolph is supported by the European Research Council (ERC) Consolidator Grant 771779 *A Grand Unified Theory of Decidability in Logic-Based Knowledge Representation* (DeciGUT).

References

Arif MF, Mencía C, Ignatiev A, Manthey N, Peñaloza R, Marques-Silva J (2016) BEACON: an efficient sat-based tool for debugging EL+ ontologies. In: Proc. of SAT 2016, Springer, LNCS, vol 9710, pp 521–530

Baader F, Suntisrivaraporn B (2008) Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Proc. of KR-MED'08, CEUR-WS, vol 410

Baader F, Brandt S, Lutz C (2005) Pushing the $\mathcal{EL}$ envelope. In: Proc. IJCAI-05, Morgan-Kaufmann

Baader F, Schulz S, Spackmann K, Suntisrivaraporn B (2009) How should parthood relations be expressed in SNOMED CT? In: Proc. of OBML 2009

Baader F, Knechtel M, Peñaloza R (2012) Context-dependent views to axioms and consequences of semantic web ontologies. J of Web Sem 12–13:22–40

Balyo T, Biere A, Iser M, Sinz C (2016) {SAT} race 2015. Artificial Intelligence 241:45–65

Ceylan II, Peñaloza R (2014) The Bayesian Description Logic $\mathcal{BEL}$. In: Proc. of IJCAR'14, Springer, LNCS, vol 8562, pp 480–494

Kazakov Y, Skocovský P (2018) Enumerating justifications using resolution. In: Proc. of IJCAR 2018, Springer, LNCS, vol 10900, pp 609–626

Liffiton MH, Malik A (2013) Enumerating infeasibility: Finding multiple MUSes quickly. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, LNCS, vol 7874, pp 160–175

Ludwig M, Peñaloza R (2014) Error-tolerant reasoning in the description logic el. In: Proc. of JELIA'14, Springer, LNAI, vol 8761, pp 107–121

Manthey N (2014) Riss 4.27. In: Proc. of SAT Comp. 2014, University of Helsinki, Department of CS Series of Publications B, vol B-2014-2, pp 65–67

Manthey N (2015) Refining unsatisfiable cores in incremental SAT solving. Tech. rep., TU Dresden

Manthey N, Peñaloza R, Rudolph S (2016) Efficient axiom pinpointing in EL using SAT technology. In: Proc. of DL 2016, CEUR Ws, vol 1577

Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: Engineering an efficient SAT solver. In: Proc. DAC 2001, ACM, pp 530–535

Peñaloza R, Mencía C, Ignatiev A, Marques-Silva J (2017) Lean kernels in description logics. In: Proc. ESWC 2017, Springer, LNCS, vol 10249, pp 518–533

Previti A, Marques-Silva J (2013) Partial MUS enumeration. In: Proc. of 27th AAAI, AAAI Press

Riguzzi F, Bellodi E, Lamma E, Zese R (2015) Probabilistic description logics under the distribution semantics. SWJ

Schenk S, Dividino R, Staab S (2009) Reasoning with provenance, trust and all that other meta knowlege in OWL. In: SWPM, CEUR-WS.org, CEUR, vol 526

Sebastiani R, Vescovi M (2009) Axiom pinpointing in lightweight description logics via Horn-SAT encoding and conflict analysis. In: Proc. of 22nd CADE, Springer, LNCS, vol 5663, pp 84–99

Sebastiani R, Vescovi M (2015) Axiom pinpointing in large $\mathcal{EL}$+ ontologies via SAT and SMT techniques. Tech. Rep. DISI-15-010, University of Trento, Italy

van der Tak P, Ramos A, Heule M (2011) Reusing the assignment trail in CDCL solvers. JSAT 7(4):133–138