Chapter 6
Exploiting Electronic Design Automation for Checking Legal Regulations: A Vision

Oliver Keszocze and Robert Wille

Abstract Legal regulations are large and complex documents that require experts such as lawyers to be understood. Working with these documents is a manual and time- consuming task. Common use cases are to decide whether a submission is conform with the regulations or to check whether certain corner cases are possible in the given set of rules. We envision to address many of these problems by treating legal regulations in the same manner as system specifications. This allows to apply sophisticated formal methods from *Electronic Design Automation* (EDA). For this, we briefly discuss the process of (semi)-automatically formalizing legal regulations. Afterwards, we illustrate the correspondence of various problems in the considered domain (here: regulations on scales and fees for medical doctors) with well-known EDA problems. We sketch the application of formal methods by means of examples and envision that in the future, the exploitation of formal methods to analyse legal regulations will greatly help lawmakers and "end users" alike.

6.1 Introduction

In *Electronic Design Automation* (EDA), circuits and systems are realised based on an initially given specification. During the design process, this specification is implemented using proper hardware or system description languages such as SystemC [10], System Verilog [11], or VHDL [9]. Due to the strive for higher levels of abstraction, the application of modelling languages such as the *Unified Modeling Language* (UML, [15]) or the *Systems Modeling Language* (SysML, [19]) found recent interest in the design of circuits and systems.

O. Keszocze (✉)
Group for Computer Architecture, University of Bremen, Germany

Cyber-Physical Systems, DFKI GmbH Bremen, Germany
e-mail: keszocze@informatik.uni-bremen.de

R. Wille
Institute for Integrated Circuits, Johannes Kepler University Linz, Austria

Cyber-Physical Systems, DFKI GmbH Bremen, Germany
e-mail: robert.wille@jku.at

© Springer International Publishing Switzerland 2016

F. Oppenheimer, J.L. Medina Pasaje (eds.), *Languages, Design Methods, and Tools for Electronic System Design*, Lecture Notes in Electrical Engineering 361,
DOI 10.1007/978-3-319-24457-0_6

All these developments have in common that they transform a specification, which is usually provided in a natural language, into a formal representation. At the same time, the resulting formal descriptions are subject to various correctness checks for which a variety of (automatic) EDA methods have been proposed in the past. More precisely, very powerful methods are in practical use today which check, e.g. the equivalence of two realizations (see, e.g., [2]), prove whether certain transitions in a system are possible (see, e.g., [1]), or generate test patterns that identify faults in the physical realization (see, e.g., [7]). These EDA methods became an integral part of today's design flow and emerged to powerful tools which can handle designs of considerable complexity.

At the same time, these methods also provide significant potential to application areas beyond EDA. In this chapter, we investigate such an application area, namely the consideration of legal regulations.

Legal regulations can be understood in a similar fashion as a specification for a technical system. The only difference is that legal regulations describe rules and requirements of a certain aspect of the daily life instead of something technical such as a circuit or system. But this difference is just of superficial relevance for the applied EDA methods. In fact, also legal regulations are initially provided in natural language which can be formalized (first ideas have been already proposed, see, e.g., [12, 17]). Based on this formal representation, checks for "correctness", e.g. plausibility, uniqueness, the existence of desired scenarios, and many more can be conducted.

In this chapter, we envision and discuss possible exploitations of existing EDA methods in order to check legal regulations. For this purpose, we briefly review a selection of EDA methods and sketch available approaches aiming for the formalization of legal regulations. Based on that, we discuss and illustrate how the EDA methods available for the design of circuits and systems together with a formal representation of legal regulations can be exploited. All considerations are conducted using the German Regulations on Scales of Fees for Medical Doctors (German: Gebührenordnung für Ärzte [8]) as an example.

The remainder of this chapter is organized as follows. In Sect. 6.2, we briefly review EDA methods that are used to verify the correctness and behaviour of circuits and systems. The reviewed solutions are based on Boolean satisfiability. Afterwards, in Sect. 6.3, the domain of interest, legal regulations that control how medical doctors may invoice their services, is introduced. Furthermore, the method of extracting a formal model from natural language texts is sketched. Then, Sect. 6.4 introduces a variety of applications for EDA methods in this context and explains them by means of examples. Finally, in Sect. 6.5, the work is concluded.

6.2 Typical Methods for Electronic Design Automation

Every year in the past decades, the design of circuits and systems grew more and more complex. The resulting complexity eventually motivated the development of methods for *Electronic Design Automation* (EDA). Moreover, search spaces

to be traversed became larger and larger so that, today, methods are in place which employ dedicated search and learning strategies in order to cope with the underlying computational complexity. Amongst many other methods, solvers for *Boolean satisfiability* (so called *SAT solvers*; see, e.g., [5, 6]) represent one of the most popular of these methods. In this section, we briefly review the basics on Boolean satisfiability, their solving schemes, as well as application areas. This provides representatives of EDA methods whose application to the domain of legal regulations is envisioned afterwards.

6.2.1 Boolean Satisfiability and SAT Solvers

The *Boolean satisfiability problem* (SAT problem) is to determine an assignment α to the variables of a Boolean function f such that $f(\alpha)$ evaluates to true or to prove that no such assignment exists. Often, f is given in *Conjunctive Normal Form* (CNF). A CNF consists of a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation.

Example 1. Let $f = (x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2)(\bar{x}_2 \vee \bar{x}_3)$. Then $x_1 = 1 = x_2 = 1, x_3 = 0$ is a satisfying assignment for f. The value of x_1 ensures that the first clause becomes 1 while x_2 ensures this for the second as does x_3 for the third clause.

The SAT problem is one of the central NP-complete problems. In fact, it was the first known NP-complete problem as was proven by Cook in 1971 [3]. Despite this proven complexity, SAT algorithms are nowadays capable of handling practical problem instances of considerable size, i.e. SAT instances which are composed of hundreds of thousands of variables, millions of clauses, and tens of millions of literals. Most of these SAT solvers are based on backtracking algorithms and use three essential procedures: (1) the decision heuristic assigns values to free variables, (2) the propagation procedure determines implications resulting from the last assignment(s), and (3) the conflict analysis tries to resolve conflicts that occur during the search by clever backtracking schemes. Advanced techniques such as *efficient Boolean constraint propagation* [14] and *conflict analysis* [13] are common in state-of-the-art SAT solvers (see, e.g., [5, 6]) and strongly contribute to their effectiveness.

6.2.2 Applications in EDA

The efficiency as well as the clever traversal schemes of SAT solvers have found numerous application for many computationally complex problems—including several EDA problems. In the following, just a selection is briefly sketched:

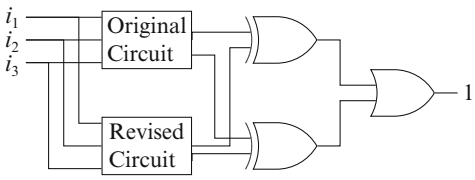


Fig. 6.1 Miter structure

- *Equivalence Checking*
  During the design process, the originally determined circuit is usually revised a couple of times (e.g. for the purpose of optimization). After each revision, it is important to prove whether or not the revised circuit is still functionally equivalent to the original one. This problem can be addressed by formulating the resulting equivalence checking problem by means of a so-called *miter structure* [2]. Here, it is made sure that always the same input assignments are applied to both circuits. Furthermore, XORs are added to corresponding output pairs in order to detect possible differences. Figure 6.1 shows the resulting structure. Then, if at least one XOR evaluates to 1 (determined by an additional OR operation), the two circuits are not equivalent. As this obviously represents a satisfiability problem ("Does there exist an assignment to the inputs of the circuits such that the output of the miter structure evaluates to true?"), the resulting SAT instance can easily be formulated. Then, all possible inputs assignments are *symbolically* considered which guarantees that a possible assignment leading to different outputs is determined when it exists.
- *Reachability Analysis*
  Recent developments of formally specifying complete systems using description languages allow for further automated analysis. One particular problem is to check whether certain states of the system, desired or undesired, are reachable [1]. The procedure is to symbolically translate all possible transitions of the system into a SAT formulation. Afterwards, a clause representing the state of interest is added. If the whole SAT formulation is satisfiable, the given state is reachable. In case of undesired behaviour, this indicates a bug in the system.
- *Automatic Test Pattern Generation (ATPG)*
  As a final example, ATPG considers the determination of an assignment to all primary inputs of a given circuit which, based on a given fault model, will show a possible erroneous behaviour at the primary outputs of this circuit [7]. This problem can also be formulated as a corresponding SAT problem ("Does an assignment showing the erroneous behaviour exist?"). ATPG is an interesting example, since existing EDA methods for this problem are capable of handling circuits and systems composed of several hundreds of thousands of components.

6.3 Formal Representation of Legal Regulations

Usually, legal regulations are provided in natural language. But in order to exploit EDA methods for checking them, a formal representation is required. Recently, researchers investigated several strategies and schemes how to efficiently formalize legal regulations (see, e.g., [12, 17]). Besides that, many software tools exist in which legal regulations (e.g. tax code) have (manually) been incorporated in order to support "end users" (e.g. citizens preparing their tax returns). In this section, we discuss existing formalizations of legal regulations which, eventually, provide possible inputs for EDA-inspired verification methods. We exemplary focus on legal regulations from the domain of doctoral fees, namely the *German Regulations on Scales of Fees for Medical Doctors* (German: Gebührenordnung für Ärzte [8], abbreviated as GOÄ in the following).

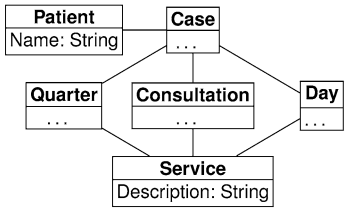
The GOÄ specifies how doctors are supposed to generate invoices for their services. The GOÄ eventually defines which services can be accounted and to what extent. As an example, Fig. 6.2a shows a particular regulation from the GOÄ.[1] Medical doctors, hospitals, etc., are supposed to prepare their invoices according to these regulations.

However, before the submitted invoices are indeed paid, corresponding authorities check whether the resulting invoices are in line with the regulations of the GOÄ. Due to the large amount of regulations, this is usually performed automatically

a

"The service 45678 is not to be billed together with GOÄ position 54687 within the third and fourth quarter of 2014." (*Die Leistung 45678 ist im dritten und vierten Quartal 2014 nicht neben Position 54687 berechnungsfähig*)

b



c

```
if (
      Case.quarter >= Quarter(2014,3) &&
      Case.quarter <= Quarter(2014,4)
) {
      if (together(45678,54687) {
            // do not bill and report error
      }
}
```

Fig. 6.2 Considered domain and problem. (**a**) A GOÄ rule describing a mutual exclusion of services (German sentence in parentheses). (**b**) Simplified domain of the GOÄ. (**c**) Resulting DSL expression

[1]Note that all regulations in the GOÄ are originally provided in German. However, in order to describe the proposed ideas, all examples have been translated.

by software tools. To this end, a formal representation of the respective GOÄ regulations has to be available.

Thus far, this formal representation is created manually—using a *Domain-Specific Language* (DSL) as well as a proper model which represents the structure of the respective processes (i.e. the performed services and the instances in time in which these services have been provided.[2]) Fig. 6.2b provides a sketch of the applied model: All possible *services* that doctors may perform are grouped by different criteria (e.g. *consultations, days*, or *quarters*) and collected in a single *case*. Each case is then related to a *patient*. A single service may require multiple consultations and a doctor may perform multiple services in a single consultation.

With this background, the original regulation from Fig. 6.2a can formally be represented as shown in Fig. 6.2b. Note that this expression is not uniquely determined and may look completely different, depending on the software developer's preferences. With formalizations like that, software tools checking invoices can easily be developed.

While this approach still requires significant manual effort in order to create the respective formal descriptions, another approach formalizing the corresponding GOÄ regulations has been proposed in [12]. Here, a (semi-) automatic method is proposed which is composed of two steps. In the first step, the sentences are preprocessed and grouped, while the second step uses methods from *Natural Language Processing* (NLP, [4, 16]) to create the DSL expression.

In the first preprocessing step, synonyms are normalized. Given a knowledge base (filled e.g. by an expert from the domain), words like "service" and "GOÄ position" can be matched in order to simplify the analysis. Afterwards, the sentences are simplified by shortening long enumerations. These enumerations do not carry any information in the NLP step besides "an amount of services". Therefore, terms such as "services 12345, 23456, 34567, and 98765" are reduced to placeholder-terms, e.g. by hash-tag-like identifiers such as "01234567.0". These are still understood as a number by NLP tools while not resulting in a complicated structure that renders the further analysis impossible. Further simplifications such as removing long descriptions of equipment are conducted as well.

After these preprocessing steps, actual natural language processing of the regulations (still provided in natural language) are performed. Here, established techniques from the domain of natural language processing such as *typed dependencies* [4] are employed. Typed dependencies create a relationship between words of a sentence and, furthermore, type them with respect to their grammatical relation. For example, the words "the" and "service" have a *determiner* dependency (short: *det*) with each other. All typed dependencies of a sentence eventually form a graph based on which a translation into a formal representation can automatically be conducted

[2]Note that further issues such as particular doctors and their qualification as well as special equipment are not considered in the following.

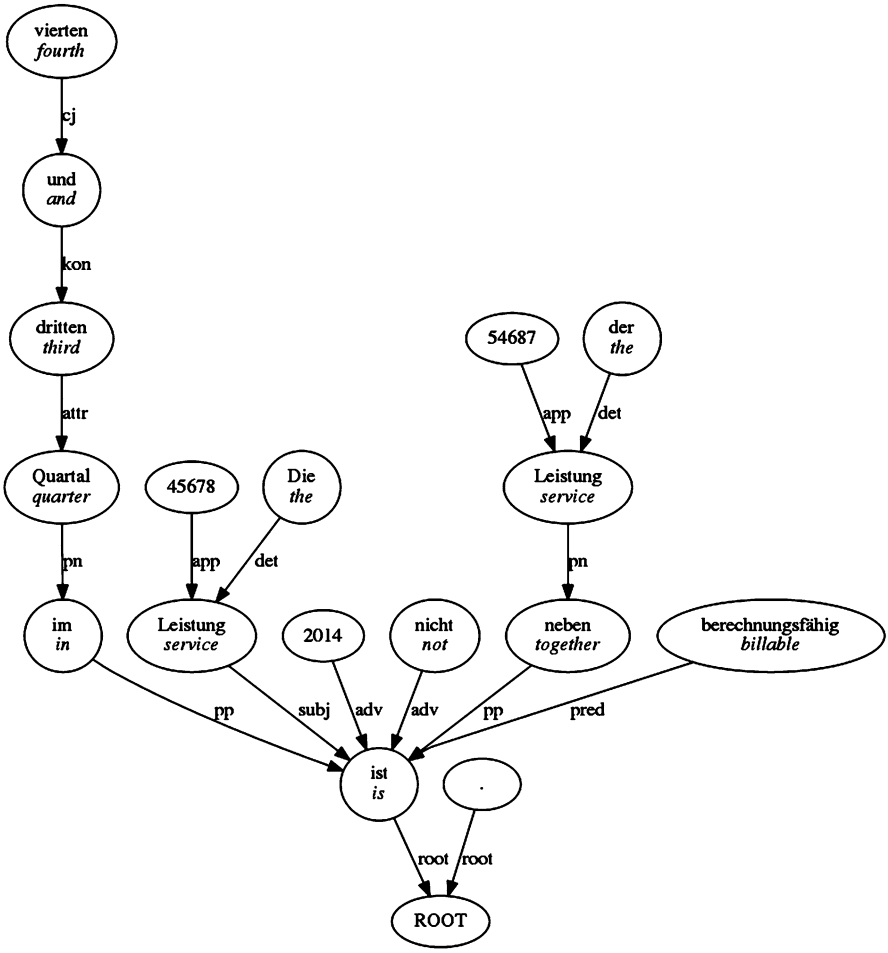


Fig. 6.3 Typed dependencies of the sentence shown in Fig. 6.2a

in many cases. The general idea is illustrated by means of the GOÄ rule of Fig. 6.2a. The corresponding typed dependency graph of the original German sentence, obtained using the tool ParZu [16], is shown in Fig. 6.3.

This analysis provides, e.g., the following information:

- The main reference, i.e. which service is of interest, is determined. This can be obtained by analysing the subject *subj* of the sentence.
- Other references to services might be found by following *pn-app* paths in the graph.
- A negation of the sentence is indicated by an *adv* path from the root note of the sentence to the word “not”.

```
context Patient inv:

-- fetch relevant quarters with invoices
let quarters =
       case.quarter.select(q| q.year=2014 and (q.n=3 or q.n=4))
in
  -- all services performed in these quarters
  let services = quarters.collect(service)
  in
    -- at most one service performed
    services.select(s | s.id=45678 or s.id=54687).size() <= 1
```

Fig. 6.4 Automatically generated formal notation of the sentence from Fig. 6.2a

Eventually, this enables an *automatic* derivations of a formal representation from a given legal regulation. In case of the considered sentence, a formal notation as sketched in Fig. 6.4 is created using OCL [18]. Note that other means of formal description are possible as well. Using the techniques described in [12], approximately 60 % of the regulations given in the GOÄ can automatically be formalized this way. The quality of the results heavily depends on the initial knowledge base used for determining synonyms and simplifying sentences.

6.4 Applying EDA Methods to the Formal Representation

As reviewed in the previous section, formal representations of legal regulations already exist and are applied, e.g. in order to check the validity of invoices. Similar application areas exist, e.g. in the domain of tax returns, applications to federal agencies in general, etc. However, all corresponding software tools basically check certain instances of invoices, tax returns, applications, etc., only. Exploiting the EDA methods reviewed in Sect. 6.2, much more potential exist. In fact, as for the design of circuits and systems, there is no guarantee that legal regulations are free of contradictions by themselves. The following example illustrates a possible problem:

Example 2. Consider the following four (simplified) GOÄ regulations:

- "Service *A* is not to be billed together with service *C* while, at the same time, it forces billing the service *B*".
- "Service *B* forces billing the service *C*".
- "Service *C* forces billing the service *D*".
- "Service *D* forces billing the service *A*".

Table 6.1a summarizes these regulations in a more compact fashion. The first column refers to the respective services, while the remaining columns provide the services which are mutually exclusive to them and the services which are forced to be billed in addition to them.

Table 6.1 Exemplary constraints for services

(a) Inconsistent model		
Service	Excludes	Forces
A	C	B
B	–	C
C	–	D
D	–	A
(b) Erroneous model		
Service	Excludes	Forces
A	C	B
B	–	C
C	A,B	–
D	C	–
(c) Example for invoice optimization		
Service	Costs $	Excludes
A	40	–
B	50	C
C	20	–
D	80	A,B

Obviously, this would be a set of contradictory legal regulations, since

- the billing of any service (*A,B,C*, or *D*) eventually forces the presence of all other services while,
- at the same time, service *A* forbids the existence of service *C*

While it is simple to detect contradictions for simple regulations as shown here, it surely becomes a crucial tasks when dozens or even hundreds of such regulations (usually written in hard language) have to be considered.

Using EDA methods, contradictions even in large (formalized) legal regulations could easily be detected. In fact, these methods do not care whether the corresponding formal representation has been derived from a hardware specification or a legal regulation. For them, legal regulations can be seen as a "specification" of how certain (real world) processes may be implemented in practice. As for circuits and systems, the goal of the methods is to detect the existence of contradictions or unwanted descriptions. While, in the domain of legal regulations, contradictions can not always be prevented (due to case-by-case decisions, trade-offs between certain rights, etc.), in particular in fields such as billings, tax returns, etc., a high degree on uniqueness is desired. In these cases, EDA methods may provide the basis for helpful tools to detect contradictions as discussed in Example 2. More precisely:

Example 3. Consider again the four GOÄ regulations from Example 2 and assume that corresponding formal representations (denoted by Booleans A, B, C, D) have been derived from them. Then, EDA methods allow to symbolically consider all possible applications scenarios and, by this, can proof, e.g. whether this set of

regulations is free of contradictions. In other words, the SAT problem "Does there exist at least one scenario which satisfies all regulations?" can be formulated. In a simplified form, this translates into the formulation

$$(A \implies \overline{C} \wedge B) \wedge (B \implies C) \wedge (C \implies D) \wedge (D \implies A)$$

for which SAT solvers can prove that no satisfying solutions exist. That is, not all regulations can be satisfied at the same time.

In a similar fashion, many other questions concerning legal regulations can be addressed by EDA methods. For example:

- *Checking for desired/undesired interpretations* of the given regulations. As an example, consider the regulations for services *A, B*, and *C* as shown in Table 6.1b. The intention of these regulations is to have two services excluding each other while services *A* and *B* force the billing of certain other services. Note that service *B* forces to also put service *C* on the invoice. Unfortunately, service *C* cannot be billed together with the other two services. This basically makes it impossible for service *B* to be ever billed. A formal model applied to an EDA method sketched above could find this inconsistency and present it to the corresponding authority. The authority then would probably decide that the intention was to have service *B* be mutually exclusive with service *C* and, therefore, move the *C* entry from the "Forces" to the "Excludes" column.

  This application is similar to the *reachability analysis* as reviewed in Sect. 6.2.
- *Fixing of certain properties.* Legal regulations are usually updated on a regular basis. Consider an update that introduces the constraint that service *A* must not be invoiced together with service *B*. When introducing this change (which, in real world documents, will not be as easily describable as this artificial example), one wants to ensure that it does not change any other rules. This means that no invoice that was legal before the update becomes invalid afterwards (while, at the same time, preserving the intended change). For checking this, a miter-like structure (see Sect. 6.2) can be employed. In this application, the "outputs" of interest are the valid invoices.
- *Automated optimization of invoices*, i.e. determine an application of regulations which leads to the best possible outcome for the doctor. As an example, consider the constraints as shown in Table 6.1c (which additionally provides the costs of a particular service). In this simple example, one can easily validate by hand that billing service *C*, and *D* yields the biggest outcome. For larger sets of regulations, this becomes an infeasible task for humans. A variant of this problem is to find the maximum amount of money that can be invoiced while staying within a certain budget.
- *Automated checks* whether a given invoice is consistent with the regulations. This helps the doctor as well as the officials deciding on the invoice at hand.
- *The automated generation of DSL expressions* (such as shown in Fig. 6.2c), which are used in software to check documents for validity. This does not only significantly reduce the development costs of software applications, but it

also ensures that the expressions are correct by eliminating the possibility of introducing errors in the creation/programming.

6.5 Conclusion

Formal descriptions of legal texts greatly improve the daily work for experts. This ranges from the direct end user of these regulations (such as a doctor as seen in the examples) to lawyers creating such texts. Applicants have the means to check whether their documents are valid before handing them in, thereby saving a lot of work on both sides. Decision makers such as politicians now have further means that aid them in understanding the full impact of their regulations. This is of great importance as it is difficult to completely anticipate the consequences of (parts of) regulations even for experts such as lawyers. We envision that in the near future, for most legal texts, formal descriptions will be available or work on formalizing them is in progress.

Acknowledgements The authors would like to thank Betina Keiner, Matthias Richter, Lucjan Suchy, and Gottfried Antpöhler for the many fruitful discussions. This work was supported by the German Federal Ministry for Economic Affairs and Energy (BMWi) under grant no. KF2054902MS2 and KF2013014MS2 as well as the German Research Foundation (DFG) under grant no. WI 3401/5-1.

References

1. Abdulla, P.A., Bjesse, P., Eén, N.: Symbolic reachability analysis based on SAT solvers. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 411–425. Springer, Berlin (2000)
2. Brand, D.: Verification of large synthesized designs. In: International Conference on CAD, pp. 534–537 (1993)
3. Cook, S.A.: The complexity of theorem-proving procedures. In: Symposium on Theory of Computing, pp. 151–158 (1971)
4. de Marneffe, M.C., MacCartney, B., Manning, C.D.: Generating typed dependency parses from phrase structure parses. In: Conference on Language Resources and Evaluation, pp. 449–454 (2006)
5. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340 (2008)
6. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: SAT 2003, LNCS, vol. 2919, pp. 502–518 (2004)
7. Eggersglüß, S., Drechsler, R.: Efficient data structures and methodologies for SAT-based ATPG providing high fault coverage in industrial application. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **30**(9), 1411–1415 (2011)
8. Gebührenordnung für Ärzte (GOÄ). Online available at http://www.e-bis.de/ (2014)
9. IEEE Standard VHDL Language Reference Manual Amendment 1: Procedural Language Application Interface (2007)
10. IEEE Standard for Standard SystemC Language Reference Manual (2012)

11. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language (2013)
12. Keszocze, O., Keiner, B., Richter, M., Antpöhler, G., Wille, R.: (Semi-)automatic translation of legal regulations to formal representations: expanding the horizon of EDA applications. In: Forum on Specification & Design Languages (FDL) (2014)
13. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Comput. 48(5), 506–521 (1999)
14. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Design Automation Conference, pp. 530–535 (2001)
15. Rumbaugh, J., Jacobson, I., Booch, G. (eds.): The Unified Modeling Language Reference Manual. Addison-Wesley Longman, Essex (1999)
16. Sennrich, R., Schneider, G., Volk, M., Warin, M.: A new hybrid dependency parser for German. In: Proceedings of the German Society for Computational Linguistics and Language Technology, pp. 115–124 (2009)
17. Soltana, G., Fourneret, E., Adedjouma, M., Sabetzadeh, M., Briand, L.: Using UML for modeling procedural legal rules: approach and a study of Luxembourg's tax law. In: Model-Driven Engineering Languages and Systems, pp. 450–466. Springer, Berlin (2014)
18. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison Wesley, New York (1999)
19. Weilkiens, T.: Systems Engineering with SysML/UML: Modeling, Analysis, Design. Morgan Kaufmann, San Francisco (2007)