Parametric Ontologies in Formal Software Engineering

Achim D. Brucker[a,1,*], Idir Ait-Sadoune[b], Nicolas Méric[c], Burkhart Wolff[c]

[a] *University of Exeter, Department of Computer Science, EX4 4RN, Exeter, UK*
[b] *Université Paris-Saclay, CentraleSupelec, LMF, Paris, 91190, France*
[c] *Université Paris-Saclay, LMF, Paris, 91190, France*

Abstract

Isabelle/DOF is an ontology framework on top of Isabelle/HOL. It allows for the formal development of ontologies and continuous conformity-checking of integrated documents, including the tracing of typed meta-data of documents. Isabelle/DOF deeply integrates into the Isabelle/HOL ecosystem, allowing to write documents containing (informal) text, executable code, (formal and semi-formal) definitions, and proofs. Users of Isabelle/DOF can either use HOL or one of the many formal methods that have been embedded into Isabelle/HOL to express formal parts of their documents.

In this paper, we extend Isabelle/DOF with annotations of λ-terms, a pervasive data-structure underlying Isabelle to syntactically represent expressions and formulas. We achieve this by using Higher-order Logic (HOL) itself for query-expressions and data-constraints (ontological invariants) executed via code-generation and reflection. Moreover, we add support for *parametric* ontological classes, thus exploiting HOL's polymorphic type system.

The benefits are: First, the HOL representation allows for flexible and efficient run-time checking of abstract properties of formal content under evolution. Second, it is possible to prove properties over generic ontological classes. We demonstrate these new features by a number of smaller ontologies from various domains and a case study using a substantial ontology for formal system development targeting certification according to CENELEC 50128.

Keywords: Ontologies, Generic Classes, Ontology Mapping, Formal Development, Formal Documents, Isabelle/HOL, Software Engineering, Certification

*Corresponding author
Email addresses: a.brucker@exeter.ac.uk (Achim D. Brucker), idir.aitsadoune@centralesupelec.fr (Idir Ait-Sadoune), nicolas.meric@universite-paris-saclay.fr (Nicolas Méric), burkhart.wolff@universite-paris-saclay.fr (Burkhart Wolff)

[1] This work was partly funded by UKRI as part of the Innovate UK project AutoCHERI.

1. Introduction

The linking of *formal* and *informal* information is perhaps the most pervasive challenge in the digitization of modern society. Extracting *knowledge* from reasonably well-structured "raw"-texts is a crucial prerequisite for any form of advanced search, classification, "semantic" validation and "semantic" merge technology. This challenge incites numerous research efforts summarized under the labels "semantic web" or "data mining". A key role in structuring this linking is played by ontologies (also called "vocabulary" in semantic web communities), i.e., a machine-readable form of the structure of documents and the document discourse. Such ontologies can be used for scientific discourse underlying scholarly articles, the conversion, and integration of semiformal content, for advanced semantic search in mathematical libraries or documentation in various domains. In other words, ontologies generate the meta-data necessary to annotate raw text allowing their "deeper analysis", in particular inside mathematical formulas or equivalent formal content such as programs or UML-models.

Throughout this paper, we will distinguish *document ontologies* from *domain ontologies*. The former are oriented towards meta-information used for the representation aspects of a target type-setting technology (e. g., LaTeX, HTML). The latter are oriented towards a specific knowledge domain. In this paper, we are particularly interested in domain ontologies concerning software developments targeting certifications (such as CENELEC 50128 [1] or Common Criteria [2]). Certifications of safety or security-critical systems, albeit responding to the fundamental need in the modern society of trustworthy numerical infrastructures are particularly complex and expensive. A major reason for this is that distributed labor requires that complex documents composed of artifacts from analysis, design, coding, and verification must preserve coherence under permanent changes. Moreover, certification processes impose a strong need of traceability within the global document structure. Last but not least, modifications and updates of a certified product usually result in a complete restart of the certification activity since the impact of local changes can usually not be mechanically checked and must be done by manual inspection. Our interest in this domain led us to the development of Isabelle/DOF (where DOF stands for Document Ontology Framework), an environment implementing our concept of *deep ontology*, i. e., ontologies represented inside a logical language such as HOL rather than a conventional programming language like Java.

1.1. Editing Annotated Documents

Isabelle/HOL [3] is a well-known semi-automated proof environment and documentation generator. Isabelle's Document Ontology Framework (Isabelle/DOF) [4] extends the Isabelle/HOL core (see Figure 1) by a number of constructs, allowing for the specification of formal ontologies (left-hand side); additionally, it provides documentation constructs (right-hand side) for text-, definition-, term-, proof-, code-, and user-defined elements that enforce document conformance to a given ontology.

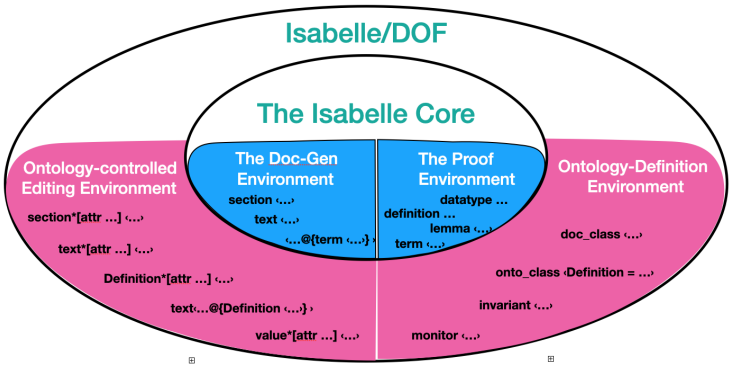


Figure 1: The Ontology Environment Isabelle/DOF.

Isabelle/DOF[2] is a new kind of ontological modelling and document validation tool. In contrast to conventional languages like OWL and development environments such as Protégé [6], it brings forward *deep ontologies* that generate strongly typed meta-information specified in HOL-theories allowing both efficient execution **and** logical reasoning about meta-data. They generate a particular form of checked annotations called *antiquotation* to be used inside code and texts. Deeply integrated into the Isabelle ecosystem [7], thus permitting continuous checking and validation, they also allow ontology-aware navigation inside large documents with both formal and informal content.

We will now detail this by example of annotated text in a document. We will assume a given ontology; an introduction to our ontology definition language ODL is given in section 3. Isabelle's document elements like **definition**‹ ... › for a HOL definition, **text**‹ ... ›- for a text or **ML**‹ ... › for code are extended to the corresponding Isabelle/DOF elements:

definition*[*label*::*cid*, *attrib_def*$_1$,... ,*attrib_def*$_n$]‹... *HOL−equation* ... ›
text*[*label*::*cid*, *attrib_def*$_1$,... ,*attrib_def*$_n$]‹... *annotated text* ... ›
ML*[*label*::*cid*, *attrib_def*$_1$,... ,*attrib_def*$_n$]‹... *annotated code* ... ›

where *cid* is an identifier of an ontological class introduced in an ontology together with attributes belonging to this class defined in ODL. For example, if an ontology provides a semiformal concept *Definition*, we can do the following:

text*[*safe*::*Definition*,name=safety*]‹*Our system is safe if the following holds...*›

The Isabelle/DOF command **text*** creates an instance *safe* of the ontological class *Definition* with the attribute *name* and associates it to the text inside the ‹...›-brackets. We call the content of these brackets the *text-context* (or

[2]The official releases are available in the Archive of Formal Proofs (AFP) [5], developer versions at `https://git.logicalhacking.com/Isabelle_DOF/Isabelle_DOF` and `https://zenodo.org/record/6810799`.

ML-context, respectively). Of particular interest for this paper is the ability to generate a kind of semantic macro called *antiquotation*, which is continuously checked and whose evaluation uses information from the background theory of this text element. We might refer to this definition in another text element:

text∗[**...**]‹*As stated in* @{*Definition* ‹*safe*›}, . . . ›

where Isabelle/DOF checks on-the-fly that the reference "safe" is indeed defined in the document and has the right type (it is not an *Example*, for example), generates navigation information (i.e. hyperlinks to *safe* and the ontological description of *Definition* in the Isabelle IDE) and specific documentation markup in the generated PDF document, e.g.:

As stated in Def. 3.11 (safety), ...

where the underline may be blue because the layout description configured for this ontology says so. Moreover, this is used to generate an index containing, for example, all definitions. Similarly, this also works for an ontology providing concepts such as "objectives", "claims" and "evidences", and invariants may be stated in an ontological class that finally enforces properties such as that "all claims correspond to evidences in the global document", and "all evidences must contain at least one proven theorem", etc. In contrast to a conventional typesetting system, Isabelle can additionally type-check formulas, so for example:

text∗[**...**]‹*The safety distance is defined by* @{*term* $dist_{safe} = sqrt(d{-}a{\cdot}(\Delta t)^2)$}...›

where functions like $dist_{safe}$, *sqrt*, ·, etc., have to be defined in the signature and logical context or background theory of this formula. Anti-quotations as such are not a new concept in Isabelle; the system comes with a couple of hand-programmed antiquotations like @{*term* ...}. In contrast, Isabelle/DOF *generates* antiquotations from ontological classes in ODL, together with checks generated from data-constraints (or: class invariants) specified in HOL.

1.2. Novelty No. 1: Term Contexts

Isabelle uses typed λ-terms as a syntactic presentation for expressions, formulas, definitions, and rules. Rather than using a classical programming language, our concept of deep ontologies led us to use HOL itself and generate the checking-code for antiquotations via reflection and reification techniques. In particular, this paves the way for a new context type called *term contexts*. As a consequence, DOF commands like:

definition∗[*t12*::*"formal_def"***...**] $inv_{safety1}$ :: *"σ ⇒ bool"*
where *"*$inv_{safety1}$ *σ ≡ (*$dist_{safe}$ *σ <* (@{*model_parameter* ‹*lab*›}*max_dist*)*)"***}**

or

theorem∗[*d15*::*"formal_def"***...**] *safety_preserving_1*
assumes *"*$inv_{safety1}$ *σ "* **and** *"*(@{*requirement* ‹*t5*›}a_{max} *< 20*::ℝ[$m{\cdot}s^{-2}$]*)"*
shows *"no_collision(system_transition σ)"* **proof ...**

can not only be referenced in ontological definitions via their *oid*'s (labels) *t12* and *d15*, they can also *contain* ontological references inside their formulas to entities such as the *model_parameter lab* and the *requirement t5* described formally or informally elsewhere.

1.3. Novelty No. 2: Polymorphism for Parametric Ontological Classes

It is a distinguishing feature of Isabelle/DOF that ontologies and meta-data are not just implemented in some implementation language such as SML (c.f. [7]). Rather, Isabelle/DOF **generates** a HOL theory defining types, terms, and definitions resulting from the definitions of an ontology. This allows for proving properties over ontology-generated meta-data and efficient execution via Isabelle's code-generator. Code-generation and reflection are the techniques used to execute validation checks generated from *class invariants*. They are also used for the execution of advanced queries.

It is, therefore, natural to take advantage of the advanced typing features of HOL in our ontology language ODL. Thus, it made it possible to declare in some classes an attribute *name* having some polymorphic type α, leaving it open how the *name* is represented.

1.4. Improvements over Earlier Work

This paper extends our earlier paper [8]: first, section 6 was added to introduce the use of polymorphism for parametric ontological classes. Second, we added a new application, i.e., the CENELEC 50128 ontology (section 7) and, third, an example showing the use of this ontology (section 8). Furthermore, we reflect on our experience of using Isabelle/DOF (section 9). Finally, we updated the content of the remaining sections and also added more details.

2. Background

2.1. Isabelle and HOL

While still widely perceived solely as an interactive theorem proving environment, Isabelle [3] has nowadays become a flexible system framework providing an infrastructure for components. This comprises extensible state management, extensible syntax, code-generation and the advanced documentation support we built upon in Isabelle/DOF. Isabelle/HOL is a kind of component offering support for Higher-order logic (HOL), a logical language similar to functional programming languages, extended by automated proof techniques and provides feedback via animations such as hyperlinks and popup windows.

Isabelle possesses an order-sorted polymorphic type system [9, 10] (see [11], pp 250, for an introduction). This means that we can not only express genericity on the level of meta-data and its relations, but also that type-variables α, β, γ,

... can be constrained in a judgement $\alpha :: C$ where C is a *sort*.[3] The judgement α::{C_1,...,C_n} is a compact form of the conjunction $\alpha::C_1 \wedge \ldots \wedge \alpha::C_n$. Alternatively, we may think of α::{C_1,...,C_n} as a notation for $C_1 \cap ... \cap C_n$, the intersection of the types belonging to the sorts C_1 to C_n. Sorts can be understood as an abstract interface to types, imposing operations and properties over them. For example, the sort constraint α::*order* may constrain the possible instances of α to those who possess an ordering operation $ord :: \alpha \Rightarrow \alpha \Rightarrow bool$ which must satisfy the properties of reflexivity, transitivity and anti-symmetry. When declaring that a concrete type, say *int*, is an instance of *order*, which is to say that the judgement *int*::*order* holds, this results in the obligation to define *ord* by a concrete operation, say $\_\leq\_ :: int \Rightarrow int \Rightarrow bool$, and proof obligations that this definition satisfies the required properties on *order*-types. Thus, it is possible to define once and for all the operation $sort :: (\alpha::order)\ list \Rightarrow (\alpha::order)\ list$, prove a bunch of theorems over it, and inherit them for the special case $sort :: int\ list \Rightarrow int\ list$, without proving them again.

2.2. The Isabelle/DOF Framework

As explained earlier, Isabelle/DOF extends Isabelle/HOL (recall Figure 1) by ways to *annotate* an integrated document with meta-data specified in an own Ontology Definition Language (ODL). Isabelle/DOF generates from an ODL ontology a family of *antiquotations* allowing to specify machine-checked links between annotated entities.

The perhaps most attractive aspect of Isabelle/DOF is its deep integration into the IDE of Isabelle (Isabelle/PIDE), which allows hypertext-like navigation and fast user-feedback during the development and evolution of the integrated document source. This includes rich editing support, namely on-the-fly semantics checks, hinting, or auto-completion. Isabelle/DOF supports LaTeX-based document generation and ontology-aware "views" on the integrated document, i. e., specific versions of generated PDF addressing, e. g., different stake-holders.

3. A Guided Tour through ODL

Isabelle/DOF provides a strongly typed ODL that provides the usual concepts of ontologies, such as

- *document class* or *ontological class* (using the **doc-class** or **onto-class** keyword, respectively),
- *attributes* specific to classes (attributes might be initialized with default values), and

[3] Throughout this paper, rather than speaking of "type-classes" and "type-class polymorphism", we will use the more ancient equivalent terms "sorts" and "order-sorted polymorphism" in order to avoid confusion with "ontological classes" as introduced in ODL.

Isabelle

```
doc-class "title" = short_title :: "string option" <= "None"
doc-class affiliation =
  journal_style :: 'α
doc-class author =
    affiliations :: "'α affiliation list"
    name         :: string
    email        :: "string"   <= "''''"
    invariant ne_name :: "name σ ≠ ''''"
doc-class "text_element" =
    authored_by :: "('α author) set"  <= "{}"
    level       :: "int  option"  <= "None"
    invariant authors_req :: "authored_by ≠ {}"
    and       level_req   :: "the (level) > 1"
doc-class "introduction" = text_element +
    authored_by :: "('α author) set"  <= "UNIV"
doc-class "technical" = text_element +
    formal_results  :: "thm list"
doc-class 'α "example" = "'α technical" +
    title :: "string"
doc-class "definition" = technical +
    is_formal   :: "bool"
doc-class "theorem" = technical +
    assumptions :: "term list"        <= "[]"
    statement   :: "term option"      <= "None"
doc-class "conclusion" = text_element +
    wrapup      :: "(definition set × theorem set)"
    invariant (∀ x∈fst wrapup. is_formal x)
              ⟶ (∃ y∈snd wrapup. is_formal y)
doc-class "article" =
    style_id    :: string   <= "''LNCS''"
    accepts "(title ~~ {|author|}+ ~~ {|introduction|}+
          ~~ {|{|definition ~~ example|}+ || theorem|}+ ~~ {|conclusion|}+)"
```

Figure 2: A Basic Document Ontology: paper^m

- a special link, the reference to a super-class, establishes an *is-a* relation between classes.

ODL can be used to specify both document and domain ontologies; in the sequel, we will discuss the example $paper^m$ in Figure 2 for the former and $cenelec^m$ in Figure 3 for the latter.[4]

The ontology $paper^m$ introduces document classes which are typical for the structure of a mathematical paper framing the canonical sequence "definition-theorem-proof", where entities are not necessarily formal.[5] Attributes like *short_title* were typed with HOL types from the Isabelle library, and default values like *None* for class-instances can be declared. ODL can refer to any predefined type from the HOL library, e. g., *string*, *int* and parameterized types, e. g., *option*, *list*. Isabelle/DOF also supports polymorphic variables in these types in order to support class schemata. For example, in *affiliation*, the precise format specification is left open due to the fact that publishers like Elsevier or ACM have very different requirements to represent them; thus, polymorphism is a means to increase reusability by abstraction. Semantically, classes have a strong similarity to HOL's *extensible records* [11] used to represent them logically. Classes can be equipped with invariants allowing to formalize meta-data-constraints in HOL. At the same time, *accepts*-clauses can specify in a regular expression language over class-id's the textual order in which instances may appear within an article.

In contrast to general handling of polymorphic variables, Isabelle/DOF is equipped with notational mechanisms to manage the propagation of polymorphic variables implicitly. This is due to the fact that:

1. Isabelle/DOF uses polymorphism to implement single-inheritance induced sub-typing (hiding the polymorphism of the "more-fields" in extensible records [11]). These type of variables appear so often in medium-size ontologies, and reveal so many technical details of the Isabelle/HOL implementation, that their management by hand appears tedious.
2. Isabelle/DOF uses polymorphism to implement true order-sorted polymorphism for the parameterization of attribute types. Again, their use in class-types is common, and the resulting lists of type-variables for class types may be daunting and difficult to maintain. In both cases, we preferred to hide these subtleties from the users of ontologies, deferring them to specialized query functions that reveal them. In proofs, these details become explicit, of course.

The types of attributes are HOL-types, and concrete values for attributes are just HOL-terms, i.e., λ-terms over the signature of imported HOL-theories. As

[4]Both are fragments of the ontologies `Isabelle_DOF.scholarly_paper` ($paper^m$) and `Isabelle_DOF.CENELEC_50128` ($cenelec^m$), which are included in the Isabelle/DOF distribution. The size of the former is about 400 loc, the latter about 1200 loc, plus the corresponding LaTeX configurations of about the same size.

[5]Proofs are abstractly represented in the $cenelec^m$ example as *result* classes whose instances will have an *evidence* attribute set to the *proof kind*.

a consequence of the logical representation, for each class, there will be a HOL type for the instances of a class. Therefore, class definitions allow for formal *links* to and between ontological concepts, as the use of the *author* class inside the *introduction* class definition, for example. The underlying Isabelle/DOF theory provides types for Isabelle types, terms, and theorems and specific means to denote them when referring to them in meta-data.

As for the domain ontology fragment $cenelec^m$ shown in Figure 3, the definition proceeds as an extension of the $paper^m$ ontology providing elements such as introductions, conclusions, formal and informal definition elements, etc. Note that $cenelec^m$ is a very condensed version of some aspects of *one* artifact out of 27 in CENELEC EN 50128, the *software requirements specification* (SRS).

Since ODL specification elements are just another kind of command in Isabelle, they can be arbitrarily mixed with standard HOL specification constructs like inductive datatype definitions—in our example, the enumeration for *role*'s, a simplified version of CENELEC's requirement to enforce a separation of author groups in a process. The Isabelle/DOF command **class-synonym** introduces equivalent names for classes; it also generates a **type-synonym** for the types induced by the ODL class, but is aware of the implicit type variables.

Note that the concept of *definition* appears in both ontologies. This is a consequence of the fact that this entity and similar common rhetoric constructions, like *assumption* or *consequence*, appear in many domains with different meanings and document representations; a mathematician may have a different understanding of these terms than a lawyer or an engineer. ODL supports namespaces that allow for a separation of these.

3.1. The Effect of ODL Specifications in Isabelle/DOF

Recall that the purpose of ODL inside the Isabelle platform is to generate support for navigation and on-the-fly validation of antiquotations *plus* a meta-theory. For example, from the class-ids *definition*, *EC* or *SRAC* of $cenelec^m$, Isabelle/DOF generates text-, term-, and code-antiquotations of the form

$$@\{\textit{“definition”}\ \langle \textit{oid} \rangle\}\ \ \ldots\ \ @\{\textit{EC}\ \langle \textit{oid} \rangle\}\ \ldots\ @\{\textit{SRAC}\ \langle \textit{oid} \rangle\}$$

where *oid* is some reference to some class instance. As text-antiquotations, they produce configurable output and checks.

Figure 4 shows an ontological annotation of a requirement and its referencing via an antiquotation $@\{\textit{requirement}\ \langle \textit{req1} \rangle\}$; the latter is generated from the above class definition. Undefined or ill-typed references are rejected, and the high-lighting displays the hyperlinking which is activated with a click. The class-definition of *requirement* and its documentation is also just a click away.

As Isabelle/DOF is based on the idea of “deep ontologies”, a logical representation for the instance *req1* is generated, i. e. a λ-term, which is used to *represent* this meta-data. For this purpose, we use Isabelle/HOL's record support [11].

For the above example, this means that *req1* is represented by:

Isabelle

imports *paper*^{m}
datatype *role* = *developer* | *verifier* | *validator*

onto-class *requirement* = *text_element* +
long_name ::"*string option*"
is_concerned::"*role set*"

onto-class *SR* = *requirement* +
class-synonym *safety_requirement* = *SR*
onto-class *FR* = *requirement* + **...**
class-synonym *functional_requirement* = *FR*

onto-class "*definition*" = *text_element* + **...** — terminological definition.

onto-class *assumption* = *text_element* + **...**

onto-class *ASS* = *assumption* + **...**
class-synonym *application_constraint* = *ASS*
onto-class *EC* = *assumption* + **...**
class-synonym *exported_constraint* = *EC*
onto-class *SRAC* = *exported_constraint* + **...**
class-synonym *safety_related_application_constraint* = *SRAC*

onto-class *model_element* = **...**

doc-class *SRS* = **...**
accepts "(⦃*requirement*⦄$^+$ ~~ ⦃*assumption*⦄$^+$ ~~ ⦃*model_element*⦄$^+$)"
class-synonym *software_requirements_specification* = *SRS*

datatype *kind* = *expert_opinion* | *argument* | "*proof*"

onto-class *result* = *technical* +
evidence :: *kind*
property :: "'α *theorem list*" <= "[]"
invariant *has_property* :: "*evidence* = *proof* ⟷ *property* ≠ []"

Figure 3: A Basic Domain Ontology: cenelec^m

```
text*[req1::requirement,
      is_concerned="{developer, validator}"]
‹The operating system shall provide secure
memory separation.›
```

(a) A Text-Element as Requirement.

```
text‹
The recurring issue of the certification
is @{requirement ‹req1›} ...›
```

(b) Referencing a Requirement.

Figure 4: Referencing a Requirement.

- the record term $(\!|long\_name = None,\ is\_concerned = \{developer, validator\}|\!)$ and the corresponding record type $requirement = (\!|long\_name{::}string\ option,\ is\_concerned{::}role\ set|\!)$,
- while the resulting selectors are written $long\_name\ r$, $is\_concerned\ r$.

In general, **onto-class**es and the logically equivalent **doc-class**es are represented by *extensible* record types and instances thereof by HOL terms (see [7] for details).

3.2. Term-Evaluations in Isabelle

Besides the powerful, but relatively slow rewriting-based proof method *simp*, there are basically two other techniques for the evaluation of terms:

- evaluation via reflection into SML [12] (*eval*), and
- normalization by evaluation [13] (*nbe*).

The former is based on a nearly one-to-one compilation of datatype specification constructs and recursive function definitions into SML datatypes and functions. The generated code is directly compiled by the underlying SML compiler of the Isabelle platform. This way, pattern-matching becomes natively compiled rather than interpreted as in the matching process of *simp*. Aehlig et al [13] are reporting on scenarios where *eval* is five orders of magnitude faster than *simp*. In special applications, e.g., the verification of security protocols, *eval* can reduce the running time from several hours to a few seconds, compared to *simp* [14]. However, *eval* is essentially restricted to ground terms. Note that *nbe* is not restricted to ground terms, its efficiency lying between *simp* and *eval*.

Isabelle/DOF uses a combination of these three techniques in order to evaluate invariants and check data-integrity on the fly during editing. For reasonably designed background theories and ontologies, this form of runtime-testing is sufficiently fast to remain unnoticed by the user.

3.3. The Previewer

A screenshot of the editing environment is shown in Figure 5. It supports incremental continuous PDF generation which improves usability. Currently, the granularity is restricted to entire theories (which have to be selected in a specific document pane). The response times can not (yet) compete with a Word- or Overleaf editor, though, which is mostly due to the checking and evaluation overhead (the processing time for the theory file containing the first three section of this paper is around 30s). However, we believe that better parallelization and evaluation techniques will decrease this gap substantially for the most common cases in future versions.

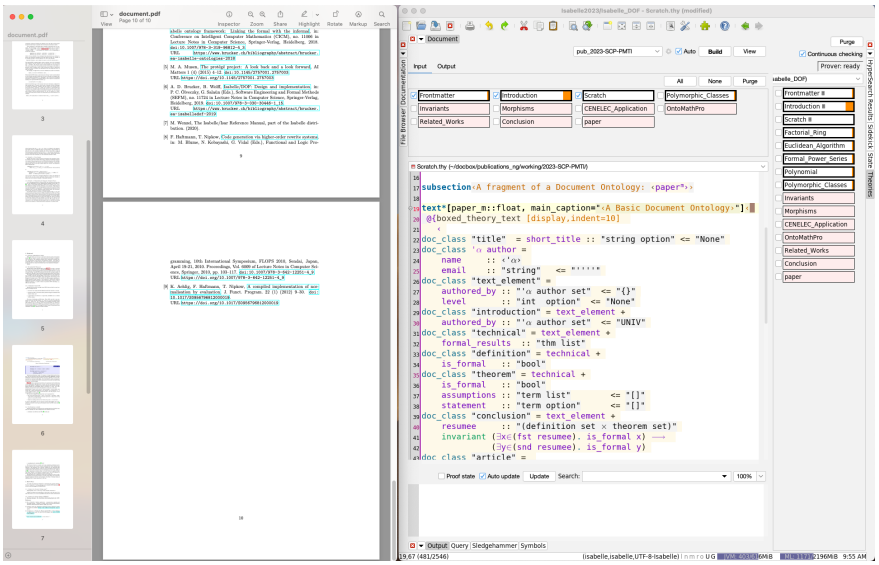


Figure 5: A Screenshot while editing this Paper in Isabelle/DOF with Preview.

4. Term-Context Support, Invariants, and Queries in DOF

As Isabelle/HOL offers a document-centric view to the *formal* theory development process, this led to strong documentation generation mechanisms over the years. A particular feature is the support of built-in text and code antiquotations. As mentioned earlier, Isabelle/DOF generates from ODL classes antiquotations for text and code contexts [4, 7]. In this section, we introduce the novel concept of *term contexts*, i. e., annotations to be made inside λ-terms.

For example, we consider the Isabelle/DOF commands **term**$*$ and **value**$*$ (replacing the traditional **term** and **value**); they parse and type-check respectively compile and execute a λ-term:

Isabelle

```
term*‹ @{thm ''HOL.refl''} = @{thm ''HOL.sym''} ›
value*‹ @{thm ''HOL.refl''} = @{thm ''HOL.sym''} ›
```

It is instructive to see what is happening here. Terms comprising term antiquotations are treated by a refined process involving the steps:

- *parsing* and *typechecking* of the term in HOL theory context,
- ontological *validation* of the term:
  - the arguments of term antiquotations are parsed and checked,
  - checks resulting from ontological invariants are applied,
- *generation of markup information* for the navigation in the IDE,

```
text*[church::author, email="‹church@lambda.org›"]‹... text›

text*[intro1::introduction, authored_by="{@{author ‹church›}}",
                            level="Some 0"]‹... text›

text*[safety::"theorem", assumptions = "[@{term ‹s = t›}]",
                         statement = "Some @{term ‹t = s›}"]‹... text›
text*[security::"theorem", assumptions = "[@{term ‹t = s›}]",
                         statement = "Some @{term ‹P s ⟹ P t›}"]‹... text›

text*[proof1::"'a result", evidence = "proof",
                    property="[@{theorem ‹safety›}, @{theorem ‹security›}]",
                        level = "Some 2",
                        authored_by = "{@{author ‹church›}}"]‹... text›
text*[proof2::"'a result", evidence = "proof",
                    property="[@{theorem ‹security›}, @{theorem ‹safety›}]",
                        level = "Some 2",
                        authored_by = "{@{author ‹church›}}"]‹... text›
```

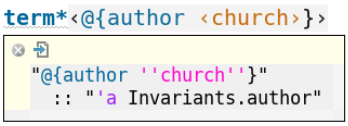
Isabelle

Figure 6: Some Instances referring to Figure 2.

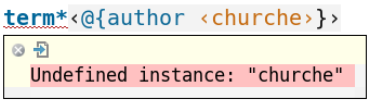
- *elaboration* of term antiquotations: depending on the antiquotation specific elaboration function, the antiquotations containing references are replaced by the object they refer to, and
- *evaluation*: HOL expressions are compiled and the result executed.

Here, **term**∗ parses and type-checks this λ-term as usual; logically, the @{*thm* ''*HOL.refl*''} is predefined by Isabelle/DOF as a constant *ISA_thm*. The validation will check that the string ''*HOL.refl*'' is indeed a reference to the theorem in the HOL-library, notably the reflexivity axiom. The type-checking of **term**∗ will infer *bool* for this expression. Now, **value**∗ will replace it by a constant representing a symbolic reference to a theorem; code-evaluation will compute *False* for this command. Note that this represents a kind of referential equality, not a "very deep" ontological look into the proof objects (in our standard configuration of Isabelle/DOF). There is a variant of **value**∗, called **assert**∗, which explicitly fails (stops evaluation) if the evaluation yields *False*.

Some class instances of the $paper^m$ and $cenelec^m$ ontologies can be defined with the **text**∗ command, as in Figure 6. In the instance *intro1*, the term antiquotation @{*author* ‹*church*›}, or its equivalent notation @{*author* ''*church*''}, denotes the instance *church* of the class *author*, where *church* is a HOL-string referring to an author text-element in the global context. One can also reference a class instance in a **term**∗ command as a term antiquotation. In the command **term**∗‹@{*author* ‹*church*›}› the term @{*author* ‹*church*›} is



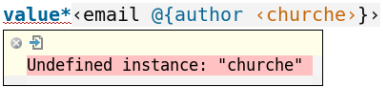
(a) Here, *church* is an existing Instance.



(b) The Instance *churche* is not defined.

Figure 7: Type-Checking of Antiquotations in a Term-Context.

(a) The Evaluation succeeds.



(b) The Evaluation fails.

Figure 8: Evaluation of Antiquotations in a Term-Context.

type-checked (see Figure 7).

The command **value**∗‹*email* @{*author* ‹*church*›}› validates @{*author* ‹*church*›} and returns the attribute-value of *email* for the *church* instance, i. e. the HOL-string *''church@lambda.org''* (see Figure 8).

Since term antiquotations are basically uninterpreted constants, it is possible to compare class instances logically. The assertion in the Figure 9 fails: the *property* attribute of class instances *proof1* and *proof2* is not equivalent because the lists sorting differs. When **assert**∗ evaluates the term, the term antiquotations @{*theorem* ‹*safety*›} and @{*theorem* ‹*security*›} are checked against the global context such that the strings ‹*safety*› and ‹*security*› denote existing *theorem* class instances.

The mechanism of term annotations is also used for the new concept of invariant constraints which can be specified in common HOL syntax. They were introduced by the keyword **invariant** in a class definition (recall Figure 2). The *authors_req* **invariant** of the class *text_element* enforces that a *text_element* instance has at least one author. Following the constraints proposed in [4], one can specify that any instance of a *result* class finally has a non-empty property list, if its *kind* is *proof* (see the *has_property* **invariant**). The *is_form* **invariant** specifies the relation between the sets of *definition* and *theorem* document elements for the *wrapup* attribute of the conclusion class and forces a *theorem* to be tagged as formal if its related *definition* also is. By relying on the implementation of extensible records in Isabelle/HOL [11], one can reference an instance attribute using its selector function. For example, in the *is_form*

Figure 9: Evaluation of an Attribute of two Class Instances.

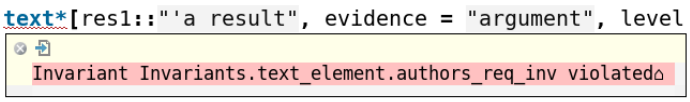


Figure 10: Inherited Invariant Violation.

invariant, *wrapup* denotes the *wrapup* attribute-value of the future *conclusion* class instance.

The value of each attribute defined for the instances is checked at run-time against their class invariants. Recall that classes also inherit the invariants from their super-classes. As the *result* class is a subclass of the *text_element* class, it inherits its invariants. In Figure 10, we attempt to specify a new instance *res1* of this class. However, the invariant checking triggers an error because the *authors_req* **invariant** forces the *authored_by* attribute to be a non-empty set and as its value was not set in the *res1* instance definition, *res1* inherits the default value from the *text_element* class which is the empty set.

Any class definition generates term antiquotations checking a class instance reference in a particular logical context; these references were elaborated to objects they refer to. This paves the way for a new mechanism to query the "current" instances presented as a HOL *list*. Using functions defined in HOL, arbitrarily complex queries can therefore be defined inside the logical language. Thus, to get the property list of the *result* class instances, it suffices to process this meta-data via mapping the *property* selector over the *result* class:

```
value*‹map (property) @{instances_of ‹result›}›
```
Isabelle

Analogously we can define an arbitrary filter function, for example the HOL *filter* definition on lists:

```
fun filter:: "('a ⇒ bool) ⇒ 'a list ⇒ 'a list"
  where "filter P [] = []"
      | "filter P (x # xs) = (if P x then x # filter P xs else filter P xs)"
```
Isabelle

to get the list of the *result* class instances whose *evidence* is a *proof*:

```
value*‹filter (λσ. evidence σ = proof) @{instances_of ‹result›}›
```
Isabelle

With Isabelle/DOF comes the concept of monitor classes [7], which are classes that may refer to other classes via a regular expression in an *accepts* clause. Semantically, monitors introduce a behavioral element into ODL and enforce the structure in a document. Monitors generate traces about a part of a document, recorded in the *trace* attribute of the monitor, and also presented as a *list* of *string*. Monitors are regular expressions composed of options, sequences, and repetitions. In the following monitor specification:

```
doc-class doc_monitor =                                                    Isabelle
  ok :: "unit"
  accepts "⟦introduction⟧ ~~ {|result|}+ ~~ ⟦conclusion⟧"
```

the *accepts* clause enforces the document structure to be a sequence of an introduction, one or more results and a conclusion. Then, one can define an *is−in* function in HOL to check the trace of a document fragment against a regular expression:

```
definition example_expression                                              Isabelle
  where "example_expression ≡ {|introduction || result || conclusion|}* "

  value*‹ (map fst @{trace_attribute ''monitor1''}) is−in example_expression
›
```

Here, the term antiquotation @{*trace_attribute ''monitor1''*} denotes the instance trace of *monitor1*. It is checked against the regular expression *example_expression*, which matches zero or more occurrences of either an introduction, a result, or a conclusion. Actually, *example_expression* is compiled via an implementation of the Functional-Automata of the AFP [15] into a deterministic automaton. On the latter, the above acceptance test is still reasonably fast.

5. Proving Morphisms on Ontologies

The Isabelle/DOF framework does not assume that all documents refer to the same ontology. Each document may even build its local ontology without any external reference. It may also be based on several reference ontologies (e. g., from the Isabelle/DOF library). Making a relationship between a local ontology and reference ontologies is a way to express that the content referencing a local ontology is not "far away" from a domain reference ontology.

Since ontological instances possess *representations inside the logic*, the relationship between a local ontology and a reference ontology can be represented by a formalized morphism. More precisely, the instances of local ontology classes may be mapped via conversion functions to one or several other instances belonging to another ontology. Since an instance representation as well as the conversion functions are constructed inside HOL, it is possible to prove formally once and for all that the morphism preserves the invariants for all meta-data. This means that morphisms may provably be injective, surjective, bijective, and thus describe abstract relations between ontologies.

To illustrate invariance preservation of a morphism, we zoom into the $paper^m$ example where *author*s for specific journals were defined. The example addresses the common problem that publishers require slightly different meta-data which might be a nuisance for an author when addressing a paper to a different journal. We specialize authors in the following:

```
doc-class acm_author = "acm author" +                                    Isabelle
  orcid :: int
  footnote :: string
doc-class elsevier_author = "elsevier author" +
  short_author :: string
  url          :: string
  footnote     :: string
```

Each class inherits the *affiliations* attribute from the *author* class and defines a list of *affiliation*s as specified by the journal. In our example, *elsevier_author* and *acm_author* implement the specification of an Elsevier article or of an ACM article respectively. Also, each class inherits the *author name* attribute and the *ne_name* invariant that enforces its *name* to be non-empty.

Now we refine the concept of *author* in a local ontology:

```
doc-class "title" = short_title :: "string option" <= "None"             Isabelle
doc-class affiliation =
   journal_style :: 'α
doc-class author =
   affiliations :: "'α affiliation list"
   firstname    :: string
   surname      :: string
   email        :: "string"  <= """"
   invariant ne_fsnames :: "firstname σ ≠ '''' ∧ surname σ ≠ ''''"
doc-class elsevier_author = "elsevier author" +
  short_author :: string
  url          :: string
  footnote     :: string
```

As a local ontology, it may have different meanings and document representations when compared to $paper^m$, which "live" together in the same document but in different name-spaces. This ontology also defines a local *elsevier_author* class that implements an Elsevier article author. It is a subclass of the local *author* class and inherits its *firstname* and *surname* attributes. It also inherits the *ne_fsnames* invariant that requires that *firstname* and *surname* are non-empty.

Using this ontology we are now able to update Elsevier article authors from the local ontology to ACM article authors from the reference ontology, for example. And if all the specification of an Elsevier article were to be defined in our local ontology, we would be able to convert the meta-data of an entire Elsevier article to an ACM article. To update an Elsevier article author, we define a relationship between the local ontology and the $paper^m$ ontology using conversion functions (also called *mapping rules* in the ATL framework [16] or in the EXPRESS-X language [17]). These rules are applied to define the relationship between one class of the local ontology to one or several other class(es) described in our $paper^m$ ontology. In our case, our morphism is represented by three conversion functions, addressing the conversion of base-data, the affiliation and finally the authors. The base-data conversion of an enumeration type (not shown here for economy of space) is defined as follows:

```
definition elsevier_to_acm_morphism :: "elsevier ⇒ acm"                    Isabelle
                                    ("_ ⟨acm⟩elsevier" [1000]999)
 where "σ ⟨acm⟩elsevier =  (| tag_attribute = 1::int,
       position = ''no position'', institution = organization σ, department = 0,
       street_address = address_line σ, city = elsevier.city σ, state = 0,
       country = ''no country'', postcode = elsevier.postcode σ  |)"
```

The more high-level conversions concerning the affiliation is detailed as:

```
definition                                                                 Isabelle
elsevier_aff_to_acm_aff_morphism :: "elsevier affiliation ⇒ acm affiliation"
                                    ("_ ⟨acm'_aff⟩elsevier'_aff" [1000]999)
  where "σ ⟨acm_aff⟩elsevier_aff = (| tag_attribute = 1::int,
        journal_style = (affiliation.journal_style σ) |> (λx. x ⟨acm⟩elsevier) |)"
```

where (|>) is simply a reverse application combinator. The top-level conversion for the author looks as follows:

```
definition elsevier_author_to_acm_author_morphism                          Isabelle
           :: "elsevier_author ⇒ acm_author"
              ("_ ⟨acm'_auth⟩elsevier'_auth" [1000]999)
 where "σ ⟨acm_auth⟩elsevier_auth = (| tag_attribute = 1::int,
                          affiliations = (author.affiliations σ)
                                         |> map (λx. x ⟨acm_aff⟩elsevier_aff) ,
                          name = acm_name (firstname σ) (surname σ),
                          email = author.email σ, orcid = 0,
                          footnote = elsevier_author.footnote σ  |)"
```

These definitions specify how *affiliation* and *elsevier_author* metadata representations are mapped to *affiliation* and *acm_author* objects as defined in $paper^m$. The *acm_author name* attribute-value is derived from the *elsevier_author firstname* and *surname* attributes using a parsing function *acm_name* that follows the ACM journal author specification. This mapping shows that the structure of a local (user) ontology may be arbitrarily different from the one of a standard ontology it refers to.

In order to support morphisms, we implemented a high-level syntax for this:

onto-morphism (*elsevier_author*) **to** *acm_author* **..**

where the ".." stands for a standard proof attempt consisting of unfolding the invariant predicates and a standard *auto* proof. This syntax also covers more general cases such as :

onto-morphism (A_1, **...**, A_n) **to** X_i **and** (D_1, **...**, D_m) **to** Y_j

were tuples of instances belonging to classes (A_1, ..., A_n) can be mapped to instances of another ontology.

After defining the mapping rules, we have to deal with the question of invariant preservation. The following nearly trivial proof for a simple but typical example is shown below:

Isabelle

```
lemma elsevier_inv_preserved :
  "ne_fsnames_inv σ ⟹ ne_name_inv (σ ⟨acm_auth⟩elsevier—auth)"
  unfolding ne_fsnames_inv_def ne_name_inv_def
            elsevier_author_to_acm_author_morphism_def
  by (simp add: combinator1_def acm_name_def)
```

After unfolding the invariant and the morphism definitions, the preservation proof is automatic. The advantage of using the Isabelle/DOF framework compared to approaches like ATL or EXPRESS-X is the possibility of formally verifying the *mapping rules*, i. e., proving the preservation of invariants once and for all rather than converting data and then relying on a post-hoc check.

6. Order-sorted Polymorphic Classes for Ontologies

The overall objective of this work is to express the machine-checkable linking between formal and informal content in documents. In database communities, algebraic structures have been proposed to express a particular form of this linking, the *provenance* of data, used in results of queries or virtual tables (cf. [18, 19, 20]) and among them polynomials [21]. Adapted to integrated documents, data provenance describes the history of a document element, for example the origin of a formal definition by declaring its dependency on other definitions expressed informally, opening new workflows for the validation of document specification targeting a certification standard. Recall that a multivariate polynomial structure (A,X,$*$,$+$) over a set of coefficients A and a set of indeterminates X forms with respect to the multiplicative and additive operations a semi-ring. As a consequence, polynomials such as:

$$P(x) = a_n * x^n + ... + a_1 * x^1 + a_0$$

or even more generally:

$$P'(x_1...x_m) = \sum a_{i_1 i_2 ... i_m} x_1^{i_1} x_2^{i_2} ... x_m^{i_m}$$

can be represented in this structure and will have a normal form which permits their comparison via partial orders. Moreover, multivariate polynomials are substitutive, i. e. an equality like $x_1 = P(x_2)$ can be used to eliminate the variable x_1 in P' and to compute again a normal form. A set of such equalities allows therefore to reduce multivariate polynomials to one based only on a subset of base multivariates.

We will mostly use $\mathbb{Z}$ or $\mathbb{N}$ for the coefficients A; the indeterminates can be interpreted by *oid*'s or arbitrary labels used to define groups over them. An interpretation of the coefficients $\mathbb{B}$ and the $(*)$-operator by the logical conjunction $(\wedge)$ and the $(+)$ by the disjunction $(\vee)$ leads to a collapse of the polynomials to disjunctive normal-forms (DNF) which can be interpreted by "this concept depends on the concept (x_1 and x_2) or x_3", for example. More general interpretations than the latter allow for expressing weights on these dependencies.

6.1. Modeling "Provenance" in ODL

Isabelle/DOF's support of order-sorted polymorphism opens a new path to use multivariate polynomials to express the linking between a document element and others where the indeterminates $x_1^{i_1}, x_2^{i_2}, \ldots x_m^{i_m}$ will capture some kind of document element references, the coefficients $a_{i_1 i_2 \ldots i_m}$ the "linking flavor" of each monomial, and the exponents $i_1, i_2, \ldots i_m$ will capture a specific "weight" of a reference (an indeterminate) within each monomial.

By leaving open the concrete computational structure for a multivariate polynomial (A,X,$*$,$+$) at the moment of the creation of a document element, we can express various forms of the linking within a single class attribute and postpone the decision of the concrete computational structure to a later stage, for example, at the point where a concrete query is formulated. Views on documents can be represented to draw different paths that lead to the same formal definition, offering alternatives for groups of validation with distinct domain expertise and help in a certification process by embedding in the same document specifications expressed in several but equivalent ways.

This constitutes a new form of abstract representation of dependency using an order sort, possibly reinforced by class invariants. Using order-sorted polymorphism, we construct an executable type for multivariate polynomial structures (A,X,$*$,$+$):

$$('\nu{::}linorder,\ '\alpha{::}semiring)\ mpoly$$

where $'\nu$ corresponds to the set of multivariates (assumed to be orderable for reasons of normal-form computations) and where $'\alpha$ corresponds to the coefficients. On a true *semiring* structure this type captures precisely the notion of *how-provenance* [21] and tracks how document elements relate to each other along the declaration of the instances. Note that this is a significantly more fine-grained approach than using invariants like the *is_form* invariant (cf. the *conclusion* class in section 4).

The construction of this type is technically quite involved and out of the scope of this paper.[6] This part of the theory also yields the constructor $var\_mpoly{::}(('\nu{::}linorder \times nat)list \times '\alpha{::}semiring\_0)list \Rightarrow ('\nu,'\alpha)\ mpoly$ that converts a list of indeterminates indexed by their exponents and pondered by a coefficient into our type-constructor providing an efficient represention for multivariate polynomials $('\nu,'\alpha)\ mpoly$.

Now we zoom into our $paper^m$ example and present an alternative specification of the *definition* class:

[6] It uses, among other Isabelle machinery, the data refinement package by [22] and the Lifting/Transfer packages [23].

Isabelle

```
doc-class relation =
  rel :: "'α::semiring"

doc-class "text_element" = "'β::semiring relation" +
    authored_by :: "'α author set" <= "{}"
    level       :: "int option" <= "None"
    invariant authors_req :: "authored_by ≠ {}"
    and       level_req   :: "the level > 1"

doc-class technical = "('α, 'β::semiring) text_element" +
  id :: nat
  formal_results :: "thm list"

doc-class "definition" = "('α, 'β::semiring) technical" +
    is_formal :: "bool"
```

The *definition* class inherits the order-sorted polymorphic *rel* attribute from the *relation* class.

It is now possible to specify relations between *definition* instances and other documents elements:

Isabelle

```
text*[def1::"('α, (nat, int) mpoly) definition",
     rel = "    var_mpoly [([((id @{theorem ‹safety›}), 1)], 1::int)]
              * var_mpoly [([((id @{theorem ‹security›}), 1)], 1::int)]
            +   var_mpoly [([((id @{result ‹proof1›}), 1)], 1::int)]
              * var_mpoly [([((id @{result ‹proof2›}), 1)], 1::int)]"]‹... text ...›

value*‹rel @{definition ‹def1›}›
```

In our example, the indeterminates of the polynomial will be instances of theorem identifiers. We use *nat* for the identifiers and *int* for the coefficients, so the type of the *def1* instance we just declared is $('\alpha, (nat, int)\ mpoly)\ definition$. *id* @{*theorem* ''*safety*''} refers to the *safety theorem* instance identifier. The evaluation of the *def1* instance *rel* attribute gives:

$$var\_mpoly\ [([(123,\ 1),\ (456,\ 1)],\ 1),\ ([(789,\ 1),\ (987,\ 1)],\ 1)]$$

The value can be understood as follows: the annotated text will depend on the *safety* instance *and* the *security* instance *or* the *proof1* instance *and* the *proof2* instance, where *and* is expressed using the $(*)$ operator and *or* using the $(+)$ operator of the *semiring* when declaring the *rel* attribute-value of *def1*. Intuitively, the *or* is justified by the wish to offer consistently either a more abstract or a more concrete (proof-object based) representation of the dependence on other document elements.

As *proof1* and *proof2* are *result* class instances, they also inherit the *rel* attribute, and could also have specified relations that should be checked. For a certification, the more abstract or concrete representations will be views on the same document for different groups involved in the document specification. A

validation team could be presented with a view where all the informal elements are present and will need to check them manually, i. e., check the content of each informal document element. Another team assigned to the writing of a particular specification document could be presented with a view with only formal elements where the checking process is fully automated using Isabelle/HOL and Isabelle/DOF checking mechanisms, and we are sure that the checking process will end due to the directed acyclic graph architecture of the document and the ordered declaration of the instances enforced by the parsing process. The same team could also specify weights on instances within a *rel* to express levels of recommendation for the presence of an element as a dependence of another document element. Should a software Safety Integrity Level (SIL), expressing the accepted risk of software failure in the certification, require that a specific element is only *Recommended* and not *Highly Recommended* in the document specification that targets this SIL, a weight will be used to reflect this recommendation.

The relation between document elements can always be specified independent of being a formal or informal document element, where the checking of informal documents elements has to be delegated to humans during the certification validation process.

6.2. An Access-Control Model for Integrated Isabelle/HOL Documents

One might object that the suggested *integrated* document model underlying our approach is incompatible with the reality of industrial projects, where their partners will need to enforce strategies to protect their intellectual property. However, a system having access to the integrated document is fundamental for mechanisms to ensure global consistency. A way out of this dilemma are fine-grained access control models allowing decisions on individual text elements for any user role. In this section, we show how such a fine-grained security model can be built with the existing mechanisms of ODL.[7]

Order-sorted polymorphism opens ways to implement security and integrity models based on lattices, such as Bell-LaPadula-like access control models. The key is to provide a *sec* attribute that is to define the basic access control status. This can also be useful when targeting certifications where roles and responsibilities of the involved entities (person, group, organization) are identified.

The *sec* attribute is implemented using a *lattice* sort and declares the security label of the document element. The *sec* attributes of document elements can be computed to express the minimum security level required to access a collection of document elements associated with each other, whether it is by the $is-a$ relation of the class inheritance or by other means, like the how-provenance just presented, or by looking at the instance attributes. First we declare the security labels using a datatype:

[7]Note that our current front-end Isabelle/jEdit does not support access-restriction functionality; however, we consider this as a current technical limitation.

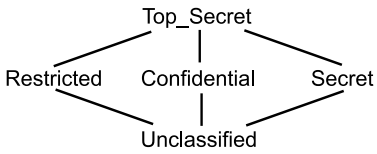


Figure 11: The Lattice of Security Labels.

Isabelle
```
datatype security = Unclassified | Restricted | Confidential
                  | Secret | Top_Secret
```

Then we make the *security* datatype a *lattice* sort instance. The resulting lattice is shown in Figure 11 where security labels are ordered from least sensitive (*Unclassified*) to most sensitive (*Top_Secret*). Finally, we add syntactic definitions for the (+) and (∗) operators.

The new specification of the *result* class, *sec_result*, now has a *sec* attribute inherited from *sec_relation*:

Isabelle
```
doc-class sec_relation =
  sec :: "'β::lattice"

doc-class "sec_text_element" = "('β::lattice) sec_relation" +
    authored_by :: "'α author set"  <= "{}"
    level       :: "int  option"    <= "None"
    invariant authors_req :: "authored_by ≠ {}"
    and       level_req   :: "the level > 1"
```

Isabelle
```
doc-class "sec_technical" = "('α, 'β::lattice) sec_text_element" +
  id :: nat
  formal_results  :: "thm list"

doc-class sec_result = "('α, 'β::lattice) sec_technical" +
  evidence :: sec_kind
  property :: "('α, 'β) sec_theorem list" <= "[]"
  invariant has_property :: "evidence = proof ⟷ property ≠ []"
```

In this setting, we can specify access control as follows:

Isabelle
```
text*[proof3::"('α, security) sec_result",
      id = "789", evidence = "proof",
      property="[@{sec_theorem ‹safety2›}, @{sec_theorem ‹security2›}]",
      level = "Some 2", authored_by = "{@{author ‹church2›}}",
      sec = "Unclassified"]‹... text›
```

The *proof3* instance *sec* attribute-value is *Unclassified*, and its *property* attribute is a list of *sec_theorem*s that also have a security level. We can have a

coarse grain policy where access control to *proof3* will use only its *sec* attribute-value. Then every user with a security level above *Unclassified* will be able to read all *proof3* information, including the information of every entry in its *property* attribute. With a more fine grain policy, the *sec* attribute-value of each *property* list element will also be checked. The *sec* attribute-value of the *security2* instance is *Confidential*, so a user with an *Unclassified* security level should be able to access *proof3* information but not *security2* information when querying *proof3 property* attribute. With the following query:

Isabelle
```
value*‹property @{sec_result ‹proof3›}
      |> filter (λx. statement x = statement @{sec_theorem ‹security2›})›
```

the access control checking mechanism should compute the involved instances security level using the $(*)$ operator for read access:

Isabelle
```
value*‹sec @{sec_result ‹proof3›} * sec @{sec_theorem ‹security2›}›
```

which evaluates to *Confidential*. Whether the access control is coarse or fine grain, a user with an *Unclassified* security level will not be able to make this query because they can not access *security2* information.

To fully integrate our access control model for integrated documents, one possible way is to rely on a remote access to overcome the access control of the local file system. We could extend the VSCode extension of Isabelle/HOL to support security models and host an online instance to access our document stored on the server, using for example an openvscode-server[8] server instance. We plan this extension as future work.

7. A Domain-specific Ontology: CENELEC 50128

In this section, we briefly describe a domain ontology for the required documentation of a CENELEC 50128 certification. The CENELEC 50128 standard is a major industry standard addressing the development and validation processes for software in the railway domain. It provides a framework for software assurance, which is aimed at creating a software package with a minimum level of error, based on quality assurance, skill evaluation, verification and validation, and independent evaluation. Interestingly, it became also relevant for vendors of operating systems such as ThreadX (formerly Microsoft's Azure, cf. [24]), which achieved the highest assurance level. From its beginning, Isabelle/DOF had been used for documents containing formal models targeting certifications.

CENELEC 50128 requires 18 different documents representing milestones; a fully fledged description of our ontology covering (most of) these documents is therefore out of reach of this paper. Instead, we present how the novel concepts

[8] `https://github.com/gitpod-io/openvscode-server/`

such as invariants and term-antiquotations are used in selected elements in this ontology.[9] According to CENELEC Table C.1, for example, we specify the category of "Design and Test Documents" as follows :

Isabelle

```
datatype phase = SR   — Softw. Requirement
               | SDES — Softw. Design  | SV — Softw. Validation | ...
datatype role =  RQM  — Requirements Mgr | DES — Designer
               | IMP  — Implementer      | VER — Verifier  | ...

doc-class cenelec_document = text_element +
  phase      :: phase
  written_by :: role     — Annex C Table C.1
  fst_check  :: role     — Annex C Table C.1
  snd_check  :: role     — Annex C Table C.1
  ...
  invariant must_be_chapter :: "text_element.level σ = Some(0)"
  invariant four_eyes_prcple :: "written_by σ ≠ fst_check σ
                                 ∧ written_by σ ≠ snd_check σ"
```

This class refers to the "software phases" the standard postulates (like *SR* for "Software Requirement" or *SDES* for "Software Design") which we model by corresponding enumeration types. Similarly, the standard postulates "roles" that certain specified teams possess in the overall process (like *verifier* for verification) and assumes that each person is assigned a unique role. We added invariants that specify certain constraints implicit in the standard: for example, a *cenelec_document* must have the textual structure of a chapter (the *level*-attribute is inherited from an underlying ontology library specifying basic text-elements) as well as the four-eyes-principle between authors and checkers of these chapters[10].

A concrete subclass of *cenelec_document* is the class *SWIS* ("software interface specification") as shown below, which provides the role assignment required for this document type:

Isabelle

```
doc-class SWIS
  = cenelec_document + — software interface specification
  phase      :: "phase" <= "SDES"     written_by :: "role" <= "DES"
  fst_check  :: "role" <= "VER"       snd_check  :: "role" <= "VAL"
  components:: "SWIS_E list"
```

The structural constraints expressed so far can in principle be covered by any hand-coded validation process and suitable editing support (e. g., Protégé [6]). However, a closer look at the class *SWIS_E* ("software interface specification element") referenced in the *components*-attribute reveals the unique power of Isabelle/DOF; rather than saying "there must be a pre-condition", Isabelle/DOF

[9] Our CENELEC 50128 ontology is part of the Isabelle/DOF library: `https://github.com/logicalhacking/Isabelle_DOF/blob/main/Isabelle_DOF-Ontologies/CENELEC_50128/CENELEC_50128.thy`.

[10] The standard assumes, as is common in role-based access control, that credentials for roles have not been shared between individuals.

can be far more precise:

Isabelle

```
doc-class SWIS_E =
  op_name          :: "string"
  op_args_res      :: "(string × typ) list × typ" — args and result type
  pre_cond         :: "(string × thm) list"       — labels and predicates
  post_cond        :: "(string × thm) list"       — labels and predicates
  invariant well_formed_pre :: "∀ cond ∈ set(map snd (pre_cond σ)).
                                  iswff_pre (op_args_res σ) (cond)"
  invariant well_formed_post:: ...
```

where the constant $iswff_{pre}$ is bound to a function at the SML-level, that is executed during the evaluation phase of these invariants, and checks the following:

- Any *cond* is indeed a valid definition in the global logical context (taking HOL-libraries but also the concrete certification target model into account).
- Any such HOL definition has the syntactic form:

  $$pre\_<op\_name> \ (a_1{::}\tau_1) \ ... \ (a_n{::}\tau_n) \equiv <predicate>,$$

  where $(a_1{::}\tau_1) \ ... \ (a_n{::}\tau_n)$ correspond to the argument list.

Note that this technique can also be applied to impose specific syntactic constraints on types. For example, via the SI-package available in the Isabelle AFP [25], it is possible to express that the result of some calculation is of type *32 unsigned* $[m \cdot s^{-2}]$, so a 32-bit natural number representing an acceleration in the SI-system. Therefore, it is possible to impose that certain values refer to physical dimensions measured in a concrete metrological system, or constraints on conversions between them, etc.

We used this ontology in the development of an odometer for train systems, tracing requirements from a high-level physics model (describing the requirements as differential equations over real numbers) down to an implementation in C (and verified using AutoCorres2 [26]), running on a microprocessor. The latter uses a discretized measurement of the distance, approximations for derivatives, and works over bounded fixed-point arithmetic. The used CENELEC 50128 ontology leverages the tracking of requirements across the refinement-chain of our models to ensure that the low-level implementation model satisfies the precision requirements expressed in the high-level physics model.

8. An Example Instance: The Distributed Interlocking System.

In this section, we will demonstrate the use of the CENELEC 50128 ontology (as introduced in section 7) in a (fragment of) a certification documentation; our example stems from the context of a distributed interlocking (IXL) method as introduced by [27]. The focus of this section is to showcase the features of the CENELEC 50128 ontology. We do not aim to introduce the IXL method in its

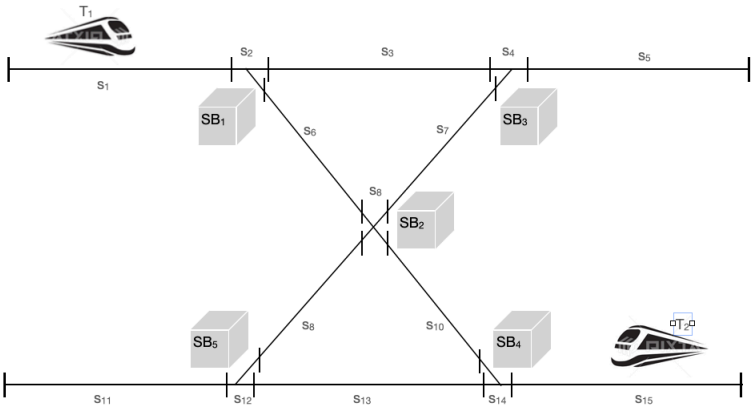


Figure 12: A Distributed Interlocking System for Autonomous Trains.

full detail. For short, the IXL methods provides route-based allocation of paths through an arbitrary railway network topology with points and crossings in order to reach a desired destination. In the scenario sketched in Figure 12 there is no centralized decision point (which would represent a single-point vulnerability); rather, a collection of protocols to so-called *switch boxes* should ensure global safety and reachability. A major advantage of the architecture is that no communication between trains is necessary.

The *cenelec_document* of the *phase SR* will start by establishing a vocabulary of the domain. We annotate these notions from the start to make them navigable and the linking structure explicit:

Isabelle

Definition∗[*def1*::*vocabulary*, *short_name*=“‹*mission specification*› ”]‹*A* ‹*mission specification*› *consists of destination location* (*like* ‹S_{15}› *in the example diagram*), *a route identification, and a start time.*›
Definition∗[*def2*::*vocabulary*, *short_name*=“‹*track element*› ”]‹*A track element to denote any straight or curved track portion, point, and crossing.*›
etc...

It is straight-forward to provide formal definitions for the introduced concepts like *mission specification* and to link them to formal concepts, in order to support document navigation and queries (“which formal definitions are related to the concept?”). An explanation of the general system assumptions capturing the “boundaries of the product” follows:

Isabelle

Assumption∗[*ass1*, *short_name*=“‹*track−elements sensored*› ”]‹*Each track segment, point, or crossing is associated with sensors.*›
Assumption∗[*ass2*, *short_name*=“‹*switch−box−control*› ”]
‹*Each switch box controls one point or one crossing (if any) and monitors the free/occupied status*
of the point and its neighboring track segments.›
etc...

These assumptions will lead to a number of formal well-formedness conditions that will shape the class of admissible networks for which the overall system is guaranteed to work.

A some point, we will have to state the major safety-requirements:

Isabelle

Requirement∗[*R1*::*SR*, *short_name*=“‹*no−collision*› ”]‹*Two different trains may never reside simultaneously on the same track element.*›
Requirement∗[*R2*::*SR*, *short_name*=“‹*no−derailing*› ”]
‹*Whenever a train enters a point or movable crossing 2 from a neighbouring segment 1,*
the point positions or positions of movable crossings must always connect 1 with 2 ...›
Requirement∗[*R3*::*SR*, *short_name*=“‹*no−deadlock*› ”]‹*...*›
etc...

In practice, these requirements will be divided into sub-requirements (in the style of *unified assurance cases* [28]) to make the task to link them more manageable, be it to formal definitions, informal and formal design-decisions and eventually code-elements.

9. Discussion

Undeniably, in a direct comparison of Isabelle/DOF with an off-the-shelf LaTeX editing environment like OverLeaf or TeXStudio, just to name two out of a plethora, the former seems to fall painfully short in terms of usability: the present and subsequent page take about 20 s in an incremental recompilation compared to 4 s in either of the mentioned two[11]. Even when taking into account that Isabelle/DOF makes currently a full triple LaTeX run with BibTeX interlude, which takes the underlying LuaLaTeX about 12s, while OverLeaf optimizes and does just one run, this seems to be long for the small amount of checking done on the present page (i.e., the citations and hyperrefs). On some pages of this document, recompilation is even worse: the section 6, which contains the import of nine theories and several proofs underlying the presented examples which were used—hidden from the reader—to fully type-check and consistency-check the presented material. Here, initial loading takes up to two minutes, and a recompilation-to-previewing up to 35 seconds.

However, we argue that this kind of comparison is misleading. Isabelle/DOF allows for type-checking and a full proof of formal content *as it is presented* in

[11] On a 2,7 GHz Intel Xeon E5 running MacOS.

the paper, up to the semantic depth and up to the degree of layout consistency desired by the author. While conventional LaTeX type-setting is possible inside Isabelle/DOF text-elements, we recommend to use Isabelle markup and antiquotations whenever possible — thus, errors can be detected in the front-end, and re-compilations can thus be avoided. Finally, we suggest shifting proofs or time-consuming evaluations into background sessions pre-compiled to editing, and to devise documents into smaller portions.

It can be safely stated that the Isabelle document generation is fairly regularly used for its technical documentation and many reports and scientific papers by the more regular users of the system (according to the AFP statistics, this figure can be estimated by about 500). As far as the plugin Isabelle/DOF is concerned, it is used by the authors and their co-authors more or less systematically in all of our publications of the last six years. This includes three technical reports and one PhD-thesis [29] of substantial size, two journal articles, and about ten conference papers. Our non-developer co-authors tell us that it feels like using a markdown-like language; in particular, they appreciate the use of antiquotations for HOL-formulas rather than LaTeX typesetting.

And last but not least, we'd like to emphasize that there are a number of obvious and less obvious ways to increase usability, to speed up editing and checking: more incremental heuristics to use the LaTeX backend, more use of Isabelle's advanced parallelization mechanisms, support for common editing patterns and better support of initial document setup.

10. Related Work

This paper has already mentioned conventional ontology modeling languages like OWL; these are often supported by development environments such as Protégé [6] which allow the documentation generation and ontology-based queries in structured texts. The platform allows also the integration of plug-ins that provide Prolog-like reasoners over class invariants in a description logic or fragments of first-order logic. In contrast to OWL, Isabelle/DOF brings forward our concept of *deep ontologies*, i. e., ontologies represented inside an extensible and expressive language such as HOL. Deep ontologies also allow using metalogical entities such as types, terms and theorems, and provide antiquotations as a means to reference *inside* them. The purpose is to establish strong, machine-checkable links between formal and informal content.

Isabelle/DOF's underlying ontology definition language ODL has many similarities with F-Logic [30] and its successors Flora-2 and ObjectLogic.[12] Shared features include object identity, complex objects, inheritance, polymorphic types, query methods, and encapsulation principles. Motivated by the desire for set-theories in modeling, F-Logic possesses syntax for some higher-order constructs but bases itself on first-order logics; this choice limits the potential for user-defined data-type definitions and proofs over classes significantly. Originally designed for object-oriented databases, F-Logic and its successors became

[12] With *OntoStudio* as a commercial ObjectLogic implementation.

mostly used in the area of the *Semantic Web*. In contrast, Isabelle/DOF represents an intermediate layer between a logic like HOL and its implementing languages like SML or OCaml (having their roots as meta-language for these systems). This "in-between" allows for both executability and logical reasoning over meta-data generated to annotate formal terms and texts.

While F-Logic and its successors have similar design objectives, Isabelle/DOF is tuned towards systems with a document-centric view on code and semiformal text as is prevailing in proof-assistants. Not limited to, but currently mostly used as *document*-ontology framework, it has similarity with other documentation generation systems such as `Javadoc` [31, 32], `Doxygen` or `ocamldoc` [33, chap. 19]. These systems are usually external tools run in batch-mode over the sources with a fixed set of structured comments similar to Isabelle/DOF's antiquotations. In contrast, our approach foresees freely user-definable antiquotations, which are in the case of references automatically generated. Furthermore, we provide a flexible and highly configurable LaTeX backend.

Regarding the use of formal methods to formalize standards, the Event-B method was proposed by Fotso et al. [34] for specifications of the hybrid ERTMS/ETCS level 3 standard, in which requirements are specified using SysML/KAOS goal diagrams. The latter were translated into Event-B, where domain-specific properties were specified by ontologies. In another case, Mendil et al. [35] propose an Event-B framework for formalizing standard conformance through formal modelling of standards as ontologies. The proposed approach was exemplified on the ARINC 661 standard. These works are essentially interested in expressing ontological concepts in a formal method but do not explicitly deal with the formalization of invariants defined in ontologies. The question of ontology-mappings is not addressed.

Another work along the line of certification standard support is Isabelle/SACM [28], which is a plug-in into Isabelle/DOF in order to provide specific support for the OMG Structured Assurance Case Meta-Model. The use of Isabelle/SACM guarantees well-formedness, consistency, and traceability of assurance cases, and allows a tight integration of formal and informal evidence of various provenances.

Obvious future applications for supporting the link between *formal* and *informal* content, i.e., between *information* and *knowledge*, consist in advanced search facilities in mathematical libraries such as the Isabelle Archive of Formal Proofs [36]. The latter passed the impressive numbers of 730 articles, written by 450 authors at the beginning of 2023. Related approaches to this application are a search engine like `http://shinh.org/wfs` which uses clever text-based search methods in many formulas, which is, however, agnostic of their logical context and of formal proof. Related is also the OAF project [37] which developed a common ontological format, called OMDoc/MMT, and six *export* functions from major ITP systems into it. Limited to standard search techniques on this structured format, the approach remains agnostic on logical contexts and an in-depth use of typing information.

11. Conclusion and Future Work

We presented Isabelle/DOF, an ontology framework deeply integrating continuous-check/continuous-build functionality into the formal development process in HOL. The novel feature of term-contexts in Isabelle/DOF, which permits term-antiquotations elaborated in the parsing process, paves the way for the abstract specification of meta-data constraints as well as the possibility of advanced search in the meta-data of document elements. Thus, it profits from and extends Isabelle's document-centric view on formal development.

Many ontological languages such as F-Logic as well as the meta-modeling technology available for UML/OCL provide concepts for semantic rules and constraints, but leave the validation checking usually to external tools or plug-ins. Using a combination of advanced code-generation, symbolic execution and reification techniques existing in the Isabelle ecosystem, we provide the advantages of a smooth integration into the Isabelle IDE. Moreover, our approach leverages the use of invariants as first-class citizens, polymorphism, and type-classes into our ontology definition language ODL which turns it into an object of formal study in, for example, ontological mappings. Such a technology exists, to our knowledge, for the first time.

Our experiments with adaptations of existing ontologies from engineering and mathematics show that Isabelle/DOF's ODL has sufficient expressive power to cover all aspects of languages such as OWL (with perhaps the exception to multiple inheritance on classes). However, these ontologies have been developed specifically *in* OWL and target its specific support, the Protégé editor [6]. We argue that Isabelle/DOF might ask for a re-engineering of these ontologies: less deep hierarchies, rather deeper structure in meta-data and stronger invariants.

The development of Isabelle/DOF led to incremental LaTeX generation and a previewing facility that will further increase the usability of our framework for the ontology-conform editing of formal content, be it in the engineering or the mathematics domain Although in an early stage, we believe that this is a valuable contribution to the Isabelle platform in itself.

Another line of future application is to increase the "depth" of term antiquotations such as @**{typ** ‹$'\tau$›**}**, @**{term** ‹$a + b$›**}** and @**{thm** ‹*HOL.refl*›**}**, which are currently implemented just as validations of *references* into the logical context. In the future, they could optionally be expanded to the types, terms, and theorems (with proof objects attached) in a meta-model of the Isabelle Kernel such as the one presented in [38] (also available in the AFP). This will allow for definitions of query-functions in, e. g., proof-objects, and pave the way to annotate them with typed meta-data. Such a technology could be relevant for the interoperability of theories across different ITP platforms.

Availability. Isabelle/DOF is available in the Archive of Formal Proofs (AFP) as "Isabelle/DOF" [5] under a 2-clause BSD-license (SPDX-License-Identifier: BSD-2-Clause). Additional extensions (e.g., ontologies, document templates, and examples) are available on Zenodo [39]. The development version is available at `https://git.logicalhacking.com/Isabelle_DOF/Isabelle_DOF/`.

References

[1] BS EN 50128:2011: Railway applications – communication, signalling and processing systems – software for railway control and protecting systems, Standard, British Standards Institute (BSI) (Apr. 2014).

[2] Common criteria for information technology security evaluation, available at https://www.commoncriteriaportal.org/. (2022).

[3] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL—A Proof Assistant for Higher-Order Logic, Vol. 2283 of LNCS, Springer, 2002. doi:10.1007/3-540-45949-9.

[4] A. D. Brucker, I. Ait-Sadoune, P. Crisafulli, B. Wolff, Using the Isabelle ontology framework: Linking the formal with the informal, in: Conference on Intelligent Computer Mathematics (CICM), no. 11006 in LNCS, Springer, 2018. doi:10.1007/978-3-319-96812-4_3.

[5] A. D. Brucker, N. Méric, B. Wolff, Isabelle/DOF, Archive of Formal Proofs,https://isa-afp.org/entries/Isabelle_DOF.html, Formal proof development (April 2024).

[6] M. A. Musen, The Protégé project: A look back and a look forward, AI Matters 1 (4) (2015) 4–12. doi:10.1145/2757001.2757003.

[7] A. D. Brucker, B. Wolff, Isabelle/DOF: Design and implementation, in: P. C. Ölveczky, G. Salaün (Eds.), Software Engineering and Formal Methods (SEFM), no. 11724 in LNCS, Springer, 2019. doi:10.1007/978-3-030-30446-1_15.

[8] A. D. Brucker, I. Ait-Sadoune, N. Méric, B. Wolff, Using deep ontologies in formal software engineering, in: U. Glässer, D. Méry (Eds.), International Conference on Rigorous State Based Methods (ABZ), no. 14010 in Springer, Springer-Verlag, 2023. doi:10.1007/978-3-031-33163-3_2.

[9] T. Nipkow, Order-sorted polymorphism in Isabelle, in: G. Huet, G. Plotkin (Eds.), Workshop on Logical Environments, 1993, pp. 164–188.

[10] T. Nipkow, C. Prehofer, Type reconstruction for type classes, Journal of Functional Programming 5 (2) (1995) 201–224.

[11] M. Wenzel, The Isabelle/Isar Reference Manual, part of the Isabelle distribution. (2023).

[12] F. Haftmann, T. Nipkow, Code generation via higher-order rewrite systems, in: M. Blume, N. Kobayashi, G. Vidal (Eds.), Functional and Logic Programming, 10th International Symposium (FLOPS), Vol. 6009 of LNCS, Springer, 2010, pp. 103–117. doi:10.1007/978-3-642-12251-4_9.

[13] K. Aehlig, F. Haftmann, T. Nipkow, A compiled implementation of normalisation by evaluation, J. Funct. Program. 22 (1) (2012) 9–30. doi:10.1017/S0956796812000019.

[14] A. V. Hess, S. Mödersheim, A. D. Brucker, A. Schlichtkrull, Performing security proofs of stateful protocols, in: 34th IEEE Computer Security Foundations Symposium (CSF), Vol. 1, IEEE, 2021, pp. 143–158. doi:10.1109/CSF51468.2021.00006.

[15] T. Nipkow, Functional automata, Archive of Formal Proofs.https://isa-afp.org/entries/Functional-Automata.html, Formal proof development (March 2004).

[16] Eclipse Foundation, ATL – a model transformation technology, https://www.eclipse.org/atl/.

[17] Y. A. Ameur, F. Besnard, P. Girard, G. Pierra, J. Potier, Formal specification and metaprogramming in the EXPRESS language, in: The 7th International Conference on Software Engineering and Knowledge Engineering (SEKE), Knowledge Systems Institute, 1995, pp. 181–188.

[18] T. Imieliński, W. Lipski, Incomplete information in relational databases, J. ACM 31 (4) (1984) 761–791. doi:10.1145/1634.1886.

[19] P. Buneman, S. Khanna, W. C. Tan, Why and where: A characterization of data provenance, in: J. V. den Bussche, V. Vianu (Eds.), Database Theory - (ICDT), Vol. 1973 of LNCS, Springer, 2001, pp. 316–330. doi:10.1007/3-540-44503-X_20.

[20] Y. Amsterdamer, D. Deutch, V. Tannen, Provenance for aggregate queries, in: M. Lenzerini, T. Schwentick (Eds.), Symposium on Principles of Database Systems, (PODS), ACM, 2011, pp. 153–164. doi:10.1145/1989284.1989302.

[21] T. J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: L. Libkin (Ed.), Symposium on Principles of Database Systems, ACM, 2007, pp. 31–40. doi:10.1145/1265530.1265535.

[22] F. Haftmann, A. Krauss, O. Kuncar, T. Nipkow, Data refinement in Isabelle/HOL, in: S. Blazy, C. Paulin-Mohring, D. Pichardie (Eds.), Interactive Theorem Proving (ITP), Vol. 7998 of LNCS, Springer, 2013, pp. 100–115. doi:10.1007/978-3-642-39634-2_10.

[23] B. Huffman, O. Kuncar, Lifting and transfer: A modular design for quotients in Isabelle/HOL, in: G. Gonthier, M. Norrish (Eds.), Certified Programs and Proofs CPP 2013, Vol. 8307 of LNCS, Springer, 2013, pp. 131–146. doi:10.1007/978-3-319-03545-1_9.

[24] Eclipse Foundation, Eclipse threadx, https://adx.io/ (2023).

[25] S. Foster and B. Wolff, A Sound Type System for Physical Quantities, Units, and Measurements, https://www.isa-afp.org/entries/Physical_Quantities.html, Accessed: 2023-03-02 (2022).

[26] M. Brecknell, D. Greenaway, J. Hölzl, F. Immler, G. Klein, R. Kolanski, J. Lim, M. Norrish, N. Schirmer, S. Sickert, T. Sewell, H. Tuch, S. Wimmer, Autocorres2, Archive of Formal Proofs,https://isa-afp.org/entries/AutoCorres2.html, Formal proof development (April 2024).

[27] A. E. Haxthausen, J. Peleska, Formal development and verification of a distributed railway control system, IEEE Trans. Software Eng. 26 (8) (2000) 687–701. doi:10.1109/32.879808.

[28] S. Foster, Y. Nemouchi, M. Gleirscher, R. Wei, T. Kelly, Integration of formal proof into unified assurance cases with Isabelle/SACM, Formal Aspects Comput. 33 (6) (2021) 855–884. doi:10.1007/s00165-021-00537-4.

[29] N. Méric, An Ontology Framework For Formal Libraries, Theses, Université Paris-Saclay (Jul. 2024).

[30] M. Kifer, G. Lausen, J. Wu, Logical foundations of object-oriented and frame-based languages, J. ACM 42 (4) (1995) 741–843. doi:10.1145/210332.210335.

[31] B. Venners, J. Gosling, Visualizing with JavaDoc, https://www.artima.com/articles/analyze-this#part3 (2003).

[32] Oracle Corp., The Java API Documentation Generator, https://docs.oracle.com/javase/1.5.0/docs/tool (2011).

[33] The OCaml Manual - Release 5, https://v2.ocaml.org/manual/ocamldoc.html (2022).

[34] S. J. T. Fotso, M. Frappier, R. Laleau, A. Mammar, Modeling the hybrid ERTMS/ETCS level 3 standard using a formal requirements engineering approach, in: Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ), Vol. 10817 of LNCS, Springer, 2018, pp. 262–276. doi:10.1007/978-3-319-91271-4_18.

[35] I. Mendil, Y. Aït-Ameur, N. K. Singh, D. Méry, P. A. Palanque, Standard conformance-by-construction with Event-B, in: Formal Methods for Industrial Critical Systems (FMICS), Vol. 12863 of LNCS, Springer, 2021, pp. 126–146. doi:10.1007/978-3-030-85248-1_8.

[36] Archive of Formal Proofs, https://afp-isa.org, Accessed: 2022-03-15 (2022).

[37] M. Kohlhase, F. Rabe, Experiences from exporting major proof assistant libraries, J. Autom. Reason. 65 (8) (2021) 1265–1298. doi:10.1007/s10817-021-09604-0.

[38] T. Nipkow, S. Roßkopf, Isabelle's metalogic: Formalization and proof checker, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction (CADE), Springer, 2021, pp. 93–110.

[39] A. D. Brucker, B. Wolff, Isabelle/DOF, Zenodo (Aug. 2019). doi:10.5281/zenodo.3370482.