Case Studies in Literate and Structured Formal Developments

Martin Simons, Matthias Weber

March, 1995

Report 95–6

Case Studies in Literate and Structured Formal Developments

Martin Simons, Matthias Weber

Technische Universität Berlin
Forschungsgruppe Softwaretechnik (FR5-6)
Franklinstr. 28/29
10587 Berlin, Germany
email: {simons,we}@cs.tu-berlin.de
fax: +49–30–314 73488

Abstract

We present an approach towards literate and structured presentations of formal developments. We discuss the presentation of formal developments within a logical framework and separate three aspects: language related aspects, structural aspects of proofs, and presentational aspects. We illustrate the approach by two examples: two mathematical proofs and a formalization of the VDM development of a revision management system. The appendix contains the complete formalizations; all of which have been mechanically checked.

Keywords: Formalized mathematics; Formalized developments; Logical frameworks; Literate Programming.

Contents

1 Introduction **3**

2 The approach in detail **4**

3 A quick overview of Deva **6**

4 The Knaster-Tarski fixpoint theorem **7**

5 A revision management system **11**
- 5.1 Abtract level of a kernel revision management system 11
  - 5.1.1 State and invariant 11
  - 5.1.2 Operations 13
  - 5.1.3 Proofs 14
- 5.2 Data-reification steps 14
- 5.3 Extensions to the kernel system 16
- 5.4 The distance to practical revision management systems 16
- 5.5 The development documentation 17

6 Discussion **17**

Appendix 23

A Mathematical proofs **23**
- A.1 The Schröder-Bernstein theorem 23
- A.2 First-order intuitionistic logic 29
- A.3 First order predicate logic 31
- A.4 Elementary set theory 32
- A.5 A theory of maps 33
- A.6 Putting it all together 36
- A.7 Table of Deva sections 36
- A.8 Index of variables 37

B Development of a revision management system **40**
- B.1 Specifications and refinements 40
  - B.1.1 Introduction and overview 40
  - B.1.2 Basic sorts and constants 40
  - B.1.3 Abstract specification of the kernel system 40
  - B.1.4 Data reification to line-based deltas 43
  - B.1.5 Data reification to global array 48
  - B.1.6 Extension by user-held locks 52
  - B.1.7 Extension to robust operations 58
  - B.1.8 Further extensions 59
- B.2 Specification of the kernel system: proofs 59
  - B.2.1 Reset operation 59
  - B.2.2 Open operation 62
  - B.2.3 Checkout operation 66
  - B.2.4 Checkin operation 67
- B.3 1st reification step: validity proofs 71
  - B.3.1 Reset operation 71
  - B.3.2 Open operation 75
  - B.3.3 Checkout operation 79
  - B.3.4 Checkin operation 80

B.4 2nd. reification: retrieve validity . . . 85
B.4.1 Auxiliary lemmas . . . 85
B.4.2 Validity of the retrieve function . . . 86
B.4.3 Operation reification condition: *RESET* . . . 89
B.4.4 Operation reification condition: *OPEN* . . . 90
B.4.5 Operation reification condition: *CHECKOUT* . . . 92
B.4.6 Operation verification condition: *CHECKIN* . . . 94
B.5 Extension by user-held locks: proofs . . . 99
B.5.1 Validity lemmas . . . 99
B.5.2 Reset operation . . . 100
B.5.3 Open operation . . . 101
B.5.4 Set lock operation . . . 102
B.5.5 Release Lck operation . . . 104
B.5.6 Checkin operation . . . 106
B.5.7 Checkout operation . . . 107
B.5.8 Evaluation . . . 108
B.6 Extension to robust operations: proofs . . . 109
B.7 Application theories . . . 110
B.7.1 Files . . . 110
B.7.2 Delta technique . . . 110
B.8 Reification methodology . . . 115
B.8.1 Operations . . . 115
B.8.2 Modules . . . 116
B.8.3 Reification . . . 118
B.8.4 Calculus of operations . . . 120
B.8.5 Module interfaces . . . 127
B.9 Library of basic theories . . . 136
B.9.1 Equality . . . 136
B.9.2 Propositions . . . 138
B.9.3 Ordered pairs . . . 141
B.9.4 Quantifiers . . . 142
B.9.5 Natural numbers . . . 144
B.9.6 Finite sets . . . 146
B.9.7 Sequences . . . 152
B.9.8 Finite mappings . . . 157
B.9.9 Trees . . . 162
B.10 Putting it all together . . . 166
B.11 Table of Deva sections . . . 167
B.12 Index of variables . . . 174

1 Introduction

In this report we present an approach to a literate and structured presentation of formal reasoning. By formal reasoning we mean reasoning expressed in a formal system which can, in principle, be checked or generated mechanically. As formal reasoning is a crucial activity in formal system design it is no longer receiving attention only from logicians: The formal methods community aims at making formal reasoning an established activity in the design of safety-critical systems; mathematicians are pondering again a mechanically proof-checked encyclopedia of mathematics (cf. [B$^+$94]). The major issue of research spawned by this interest in *applied* formal reasoning is to devise formal systems that are suited for a particular application area and that enable a machine to effectively check or generate formal developments. The primary concern is the correctness of these developments which may be, for instance, mathematical proofs, data or program refinements, or program transformations.

One should, however, not forget another concern of equal importance. With applied formal reasoning, we want to convince human beings as well as the machine of the correctness of our arguments. There exists, however, a considerable gap between conventional mathematical reasoning that can be comprehended by human beings and formal reasoning that can be checked by machine. The reason is, of course, the large amount of technical details required by formality and which humans are only too happy to abstract away from. In fact, this *formal noise* usually hides the basic line of reasoning and prevents human comprehension. Besides, these technical details also make it such a laborious task to express formal developments. The problem at hand then is how to bridge this gap.

Before presenting our approach in detail, we would like to briefly recall some of the background work by which it has been influenced. In particular, we want to mention a specific class of formal systems, namely so called *logical frameworks* (cf. [HP91, HP93] for recent compilations of research papers on the subject). The idea here is that, in view the current influx of different logics, it would be nice to have a calculus in which one can encode, or *formalize*, these logics together with their rules of inference. Given an implementation of such a calculus, an implementation of a particular logic can be obtained from a successful encoding of that logic in the calculus. Recognition of the fundamental correspondence between propositions and types has led to a number of different realizations of this idea. The earliest were developed during the Dutch AUTOMATH project (cf. [Bru80]). Among their descendants are more recent developments such as the Calculus of Construction (cf. [CH88]) or LF (cf. [HHP93]). Another well known system is Nuprl, a tactical theorem-prover based on a variant of Martin-Löf's Constructive Type Theory (cf. [C$^+$86]). Implementing higher-order logics has also led to several systems, some well-known examples being HOL (cf. [GM93]), Isabelle (cf. [Pau90]), or IMPS (cf. [FGT93]).

In the next section, we will describe our approach in detail. We will illustrate this approach by two case studies after giving a brief summary in Section 3 of the particular formal system we will use. The first case study is a proof of the Knaster-Tarski fixpoint theorem and is presented in Section 4. The second case study is a formal development of a revision management system that follows the VDM-methodology and is presented in Section 5. These apparently disparate application areas, i.e., mathematical proof and formal system development, demonstrate the general applicability of our approach and moreover give rise to an interesting comparative evaluation. It is for these reasons that we chose to jointly present these two case studies, parts of which have already been published separately [SBR94, Web94]. Both case studies have been mechanically checked for correctness. They are, however, too large to be given in full detail. This is done in a companion technical report [SW94]. In Section 6, we discuss the extent to which the approach attains the aims outlined above and discuss topics for further research.

2 The approach in detail

Our approach can best be described by separating three different aspects of how the gap between mathematical reasoning and formal reasoning can be bridged or, rather, narrowed. The first aspect relates to the language underlying the formal system and in which formal developments are expressed; the second aspect relates to the local and the global structure of a development; and the third aspect relates to the overall presentation of a development. Note that we will attempt to narrow the gap while insisting on still being able to check the formal developments for correctness. This will account for some of the compromises we necessarily have to make.

Recently there has been an increasing amount of work, coming mainly from the computing community, that advocates a more rigorous and disciplined style of mathematical proof with the intention of avoiding mistakes and making proofs more comprehensible. Notable examples include Lamport's proof style described in [Lam93] or the *calculational* proof style described by Dijkstra in [DS90] or by Gries and Schneider in [GS93]. This work can be seen as an attempt to bridge the gap from the other, i.e., the mathematical, side.

The focus of our approach is on *structuring and presenting* formal developments. In the present context, we are not so much interested in the process of *producing* a formal development. This is one of the main differences that distinguish the research reported here from the research generally associated with the logical frameworks mentioned above. That research focuses more on the interactive production of formal developments whereas we focus on the intelligible and well-structured presentation of idealized and stable formal developments. These two approaches should be joined in the future and some ideas how this can be achieved are mentioned in [AJS94].

The formal system we will use as a basis is called Deva and is another descendant of a member of the AUTOMATH family of languages, namely Nederpelt's Λ (cf. [Ned73]). In principle, it is a higher order functional language (see Section 3 below for a quick overview). Deva was originally developed in the context of the Esprit project ToolUse (cf. [Gro90, Sin80, Web91b]) in order to study formal software development methods. Several case studies on formalized software developments have been conducted, some of which have been published (cf. [BL92, Laf90, LLS90, Web90, Web91a, Web94]). A self-coherent presentation of Deva's language theory is given in [Web93a]. An introduction to the language, its theory, and a presentation of two extensive case studies is given in [WSL93]. This book also contains a formalization of much of the VDM methodology. Although the two case studies presented in this article will be expressed in Deva, we will try to keep the following discussion independent from it.

Language

We will call a language in which formal developments are expressed a *proof programming language*. This accounts for the fact that, in general, expressing formal developments, i.e., proofs in the widest sense, is an activity based on principles very similar to programming. Just as the amount of technical detail contained in a program written in an assembly language hinders comprehensibility, the amount of technical detail contained in a formal proof expressed in a (low level) formal system does the same. The goal should be to design a *high level* proof programming language. Such a language should provide a flexible mechanism to define various notations specific to the modeled object theories. 'Rules' should be treated as first class objects, i.e., it should be possible to compose and apply rules and to compose them in various ways to form new rules. The language should be expressive enough to encode various concepts of deductions like forward proofs, backward proofs, calculational proofs, etc. Finally, the language should provide mechanisms for abstracting from messy technical details with an implementation filling in the details during the checking phase. Such mechanisms can range from simple instantiations of first-order parameters of rules to full-fledged theorem proving capabilities. Deva features the possibility to declare mixfix operators; rules are treated as λ-abstractions and can be composed in several ways; various notations are provided to express rule application. Formal developments "with holes" can be expressed at a so called *implicit level*, where higher-order parameters of rules may be left out. They are resolved, where possible, during check-time. These issues are treated in detail in [Web93a, WSL93] and an implementation is described in [AJS94, Anl94].

Most current systems for interactive proof construction support the abstraction from proof details via *tactics* or *tacticals*, which serve essentially to drive the generation of proofs. We want to point out, however, that tactics are not intended and indeed cannot serve as high-level structures for expressing proofs, rather they serve as a command language for batch-like proof generation. Indeed, apart from recent work such as [FGNT94, FH94, SBR94, Web94], current theorem-proving systems do not aim at the presentation of high-level expressions of proofs.

Structure

The global organization and structure of a formal development can also effect decisively comprehensibility. First, a formal development is rarely self-contained, usually being based on a library of basic theories. In addition, auxiliary theories may be developed supporting the core formalization. We believe that traditional concepts and methods from software engineering, such as modularization, encapsulation, library management, etc., can be profitably applied to the *engineering* of formal proofs. Second, there is the issue of how to structure a single proof, derivation, refinement, etc. Here we can try to apply existing techniques and styles used for informal proofs. Leslie Lamport proposes in [Lam93] a method for writing proofs, based on hierarchical structuring. On the highest level, a proof is very sketchy, with more detail being added the lower one gets in the hierarchy. The difference between a good informal proof and a formal one should only be that the formal one needs more levels of detail. We hope to demonstrate in Section 4 that the style proposed by Lamport can be very easily adapted for the presentation of formal Deva proofs. A similar remark can be made about the proof style for VDM advocated by Cliff Jones [Jon90], which is in fact quite similar to the style proposed by Lamport. Where possible, calculational reasoning should be used because it is the easiest form of reasoning that humans can comprehend (cf. [Bac89, DS90, Gas87, GS93]). Calculational reasoning also blends easily and naturally with Lamport's proof style or the VDM proof style. The various case studies performed with Deva and the two contained in this article show that one can indeed mimic various proof styles in Deva. This is achieved by applying the syntactic features of Deva appropriately and by making use of the features provided by the DevaWEB system (see below).

Presentation

It is well known that the way we explain something to a machine is probably not the best way to explain it to a human being. In particular, the syntax of a formal language usually prescribes a fixed order for writing a sentence of the language.

A case in point are computer programs, which are quite hard to understand if read or explained from the first to the last line. It makes much more sense to begin by explaining the central component, and to continue with its dependent components, and so forth. In order to reconcile the requirements of a human being with those of a machine, Donald Knuth designed and implemented his WEB system of structured documentation (cf. [Knu84, Knu92]). In his motivation, he states: "Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do." A similar statement should be made with regard to formal developments: instead of imagining that our main task is to explain to a computer why a formal development is correct, let us concentrate rather on explaining to human beings why it is correct. Thus, to enable us to present a Deva formalization in a "literate programming style", we have designed and implemented a WEB system for Deva [BRS93]. The key feature of the DevaWEB system — like any other WEB system — is that the formalization can be developed together with its documentation. One tool translates ("tangles") a WEB document into a form that is suitable as input for a checker. A second tool translates ("weaves") a WEB document into a TeX-document that can be subsequently processed by the TeX-processor. During this translation, useful cross-reference information is gathered and prepared for typesetting.

The two case studies that will be presented in Section 4 and 5 were developed by following the approach outlined above. They have both been prepared with the assistance of the DevaWEB system and have been mechanically checked by an implementation of Deva. In fact, the source for this article is the actual WEB source.

3 A quick overview of Deva

In the following, we assume that the reader is familiar with basic notions of typed λ-calculi as well as the fundamental principle of "propositions-as-types" and "proofs-as-objects" as it is for example presented in [GLT89]. Deva is a typed λ-calculus with λ-structured types, i.e., the types are λ-terms as well. In Deva λ-terms are called *texts*. So called *contexts* form a second syntactic category and serve to structure formalizations.

In principle, contexts are sequences of bindings that can be named at one place, used at another place, and that can be nested. There are three elementary bindings of text identifiers: A *declaration* of the form '$x\colon t$' introduces the identifier x and declares it to be of type t. A *definition* of the form '$x\colon= t$' introduces the identifier x as an abbreviation for the text t. An *implicit definition* of the form '$x?t$' introduces the identifier x and declares it to be of type t, just like a declaration. In contrast to a declaration, however, in a situation where x has to be instantiated such an instantiation of type t need not be given explicitly but is inferred implicitly. Since t may be any type, a λ-structured type in particular, such inference involves higher order unification. These bindings constitute the atomic contexts. Two contexts may be concatenated by placing a semicolon between them and enclosing them within context brackets '$[\![c_1; c_2]\!]$. This concatenation is associative and therefore all brackets may be omitted but for the outermost ones.

The central λ-terms or texts of Deva are, as usual, *abstraction* and *application*. In an abstraction $[c \vdash t]$, the context c can be seen as a set of parameter declarations or a set of assumptions, whereas the text t can be seen as a function body or conclusion. Thus, an abstraction plays the role of a function body or an inference rule. Because of the second interpretation, there is an equivalent two-dimensional notation for application: $\frac{c}{t}$. Application '$t_1(t_2)$' is defined as usual, where t_2 instantiates the first declared identifier of t_1. Application comes in two additional syntactical variations in order to better express forward or backward directed proofs: Both '$t_1 \mathbin{/} t_2$' and '$t_2 \mathbin{\backslash} t_1$' are equivalent to '$t_1(t_2)$'. (One way to keep the two application dashes apart is to remember that the dash always points towards the function.) A *judgement* '$t_1 \mathrel{.\!\cdot.} t_2$' is a text t_1 together with the assertion that t_2 is its type. Judgements enforce an explicit type check and as such they are often used as a debugging aid. Next, they influence the inference of implicit arguments. Their primary use, however, is to document the progress of a formal development, and all of the judgements in this paper fall into this category. A *named product* '$\langle\!|x_1 := t_1, \ldots, x_n := t_n|\!\rangle$' introduces a finite list of named texts. Access to the components of a product is via *projection* along these names and is expressed as '$t.x$'.

This completes our quick scan of the central constructs of Deva. Their use will be demonstrated in the two examples. Further constructs will be explained in situ. For a more detailed treatment of the language the reader is referred to [Web93a] or [WSL93] which also contains a tutorial-style presentation of the language.

4 The Knaster-Tarski fixpoint theorem

As a first illustration, we will formalize the well known Knaster-Tarski fixpoint theorem, which states that a monotonic map over a lattice has a fixpoint. For a nice overview of various fixpoint theorems which all carry the names of Knaster and Tarski see [LNS82]. A modern introduction to lattice theory is [DP90].

We will first present a proof of Tarski's theorem in Lamport's proof style. Lamport's proof style is, in principle, an adaptation of natural deduction style proofs to proofs in ordinary (viz. non formal) mathematics. Lamport claims and gives convincing arguments that this proof style helps to avoid many mistakes. A proof is structured hierarchically in order to break down and manage its complexity. The lower the level the more detailed it is expressed. If one is only interested in an overview of the proof, one just reads the highest level. More detail is presented at lower levels which one may look up at will. The first thing to do is to state the theorem to be proved:

Theorem (Knaster-Tarski). A monotonic map $\Phi: \mathcal{U} \to \mathcal{U}$ over a complete lattice has a fixpoint.

Next, we give a sketch of the proof, i.e., an informal description of the arguments behind the proof.

PROOF SKETCH. The idea is to show that the least upper bound of the set $M \triangleq \{x : x \in \mathcal{U} : x \sqsubseteq \Phi(x)\}$ is a fixpoint of Φ. We show that $\sqcup M \sqsubseteq \Phi(\sqcup M)$ and that $\Phi(\sqcup M) \sqsubseteq \sqcup M$. And so, by anti-symmetry, equality follows. $\square$

Finally, we give the highest-level of the proof.

LET: $M \triangleq \{x : x \in \mathcal{U} : x \sqsubseteq \Phi(x)\}$
PROVE: $\Phi(\sqcup M) = \sqcup M$
1. $\sqcup M$ is atmost $\Phi(\sqcup M)$.
2. $\Phi(\sqcup M)$ belongs to M.
3. $\Phi(\sqcup M)$ is atmost $\sqcup M$.
4. QED.

Each of the high-level steps is then proved in turn.

1. $\sqcup M$ is atmost $\Phi(\sqcup M)$.
PROOF: It suffices to show that $\Phi(\sqcup M)$ is an upper bound of M, i.e.,
ASSUME: $x \in M$

PROVE: $x \sqsubseteq \Phi(\sqcup M)$
1.1. $x \sqsubseteq \Phi(x)$
PROOF: Definition of M.
1.2. $\Phi(x) \sqsubseteq \Phi(\sqcup M)$
PROOF: Definition of $\sqcup$ and monotonicity of Φ.
1.3. QED.
PROOF: 1.1. and 1.2. and transitivity.

2. $\Phi(\sqcup M)$ belongs to M.
PROOF: Definition of M, 1. and monotonicity of Φ.

3. $\Phi(\sqcup M)$ is atmost $\sqcup M$.
PROOF: Definition of $\sqcup$ and 2.

4. QED (Knaster-Tarski).
PROOF: Anti-symmetry, 1. and 3.

Before one can try to formalize a proof in any calculus, one has to make sure that there exists a formalized theory of the domain of discourse. That also happens informally in any textbook where the axioms of a theory are stated or the key concepts are defined before any theorems are stated and proved. The existence of well designed formalized basic theories is an important requirement for a computer aided formal reasoning system. All systems mentioned in the introduction come equipped with a more or less extensive library of theories. For instance, for Isabelle exists a formalization of ZF-axiomatization of set theory (cf. [Noe93, Pau93]) and several other libraries. For Deva, such a library of theories is currently under development. Due to lack of space, the part of this library which is necessary for our proofs is not included in this paper but [SW94] contains the complete formalization including the library. It comprises a natural deduction style formalization of the predicate calculus with equality, a little set theory and some theory about functions.

Natural deduction calculi are easily and straightforwardly formalized in Deva. For instance the introduction and elimination rules for conjunction are expressed as follows:

$$[P, Q \,?\, prop \vdash \langle\!| \, intro := \left| \frac{P; Q}{P \wedge Q} \right., elim := \langle\!| \left| \frac{P \wedge Q}{P} \right., \left| \frac{P \wedge Q}{Q} \right. |\!\rangle |\!\rangle]$$

In the following, we will make use of **WEB**'s module mechanism to simulate the hierarchical structuring.

4.1. To set the stage for the proof of Knaster-Tarski's fixpoint theorem, we will first give a definition of the concept of a complete lattice.

Definition (complete lattice). A partially ordered set $\langle \mathcal{U}, \sqsubseteq \rangle$ such that any subset A of U has a least upper bound $\sqcup A$ is called a *complete lattice.* The *least upper bound* is uniquely characterized by the following equivalence, where it is understood that A and x are implicitly universally quantified.

$$\forall (y \colon y \in A \colon y \sqsubseteq x) \quad \equiv \quad \sqcup A \sqsubseteq x$$

This definition is formally expressed in Deva as follows:

$\langle$ Complete lattice. 4.1 $\rangle \equiv$

$\mathcal{U}$	$:$ *sort*
$; \, (\cdot) \sqsubseteq (\cdot)$	$: [\mathcal{U}; \mathcal{U} \vdash prop\,]$

$$
\begin{array}{lll}
; \sqcup(\cdot) & : [\, set\,(\mathcal{U}) \vdash \mathcal{U}\,] & \\
; \mathit{partial\_order} & : [x, y, z \mathrel{?} \mathcal{U} \vdash \langle\!| \; \mathit{refl} & := x \sqsubseteq x \qquad\qquad\qquad\qquad] \\
 & \qquad\qquad\qquad\quad , \mathit{anti\_sym} & := [x \sqsubseteq y \wedge y \sqsubseteq x \vdash x = y\,] \\
 & \qquad\qquad\qquad\quad , \mathit{trans} & := [x \sqsubseteq y \wedge y \sqsubseteq z \vdash x \sqsubseteq z\,] \\
 & \qquad\qquad\qquad\quad |\!\rangle & \\
; \mathit{complete\_lattice} & : [\, A \mathrel{?} set\,(\mathcal{U}); y \mathrel{?} \mathcal{U} \vdash \dfrac{[\, x \mathrel{?}\mathcal{U}; x \in A \vdash x \sqsubseteq y\,]}{\sqcup A \sqsubseteq y}\,] &
\end{array}
$$

Remark. The notation '$(\cdot) \sqsubseteq (\cdot)$' declares an infix operator. In fact, in Deva it is possible to declare arbitrary mixfix operators in this way. For instance, '$\sqcup$' is declared to be a prefix operator. It is also possible to assign associativities and precedences to mixfix operators in a very flexible manner. We made use of this facility in this paper to a great extent but the details were hidden because our choice of precedence conforms with standard mathematical tradition. The notation '$[t_1 \vdash t_2]$' is a shorthand for '$[x{:}\, t_1 \vdash_2]$' where x does not occur free in t_2. The notation '$[t_1 \models t_2]$' and its two-dimensional variant is a shorthand for '$\langle\!| \mathit{down}{:}= [t_1 \vdash t_2], \mathit{up}{:}= [t_2 \vdash t_1] |\!\rangle$'. In this paper, we model base sets as objects of type *sort*. Subsets of such base sets are modelled as extension of predicates. More precisely, if $\mathcal{U}$ is of type *sort* (or "is a sort"), then $set\,(\mathcal{U})$ is a sort modelling the power-set of $\mathcal{U}$. If P is a predicate over $\mathcal{U}$, i.e., of type $[\mathcal{U} \vdash prop\,]$ then $ext\,(P)$ is the subset of $\mathcal{U}$, i.e., of type $set\,(\mathcal{U})$, consisting of all those elements for which P holds. The containment relation '$\in$' is specified accordingly.

4.2. By instantiating x to $\sqcup A$ in the characterizing equivalence of the least upper bound one obtains that $\sqcup A$ is indeed an upper bound of A.

$$
\begin{array}{l}
\langle\, \text{Complete lattice. } {\scriptstyle 4.1} \,\rangle{+}\equiv \\
; \mathit{lub\_ub} := \mathit{partial\_order} \,.\, \mathit{refl} \\
\qquad\qquad\quad \therefore [\, A \mathrel{?} set\,(\mathcal{U}) \vdash \sqcup A \sqsubseteq \sqcup A\,] \\
\qquad\qquad \setminus \mathit{complete\_lattice}.\, \mathit{up} \\
\qquad\qquad\quad \therefore [\, A \mathrel{?} set\,(\mathcal{U}); x \mathrel{?} \mathcal{U} \vdash [x \in A \vdash x \sqsubseteq \sqcup A\,]]
\end{array}
$$

4.3. As with Lamport's proof style, we we will now first state (informally) the theorem and give an informal description of the proof. Then we present the highest level of the proof, where lower levels are encapsulated in `WEB` modules. A short informal description of a sub-level proof is used as a name of the module. The section number associated with each module serves as a pointer to the section in which the sub-level proof is continued.

Theorem (Knaster-Tarski). A monotonic map $\Phi\colon U \to U$ over a complete lattice has a fixpoint.

PROOF SKETCH. The idea is to show that the least upper bound of the set $M \triangleq \{x{:}\, x \in U{:}\, x \sqsubseteq \Phi(x)\}$ is a fixpoint of Φ. We show that $\sqcup M \sqsubseteq \Phi(\sqcup M)$ and that $\Phi(\sqcup M) \sqsubseteq \sqcup M$. And so, by anti-symmetry, equality follows. □

$\langle$ Proof of the Knaster-Tarski theorem. ${\scriptstyle 4.3}$ $\rangle \equiv$

$[\,\Phi \quad : \quad [\mathcal{U} \vdash \mathcal{U}\,]$

$;\ monotonic : \quad [x, y\ ?\,\mathcal{U} \vdash [x \sqsubseteq y \vdash \Phi(x) \sqsubseteq \Phi(y)\,]\,]$

$;\ M \quad := \ ext\ ([\ x : \mathcal{U} \vdash x \sqsubseteq \Phi(x)\,])$

$\vdash[\langle \sqcup M$ is atmost $\Phi(\sqcup M)$. 4.3.1 $\rangle$

$;\langle \Phi(\sqcup M)$ belongs to M. 4.3.2 $\rangle$

$;\langle \Phi(\sqcup M)$ is atmost $\sqcup M$. 4.3.3 $\rangle$

$\vdash\langle$ QED (Knaster-Tarski). 4.3.4 $\rangle$

$]\ \therefore \Phi(\sqcup M) = \sqcup M$

$]$

In the following sections, we give the complete proofs for each of the three sublevels and the concluding "QED".

4.3.1. First, we prove $\sqcup M \sqsubseteq \Phi(\sqcup M)$, which is equivalent to $\forall(x\colon x \in M\colon x \sqsubseteq \Phi(\sqcup M))$.

$\langle \sqcup M$ is atmost $\Phi(\sqcup M)$. 4.3.1 $\rangle \equiv$

$lemma_1 := complete\_lattice\,.\,down$

$\quad {}^{/}\,[\ x\ ?\,\mathcal{U};\ hyp\ : x \in M$

$\quad \vdash(conj\,.\,intro \diamond\!\!> partial\_order\,.trans)$

$\quad\quad {}^{/}\,(in\_def\,.\,up\ (hyp) \therefore x \sqsubseteq \Phi(x))$

$\quad\quad {}^{/}\,(monotonic(lub\_ub(hyp)) \therefore \Phi(x) \sqsubseteq \Phi(\sqcup M))$

$\quad\quad\quad \therefore x \sqsubseteq \Phi(\sqcup M)$

$\quad] \therefore \sqcup M \sqsubseteq \Phi(\sqcup M)$

Remark. The construct '$\diamond\!\!>$' is called a *cut* and denotes the composition of two texts both acting as functions. For instance, $[\ x : t \vdash f(x)\,] \diamond\!\!> [\ y : s \vdash g(y)\,]$ is equivalent to $[\ x : t \vdash g(f(x))\,]$, modulo certain type restrictions. Let P and Q be two propositions, then $conj\,.intro$ is of type $[P; Q \vdash P \wedge Q\,]$; for x, y, and z elements of type $\mathcal{U}$, the type of $partial\_order\,.trans$ was declared above as $[x \sqsubseteq y \wedge y \sqsubseteq z \vdash x \sqsubseteq z\,]$. Hence, $conj\,.\,intro \diamond\!\!> partial\_order\,.trans$ is of type $[x \sqsubseteq y; y \sqsubseteq z \vdash x \sqsubseteq z\,]$.

4.3.2. The preceding lemma, by monotonicity, implies that $\Phi(\sqcup M)$ belongs to M.

$\langle \Phi(\sqcup M)$ belongs to M. 4.3.2 $\rangle \equiv$

$lemma_2 := monotonic\ (lemma_1)$

$\quad\quad \therefore \Phi(\sqcup M) \sqsubseteq \Phi(\Phi(\sqcup M))$

$\quad {}^{\backslash}\ in\_def\,.\,down$

$\quad\quad \therefore \Phi(\sqcup M) \in M$

4.3.3. Being a member of M, $\Phi(\sqcup M)$ is atmost $\sqcup M$.

$\langle \Phi(\sqcup M)$ is atmost $\sqcup M$. 4.3.3 $\rangle \equiv$

$lemma_3 := lub\_ub\ (lemma_2) \therefore \Phi(\sqcup M) \sqsubseteq \sqcup M$

4.3.4. Now we are done, because, by anti-symmetry, $\Phi(\sqcup M) = \sqcup M$.

$\langle$ QED (Knaster-Tarski). 4.3.4 $\rangle \equiv$

$(conj\,.\,intro \diamond\!\!> partial\_order\,.anti\_sym)(lemma_3, lemma_1) \therefore \Phi(\sqcup M) = \sqcup M$

5 A revision management system

The presentation of this development case study begins with the abstract description of a kernel system and then extends this kernel in several steps. To emphasize the commonalities and differences of VDM and Deva notations, some pieces of the development will be contrasted in VDM/Deva-displays. The reader may use them as *Rosetta-like stones*[1] for understanding.

5.1 Abtract level of a kernel revision management system

The formalized description of the kernel system is a context consisting of several parts describing the state, its invariant, some basic operations, their assembly into a specification module and the proofs required to discharge the VDM proof obligations.

5.1.1 State and invariant

The kernel state of a revision management system can be described as a tree with finite branching. Revisions are given as text-files, and, for identification, unique identifiers are associated with each revision. The mathematical specification of the kernel state in VDM is presented in the upper half of Figure 1. The revision tree is modeled in two parts: the functional relationship between revision identifiers and revision files ($cont$), and the dependency tree of the revision identifiers (dep). The definition of a finitely-branching tree of revision identifiers ($Tree(Rid)$), an easy exercise in using VDM, has been omitted for reason of space. The invariant ensures that the same revision identifiers are used in both parts of the description, that no revision identifier appears more than once in the dependency tree ($nodup(dep)$), that the number of revisions does not exceed a maximal bound $RevMax$, and that all revision files obey certain well-formedness restrictions ($wff \simeq well\text{-}formed\text{-}file$) , including e.g. restrictions on line-length. Again, for reason of space, the formal definitions of the auxiliary functions and predicates are omitted.

The description in Deva is presented in the lower half of Figure 1. The state specification has been split into three pieces. First, the signature (K_{st}) of the state is defined as a cartesian product ($\otimes$). This suggests to take as state constructor the pair constructor ($\mapsto$) and to define the state selectors as projections. States with more than two components could be modelled by using nested cartesian products. Third, the actual invariant is defined. The definition of the selectors, which is implicit in the VDM notation, has been made explicit in the Deva notation. Similarly, the quantification over the abstract state, which is implicit in the VDM-notation, has been made explicit in the Deva notation. The notations in both descriptions are mostly the same; note, however, that the Deva formalization uses a different notation ($m \nabla a$) for the application of a finite map to an argument. Note that our modelling the VDM tuples by pairs is not faithful, e.g. because of different typing conventions. However, in the context of our case study this modelling was found to be adequate.

It is important to realize the scope of what must already be available in order to express this specification in Deva. In fact, before a development can be formalized, all the background theories must be formalized: First of all, basic theories about logic and the basic datatypes must be formalized. These theories define the notations and properties of operations such as conjunction $\wedge$, universal quantification

[1] Rosetta stone: A black basalt stone found in 1799 that bears an inscription in hieroglyphics, demotic characters, and Greek and is celebrated for having given the first clue to the decipherment of Egyptian hieroglyphics (Webster)

$$
\begin{aligned}
K \;::\; & cont \;:\; Rid \xrightarrow{m} File \\
& dep \;:\; Tree(Rid)
\end{aligned}
$$

where

$$
\begin{aligned}
& inv\text{-}K(mk\text{-}K(cont, dep)) \;\; \triangleq \\
& \quad \mathbf{dom}\, cont = nodes(dep) \\
& \quad \wedge\, nodup(dep) \\
& \quad \wedge\, RevMax \geq \mathbf{card}\,\mathbf{dom}\, cont \\
& \quad \wedge\, \forall r \in \mathbf{dom}\, cont . wff(cont(r))
\end{aligned}
$$

$\langle$ Kernel state and invariant. 5.1.1 $\rangle \equiv$

$$
\begin{aligned}
[\![\; K_{st} \;\; & := \;\; Rid \xrightarrow{m} File \\
& \qquad \otimes\, Tree(Rid) \\
;\; K \;\; & := \;\; \langle\!| \, mk := (\mapsto), cont := sel_1 , dep := sel_2 \, |\!\rangle \\
;\; inv_K \;\; & := \;\; [\, rg : K_{st} \\
& \qquad \vdash \mathbf{dom}\, K.cont\,(rg) = nodes(K.dep\,(rg)) \\
& \qquad\quad \wedge\, nodup(K.dep\,(rg)) \\
& \qquad\quad \wedge\, RevMax \geq \mathbf{card}\,(\mathbf{dom}\, K.cont\,(rg)) \\
& \qquad\quad \wedge\, \forall [\, r : Rid \\
& \qquad\qquad \vdash r \in \mathbf{dom}\, K.cont\,(rg) \Rightarrow wff(K.cont\,(rg) \nabla r) \\
& \qquad] \quad] \\
]\!]
\end{aligned}
$$

Figure 1: Description of the kernel state in VDM and in Deva

$\forall$, cartesian product $\otimes$, binary tuple construction $\mapsto$, and projections sel_1, sel_2. Then, theories about the VDM methodology, in particular about its development relations must be formalized. These theories define operations such as finite mappings ($a \xrightarrow{m} b$) and the domain of a mapping **dom**. Finally, theories related to the application area of the particular development itself must be formalized. These theories define e.g. the well-formedness predicate on files *wff*. The formalization of the actual development has been built upon such theories. The need for well-organized libraries in formal development is thus certainly not smaller than it is in case of programming.

At this point, the reader may wonder what is gained by producing a formal description in Deva, since the VDM description already was formal and even shorter. The answer is that, of course, formal specification is expressible both in VDM and Deva, but, as it will soon be seen, the Deva notation *also* allows expression of formal proofs and of relations between specifications and proofs within development steps. The notational overhead, usually resulting in an expansion of 25-50%, is essentially due to this increase in expressive power. On the other hand, as the previous discussions have illustrated to some degree, the description in Deva preserves the main structural aspects of VDM and does not contain information not already present, explicitly or implicitly, in the VDM description. Hence, it is feasible to define a procedure to translate VDM specifications into Deva, i.e. to generate the lower part of Figure 1 from the upper one. This would allow to keep VDM as it stands, or with minimal adaptations, and just add to it the capability of constructing formal proofs.

5.1.2 Operations

Four operations are defined on the kernel abstract state: an operation to reset the system, an operation to open an empty system with a "root" revision, and two operations to check-in and check-out revisions.

The implicit specifications of the check-in operation in VDM and in Deva are shown in Figure 2. There are three input parameters: the new revision file (f),

$$
\begin{array}{l}
\mathit{CHECKIN}\ (f\colon \mathit{File}, \mathit{prev}, \mathit{new}\colon \mathit{Rid})\\
\mathbf{ext}\ \ \mathbf{wr}\ \mathit{rg}\ :\ K\\
\mathbf{pre}\ \ \mathit{prev} \in \mathbf{dom}\, \overleftarrow{\mathit{rg}}.\mathit{cont}\\
\qquad \wedge\ \mathit{new} \notin \mathbf{dom}\, \overleftarrow{\mathit{rg}}.\mathit{cont}\\
\qquad \wedge\ \mathit{RevMax} > \mathbf{card}\,\mathbf{dom}\, \overleftarrow{\mathit{rg}}.\mathit{cont}\\
\qquad \wedge\ \mathit{wff}(f)\\
\mathbf{post}\ \ \mathit{rg} = K\text{-}\mathit{mk}(\{\mathit{new} \mapsto f\} \odot \overleftarrow{\mathit{rg}}.\mathit{cont},\\
\qquad\qquad\qquad \mathit{insert}(\mathit{prev}, \mathit{new}, \overleftarrow{\mathit{rg}}.\mathit{dep}))
\end{array}
$$

$$
\begin{array}{l}
\langle\, \text{CHECK IN a file. } {\scriptstyle 5.1.2} \,\rangle \equiv\\
[\ \mathit{in}\ :\ \mathit{File} \otimes (\mathit{Rid} \otimes \mathit{Rid});\ \overleftarrow{\mathit{rg}}\ :\ K_{st}\\
\vdash [\ f\ := \ \mathit{sel}_{13}\,(\mathit{in});\ \mathit{prev}\ := \ \mathit{sel}_{23}\,(\mathit{in});\ \mathit{new}\ := \ \mathit{sel}_{33}\,(\mathit{in})\\
\ \ \vdash \langle\!|\ \mathit{pre}\ \ := \mathit{prev} \in \mathbf{dom}\, K.\mathit{cont}\ (\overleftarrow{\mathit{rg}})\\
\qquad\qquad\qquad \wedge\ \mathit{new} \notin \mathbf{dom}\, K.\mathit{cont}\ (\overleftarrow{\mathit{rg}})\\
\qquad\qquad\qquad \wedge\ \mathit{RevMax} > \mathbf{card}\ (\mathbf{dom}\, K.\mathit{cont}\ (\overleftarrow{\mathit{rg}}))\\
\qquad\qquad\qquad \wedge\ \mathit{wff}(f)\\
\qquad ,\ \mathit{post}\ := [\_\_ : \mathbf{void}\,;\mathit{rg} :\ K_{st}\\
\qquad\qquad\qquad \vdash [\ \mathit{update}\ := \ K.\mathit{mk}\ ((\mathit{new} \mapsto f) \odot K.\mathit{cont}\ (\overleftarrow{\mathit{rg}})\\
\qquad\qquad\qquad\qquad\qquad\qquad ,\ \mathit{insert}(\mathit{prev}, \mathit{new}, K.\mathit{dep}\ (\overleftarrow{\mathit{rg}})))\\
\qquad\qquad\qquad\ \ \vdash \mathit{rg} = \mathit{update}\\
]\,]\ |\!\rangle \qquad\qquad\quad]\,]\\
\quad \therefore\ \mathit{op}(\mathit{File} \otimes (\mathit{Rid} \otimes \mathit{Rid}), \mathbf{void}, K_{st})
\end{array}
$$

Figure 2: Specification of Checkin in VDM and in Deva

the name of the revision to be replaced (prev), and the name of the new revision (new). The four preconditions ensure that the revision to be replaced actually exists, that the new revision name has not yet been used, that the system is not yet full, and that the new revision file satisfies the well-formedness restrictions. The postcondition describes how the post-state is obtained by modifying the two components ($K.\mathit{cont}\ (\overleftarrow{\mathit{rg}}), K.\mathit{dep}\ (\overleftarrow{\mathit{rg}})$) of the pre-state. The function *insert* modifies the dependency tree by inserting *new* as a new child of *prev*.

The differences between the two descriptions result from several sources: First, in the Deva description the scope of the post-state (rg) is explicitly limited to the post-condition. The VDM description is shorter in this aspect, since the equivalent information is expressed by scope conventions. Second, due to the homogenity imposed by the Deva concept of typing , all operations specification must be of a similar shape to be typable to a common type. In particular, they must always have exactly one input and one output parameter. Multiple input parameters are handled indirectly by composed types ($File \otimes (Rid \otimes Rid)$) and explicit decoding of the parameters by projecting from such types ($f := sel_{13}\,(in)\ \ldots$). Absence of parameters is expressed by using an empty type (*void*).

Finally, the Deva description includes a judgement about the signature of the operation specification. The signature displayed in Figure 2 is an instantiation of the general scheme

$$op(input\text{-}signature, output\text{-}signature, state\text{-}signature).$$

It is typical for the Deva approach to organize all descriptions under such type schemes and, for clarity of presentation, to display these types by judgements. In this context, the type information is redundant and could be omitted. However, when proofs are to be described in Deva, the display of type information becomes crucial for understanding.

5.1.3 Proofs

The VDM methodology identifies a number of proof obligations to be checked on specifications and developments. For example, for given input and output on which pre- and postcondition hold, all operations must preserve the state invariant.

As a part of one of these proofs, one has to check that the checkin operation does not exceed the maximum size of revisions. In VDM, this proof could be presented as follows in the semi-formal notation proposed by Cliff Jones [Jon90, BFL+94] (upper half of Figure 3).

The size condition to be verified, henceforward called *goal*, is checked by a rather simple calculational reasoning. The idea is to prove *goal* by transforming the tautology $goal \Leftrightarrow goal$ into $goal \Leftrightarrow true$ using properties of the underlying datatypes. The expression *update* stands for the expression on the right hand side of the equation in the postcondition of the check-in operation (Figure 2).

The formal, machine verified, presentation (lower half) in Deva is not very far from that style except that all steps must be justified by giving proof texts involving the underlying laws. The description of these laws is part of the background theories. For example, in the second step (i.e. from formula 2 to formula 3), a property about the domain of finite mappings is used to transform the formal expression. In the background theory of finite mappings the identifier *domain* has been declared with a product type where the component labelled by *recur* is the (simple) law that the domain of a given mapping extended by an additional association pair is equal to the extension of the domain of the given mapping by the left component of the new association. In order to apply this equation within the calculation, it must be lifted using the *unfold* law, declared in the background theory about logical calculi. This law is a special case of the more general Leibniz principle of substitution. Thus in summary, the expression $unfold\,(domain.recur)$ is applied to the previous calculation in order to derive the next formula.

5.2 Data-reification steps

The next step of the development is a data reification that introduces the so-called *delta-technique* which is based on the assumption that subsequent revisions, although huge files, do not differ very much. The idea then is to just store the differences (as tables of lines) between revisions, instead of the revisions themselves. The content of a particular revision thus becomes distributed over the whole revision tree. Here we concentrate on so-called *forward-deltas* (cf. [Tic85]). Forward deltas let $goal := RevMax \geq \mathbf{card}\,(\mathbf{dom}\; cont(update))$ in

1 $goal \Leftrightarrow RevMax \geq \mathbf{card}\,(\mathbf{dom}\; cont(update))$

Definition of $goal$

2 $goal \Leftrightarrow RevMax \geq \mathbf{card}\,(\mathbf{dom}\,(\{new \mapsto f\} \odot cont(\overleftarrow{rg})))$

Projection to $cont(1)$

3 $goal \Leftrightarrow RevMax \geq \mathbf{card}\,(new \odot \mathbf{dom}\; cont(\overleftarrow{rg}))$

Property of finite mappings (2)

4 $goal \Leftrightarrow RevMax \geq 1 + \mathbf{card}\,\mathbf{dom}\; cont(\overleftarrow{rg})$

Since $new \neg \in \mathbf{dom}\; cont(\overleftarrow{rg})$ (3,h1)

5 $goal \Leftrightarrow \mathbf{true}$

Since $RevMax > \mathbf{card}\,(\mathbf{dom}\; cont(\overleftarrow{rg}))$ (4,h1)

6 $goal$

Logic(4)

$\langle$ Proof of the third part of the invariant (CHECKIN). 5.1.3 $\rangle \equiv$

$[\, goal := RevMax \geq \mathbf{card}(\mathbf{dom}\; K.cont\,(update))$

$\vdash refl$

$\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\; K.cont\,(update))$

$\setminus\; unfold(def\_sel_1)$

$\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}(new \mapsto f) \odot K.cont\,(\overleftarrow{rg}))$

$\setminus\; unfold(domain.recur)$

$\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(new \odot \mathbf{dom}\; K.cont\,(\overleftarrow{rg}))$

$\setminus\; unfold(def\_card.\,recur\,.\,new\,(hyp\_pre\_new))$

$\therefore goal \Leftrightarrow RevMax \geq succ(\mathbf{card}(\mathbf{dom}\; K.cont\,(\overleftarrow{rg})))$

$\setminus\; unfold(valid.\,up\,(gth\_prop(hyp\_pre\_safe)))$

$\therefore goal \Leftrightarrow true$

$\setminus\; valid.\,down$

$\therefore goal$

]

Figure 3: Typical snapshot from a invariant preservation proof in VDM and in Deva

are sequences of modification commands to be applied to a revision to generate the next revision. Analogously, backward-deltas are applied to a revision to generate its predecessor.

The description of this development step is of the same overall structure as used in Sect. 5.1, except that, in addition to the usual components, a function is specified that retrieves the original files from the deltas and that formal proofs are given that the development step, i.e. the representation of files by deltas, is a correct data reification in the sense of the VDM methodology. To describe this development step, the VDM standard operators on sets and lists were extended by some second-order operators (e.g. the map of a function over a set/list or the filtering of a set/list with a boolean function). The use of these operators and their laws lead to a significant reduction in specification size and allowed to present most proofs in a calculational style, rather than by inductive arguments. Unfortunately, it is beyond the scope of this paper to present this in detail.

The next development step is another data reification that summarizes the description of all the modification commands used in the deltas into a single global array. In practice, this leads to a significant speed up in computation time. On the other hand, because of the somewhat technical indexing into the global array, the specification has a technical, low-level appearance and becomes hard to read.

The data representation reached after performing these two reification steps can efficiently be implemented in conventional imperative languages.

5.3 Extensions to the kernel system

The specification of the kernel system does not appropriately account for the typical multi-user environment in which a revision system is used. In such an environment, several users may simultaneously check out and check in revisions. This obviously requires some sort of coordination in order to prevent an inconsistent state of the project. A solution is to extend the kernel state K_{st} by a new component *locks*, a partial function from revision identifiers to user identifiers that records which revisions are locked an by which user.

The presence of locks gives rise to two new operations to set and release locks. The other operations are adapted to the new state. For example, the precondition of the check-in operation is strengthened to require that a user must own a lock for a revision in order to check it in.

In order to present these extensions and adaptations and their associated proofs as economically as possible, a simple calculus of operations on specifications, comparable to a simple subset of the schema-calculus of Z, has been used. This calculus defines, for example, an operation to strengthen the precondition of an existing operation by a new restriction. A useful derived law about this strengthening operation relates proof obligations of the given operation to those of the strengthened operation in such a way that the proofs about the given operation can be reused within the proofs about the strengthened operation. Unfortunately, a detailed illustration of this proof structuring technique is beyond the scope of this presentation.

A further extension of the specification consists of a straightforward step in which each operation is augmented so as to keep the revision group unchanged in case its precondition is not satisfied. This has been done using once more simple operations on specifications, in particular those for precondition complementation and operation disjunction.

5.4 The distance to practical revision management systems

The development case study reported here has concentrated on the data representation aspects of the formal development of a revision management system. A practical revision management system, such as RCS [Tic85] or SHAPE [Lam91], has several additional features such as the systematic generation of revision names, the attribution of the revisions with information about the history of their origin, a designated user with special access and modification rights, or the integration with a file management system and its functionality.

The current development could be extended further into these directions. It seems to be quite unclear where to naturally stop this process, because a precise modeling of e.g. the interrelationship of RCS or SHAPE with the UNIX operating system implies that numerous new (including low-level) aspects must be handled appropriately. Selected parts of these aspects have already been studied in the context of formal specification, for example see [MS84] for a specification of the Unix-filing system, but to the author's knowledge there does not exist a complete formal specificiation of systems such as RCS or SHAPE. Such a specification could serve as a starting point for constructing and formalising the development in the style advocated in this report.

5.5 The development documentation

In summary, the literate mathematical presentation of the development of a revision management system is described in a single DevaWEB document containing sections of formal specifications and formal derivations written in Deva, interspersed by textual sections. Sect. 5.1 provides a good idea of what the document looks like. The document is structured into five chapters (Figure 4): Abstract specification of a kernel system (*kernel system*, see Sect. 5.1), successive data reifications of the kernel (*delta technique, global array*, see Sect. 5.2), and two specification extensions of the kernel (*locks, exceptions*, see Sect. 5.3). Except for the second reification, all proof obligations have been discharged by formal proofs and these are contained in the development document. In total the development document has 92 pages, 23 of which amount to formal specifications and 69 of which contain formal proofs. There are 12 long proofs, usually including proofs for local auxiliary lemmas, and 18 short proofs.

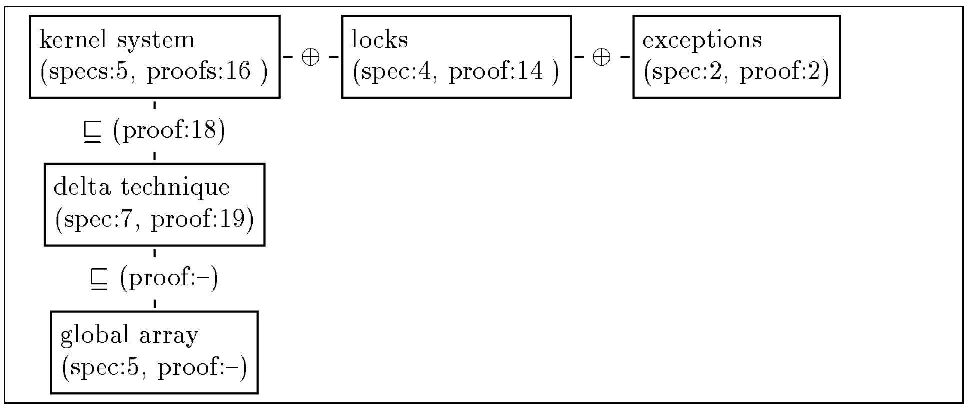

Figure 4: Structure of the development document (with page numbers)

For a fair evaluation, one also has to take into account how much material has been used from libraries (Figure 5). This background material can be divided into three parts: the basic underlying calculi and datatypes, the development objects and relations, and the application-specific theories. Note that the page numbers on Figure 5 refer just to the presentation of axioms and derived laws of the libraries, not to the presentation of the proofs of the derived laws. These proofs are currently formalized, following textbook proofs if available, in an effort to build up a consolidated background library. All these parts taken together with the actual development document and the WEB generated indexes add up to a development document of 186 pages [Web93b].

6 Discussion

This report has presented an approach to formal development whose central underlying goal is to adequately combine mechanical proof support and mathematical presentation style, i.e., to narrow the gap between formal proof and mathematical proof. The application of this approach has been presented by two case studies, one in mathematical proof and one in formal program development. Based on our experiences within these case studies, it is now time for a discussion of the merits and shortcomings of our approach and, based on this discussion, some outlines of

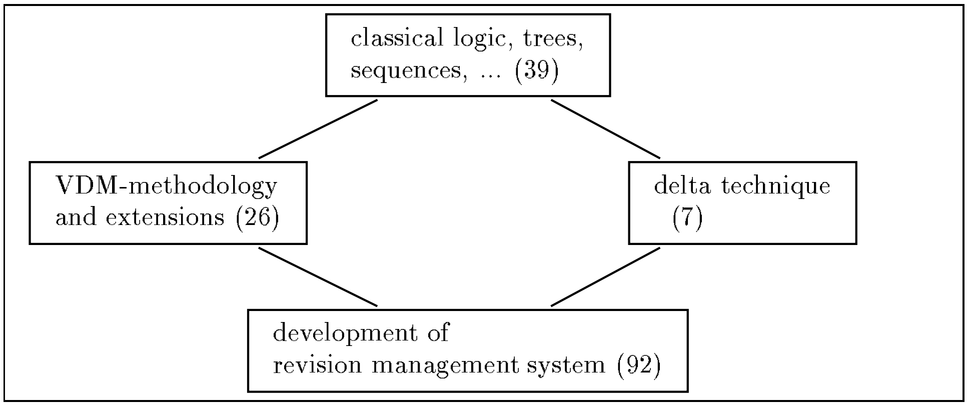

Figure 5: Structure of the background theories (with page numbers)

further work. As in Section 2, we would like to separate this discussion with respect to the aspects of proof language, formalization structure, and presentation.

Language

First of all, it was a pleasant experience to see that this sizeable amount of formal material could be successfully constructed. Thus, the Deva language in principle proved to be a usable proof language. This was mainly due to the quality of the support tools used: It was certainly fun and kept spirits up to use the professionally designed user-interface of Devil and admire the system's ability to check huge formal documents. Somewhat more honestly, we should admit that we mostly had to admire its ability to find plenty of errors in our apparently flawless proofs. This points out a significant advantage of using a proof programming language: human sloppiness in informal proofs is effectively removed. While most of the errors where of minor importance and could be quickly fixed, some errors occuring in the VDM case study pointed to underlying misconceptions and required significant reorganizations or extensions of the development.

However, these positive aspects do not quite diminish the fact that the proof development in the Deva language was very time-consuming and, especially in the VDM case study, forbidding from a practical standpoint. The main reason is the lack of higher-level proof concepts in the Deva language. In fact, when working with the Deva language, we often felt like programming in an assembly-language. This was especially cumbersome since many proofs in the VDM case study amounted more to technical checks rather than deep constructions. The Deva language must thus be seen as a step towards a satisfactory language for expressing formal deductions.

During the design and construction of the formalizations, we were intrigued by the close similarity to the design of software systems. In fact, the well-known principles of designing modular systems applied equally well to the design of formal developments. This attractive relationship should be further investigated and could provide the basis for the design of a truly *high-level* proof programming language (cf. [Sin]).

Structure

Within our case studies, we did not invent our organization from scratch, but rather tried to stick to existing proposals for hierarchically structured mathematical proof

(Leslie Lamport) and systematic software development (Cliff Jones). However, while these two proposals suggest a rigorous but informal setting of development, we have tried to apply them in a completely formal setting. In general, we can say that the formalization has led to an increase, but rarely to more than a duplication, of the size of the corresponding informal development. On the other hand, as we have tried to illustrate in this report, the main structural aspects of both presentation styles have been preserved.

A significant amount of work went into the design and continuous extension and adaptation of the basic libraries. In fact, within the VDM case study these activities accounted for about half of the overall work. While this may be due to the fact that the libraries are relatively new and under ongoing construction, it is clear that some parts of the libraries, e.g. the delta-technique, have a rather narrow application area. Clearly, in order to tackle significant formalizations in a more economical way, a well-structured standard library of commonly used theories must be available.

Another major source of work was the design of elegant proofs. It was a major goal of the VDM case study to organize proofs as much as possible in a calculational style. In a step-wise decomposition process, first versions of the formal proofs were constructed (with backtracking, of course), and subsequently subjected to several iterations of restructuring and simplification. This process could have been much improved and accelerated by relatively simple machine support such as automatic simplification and rewriting or simple inductive proof strategies. The current versions of our tools mainly support the static documentation of a development; they should be adapted towards a more interactive, semi-automatic, style of work.

Presentation

Both case studies were presented in a literate style following the literate programming paradigm. We hope to have demonstrated that this proved to be worthwhile. There are essentially two reasons why we are so pleased with this approach: First, literate development blends nicely with an incremental working mode, i.e., from beginning to end we had available a well-structured documentation of the current status of our work. This helped greatly to keep the complexity under control and concentrate on the truly interesting problems of the case study. Second, the hierarchical organization of our documents according to the literate style was an effective way to document the *higher-level* structures of our developments, which, as we have mentioned above, are not expressible in currently available proof programming languages.

The overall presentation can still be improved in various ways. For instance, the flat structure leaves the reader alone with finding his way through the presentation, although the `WEB` sectioning gives him some guidance. Hypertext technology would be a perfect tool for making the hierarchical structure visible. Another improvement would result from integrating the Deva system with so called "mathematical editors" such as MathSPad [BVW94] or Proxac [Sne93]. They would help in typing and manipulating formal text which for the moment is done by using normal editors.

References

[AJS94] M. Anlauff, S. Jähnichen, and M. Simons. A support system for formal mathematical reasoning. In Naftalin et al. [NDB94], pages 421–440.

[Anl94] M. Anlauff. *A Support Environment for a Generic Devlopment Language.* Forthcoming dissertation., TU Berlin, 1994.

[B+94] R. S. Boyer et al. The QED manifesto. In Bundy [Bun94], pages 238–251. Can also be obtained from `info.mcs.anl.gov` in the directory `/pub/qed`.

[Bac89] R. C. Backhouse. Making formality work for us. *Bulletin of the EATCS*, (38):219–249, June 1989.

[BFL+94] J. C. Bicarregui, J. S. Fitzgerald, P. A. Lindsay, R. Moore, and B. Ritchie. *Proof in VDM: A Practitioner's Guide.* Springer-Verlag, 1994.

[BJ90] M. Broy and C. B. Jones, editors. *Programming Concepts and Methods.* North Holland, 1990.

[BL92] D. Bert and C. Lafontaine. Integration of semantical verification conditions in a specification language definition. In *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology, AMAST '91*, Workshops in Computing. Springer-Verlag, 1992.

[BRS93] M. Biersack, R. Raschke, and M. Simons. The DevaWEB system: Introduction, tutorial, user manual, and implementation. Technical Report 93-39, TU Berlin, 1993.

[Bru80] N. G. de Bruijn. A survey of the project AUTOMATH. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, 1980.

[Bun94] A. Bundy, editor. *Automated Deduction — CADE-12*, volume 814 of *LNCS* . Springer-Verlag, 1994.

[BVW94] R. C. Backhouse, R. Verhoeven, and O. Weber. MathSPad user manual. Technical report, Technical University of Eindhoven, Department of Computer Science, 1994.

[C+86] R. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System.* Prentice Hall, 1986.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[DS90] E. W. Dijkstra and C. Scholten. *Predicate Calculus and Predicate Transformers.* Springer-Verlag, 1990.

[FGNT94] W. M. Farmer, J. D. Guttman, M. E. Nadel, and F. J. Thayer. Proof script pragmatics in IMPS. In Bundy [Bun94], pages 356–370.

[FGT93] W. M. Farmer, J. D. Guttmann, and F. J. Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11:213–248, 1993.

[FH94] A. Felty and D. Howe. Tactic theorem proving with refinement-tree proofs and metavariables. In Bundy [Bun94], pages 605–619.

[Gas87] A. J. M. van Gasteren. *On the Shape of Mathematical Arguments*, volume 445 of *LNCS* . Springer-Verlag, 1987.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proof and Types.* Cambridge University Press, 1989.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.

[Gro90] P. de Groote. *Définition et Properiétés d'un Métacalcul de Représentation de Théories.* PhD thesis, University of Louvain, 1990.

[GS93] D. Gries and F. B. Schneider. *A Logical Approach to Discrete Math.* Springer-Verlag, 1993.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[HP91] G. Huet and G. Plotkin, editors. *Logical Frameworks.* Cambridge University Press, 1991.

[HP93] G. Huet and G. Plotkin, editors. *Logical Environments.* Cambridge University Press, 1993.

[Jon90] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 2 edition, 1990.

[Knu84] D. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.

[Knu92] D. Knuth. *Literate Programming.* Center for the Study of Language and Information, 1992.

[Laf90] C. Lafontaine. Formalization of the VDM reification in the Deva metacalculus. In Broy and Jones [BJ90], pages 333–368.

[Lam91] Andreas Lampen. Advancing files to attributed software-objects. In *Procceedings of the Winter 1991 USENIX Conference*, pages 219–229, Berkeley (CA), January 1991. USENIX Association.

[Lam93] L. Lamport. How to write a proof. Technical Report 94, DEC Systems Research Center, 1993.

[LLS90] Ch. Lafontaine, Y. Ledru, and P. Schobbens. Two approaches towards the formalisation of VDM. In D. Bjorner, C. Hoare, and H. Langmaack, editors, *Proceedings of VDM'90: VDM and Z*, volume 428 of *LNCS*, pages 370–398. Springer, 1990.

[LNS82] J.-L. Lassez, V. L. Nguyen, and E. A. Sonenberg. Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116, 1982.

[MS84] C. Morgan and B. Sufrin. Specification of the unix filing system. *IEEE Transactions of Software Engineering*, SE-10(2):128–140, March 1984.

[NDB94] M. Naftalin, T. Denvir, and M. Bertran, editors. *FME'94: Industrial Benefits of Formal Methods.* Springer-Verlag, 1994.

[Ned73] R. P. Nederpelt. *Strong Normalization in a typed lambda calculus with lambda structured types.* PhD thesis, Technical University of Eindhoven, 1973.

[Noe93] P. A. J. Noel. Experimenting with Isabelle in ZF set theory. *Journal of Automated Reasoning*, 10(1):15–58, 1993.

[Pau90] L. C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[Pau93] L. C. Paulson. Set theory for verification: I. From foundations to functions. *Journal of Automated Reasoning*, 11:353–389, 1993.

[SBR94] M. Simons, M. Biersack, and R. Raschke. Literate and structured presentation of formal proofs. In E.-R. Olderog, editor, *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, pages 61–81. North Holland, 1994.

[Sin] Michel Sintzoff. A proof programming language founded on quantales. submitted to PoPL'95.

[Sin80] M. Sintzoff. Understanding and expressing software construction. In P. Pepper, editor, *Program Transformations and Programming Environments*, pages 169–180. Springer-Verlag, 1980.

[Sne93] J. L. A. van de Snepscheut. Proxac: an editor for program transformation. Technical Report Caltech-CS-TR-93-33, California Institute of Technology, 1993.

[SW94] M. Simons and M. Weber. Case studies in literate and structured formal developments. Technical Report 94-??, TU Berlin, 1994.

[Tic85] W. F. Tichy. RCS — A system for version control. *Software - Practice and Experience*, 15(7):637–654, 1985.

[Web90] M. Weber. Formalization of the Bird-Meertens algorithmic calculus in the Deva meta-calculus. In Broy and Jones [BJ90], pages 201–232.

[Web91a] M. Weber. Deriving transitivity of VDM reification in the Deva meta-calculus. In S. Prehn and W. J. Toetenel, editors, *VDM'91 Formal Software Development Methods*, volume 551 of *LNCS*, pages 406–427. Springer, 1991.

[Web91b] M. Weber. *A Meta-Calculus for Formal System Development.* Oldenbourg Verlag, 1991.

[Web93a] M. Weber. Definition and basic properties of the Deva meta-calculus. *Formal Aspects of Computing*, 5:391–431, 1993.

[Web93b] M. Weber. Development of a revision management system. research report, TU-Berlin, 1993.

[Web94] M. Weber. Literate mathematical development of a revision management system. In Naftalin et al. [NDB94], pages 441–460.

[WSL93] M. Weber, M. Simons, and Ch. Lafontaine. *The Generic Development Language Deva: Presentation and Case Studies*, volume 738 of *LNCS*. Springer-Verlag, 1993.

A Mathematical proofs

A.1 The Schröder-Bernstein theorem

As a more complex example, we will present a proof of the Schröder-Bernstein theorem. Actually, the proof is posed as an exercise in [DP90] as an application of the Knaster-Tarski fixpoint theorem.

Theorem (Schröder-Bernstein). Let X and Y be sets. Suppose there exist one-to-one maps $f\colon X \to Y$ and $g\colon Y \to X$ then there exists a bijective map h from X onto Y.

A.1.1. The proof of the Schröder-Bernstein theorem will make use of Banach's decomposition theorem which we will prove later. Let $f^{\dagger}(X_1)$ denote the canonical lifting of $f\colon X \to Y$ to $f^{\dagger}\colon set(X) \to set(Y)$.

Theorem (Banach decomposition). Let X and Y be sets and let $f\colon X \to Y$ and $g\colon Y \to X$ be maps. Then there exist disjoint subsets X_1 and X_2 of X and disjoint subsets Y_1 and Y_2 of Y such that $Y_1 = f^{\dagger}(X_1)$, $X_2 = g^{\dagger}(Y_2)$, $X_2 = X \setminus X_1$, and $Y_2 = Y \setminus Y_1$.

$\langle$ Type of Banach's decomposition theorem. A.1.1 $\rangle \equiv$

$[X, Y \mathrel{?} \mathit{sort}\,;\ f : [X \vdash Y];\ g : [Y \vdash X]$

$\vdash \exists_4 [X_1, X_2 : \mathit{set}\ (X);\ Y_1, Y_2 : \mathit{set}\ (Y)$

$\quad \vdash Y_1 = f^{\dagger}(X_1) \wedge X_2 = g^{\dagger}(Y_2) \wedge X_2 = X \setminus X_1 \wedge Y_2 = Y \setminus Y_1$

$]\quad]$

A.1.2. The basic idea behind the proof of the Schröder-Bernstein theorem is as follows.

Proof sketch. By Banach's decomposition theorem, there exists a decomposition of $X = X_1 \cup X_2$ and $Y = Y_1 \cup Y_2$ such that $Y_1 = f(X_1)$ and $X_2 = g(Y_2)$. Define $h\colon X \to Y$ to act on X_1 like f and like the inverse of g on X_2. □

$\langle$ Proof of the Schröder-Bernstein theorem. A.1.2 $\rangle \equiv$

$[X, Y \mathrel{?} \mathit{sort}\,;\ f : [X \vdash Y];\ g : [Y \vdash X]$

$;\ \mathit{inj}_f\ :\ \mathit{injective}\ (f)$

$;\ \mathit{inj}_g\ :\ \mathit{injective}\ (g)$

$\vdash \mathit{exiv}\,.\,\mathit{elim}\ (\mathit{banach}(f, g))$

$\quad {}^{\prime}[X_1, X_2 \quad ?\quad \mathit{set}\ (X);\quad Y_1, Y_2 \mathrel{?} \mathit{set}\ (Y)$

$\quad ;\ \mathit{decomposition}\ :\quad Y_1 = f^{\dagger}(X_1) \wedge X_2 = g^{\dagger}(Y_2) \wedge X_2 = X \setminus X_1 \wedge Y_2 = Y \setminus Y_1$

$\quad ;\ h \quad := f \oplus_{X_1} (g^{-1})$

$\quad \vdash [\langle$ h is injective. A.1.2.1 $\rangle$

$\quad\ ;\langle$ h is surjective. A.1.2.3 $\rangle$

$\quad\ \vdash\langle$ QED (Schröder-Bernstein). A.1.2.4 $\rangle$

$]\quad]]$

A.1.2.1. We will first prove that h is injective, i.e., $x = y$ follows from $h(x) = h(y)$ for any x and y. The proof will run by cases, depending on which particular sets of the decomposition of X the elements x and y belong to.

$\langle$ h is injective. A.1.2.1 $\rangle \equiv$

$$
\begin{array}{l}
inj_h \ := [x, y : X \\
\quad \vdash [\ main\_hyp \ : \ h\ (x) = h(y) \\
\quad\quad ; \langle \text{Case: } x, y \in X_1.\ \text{A.1.2.1.1} \rangle \\
\quad\quad ; \langle \text{Case: } x \in X_1, y \in X_2.\ \text{A.1.2.1.2} \rangle \\
\quad\quad ; \langle \text{Case: } x \in X_2, y \in X_1.\ \text{A.1.2.1.4} \rangle \\
\quad\quad ; \langle \text{Case: } x, y \in X_2.\ \text{A.1.2.1.5} \rangle \\
\quad\quad \vdash \langle \text{QED, i.e., } x = y.\ \text{A.1.2.1.6} \rangle \\
\quad\quad]\backslash\ imp.\ intro \\
\quad\quad\quad \therefore h(x) = h(y) \Rightarrow x = y \\
\quad]\backslash\ univii.\ intro \\
\quad\quad \therefore injective(h)
\end{array}
$$

A.1.2.1.1. CASE: $x, y \in X_1$. In this case, $x = y$ follows because f is injective.

$\langle$ Case: $x, y \in X_1$. A.1.2.1.1 $\rangle \equiv$

$$
\begin{array}{l}
lemma_1 \ := [\ local\_hyp_1 \ : x \in X_1;\ local\_hyp_2 \ : y \in X_1 \\
\quad\quad \vdash main\_hyp \\
\quad\quad\quad \therefore h(x) = h(y) \\
\quad\quad \backslash\ Leibniz(mapsum.\ pos\ (local\_hyp_1)) \\
\quad\quad\quad \therefore f(x) = h(y) \\
\quad\quad \backslash\ Leibniz(mapsum.\ pos\ (local\_hyp_2)) \\
\quad\quad\quad \therefore f(x) = f(y) \\
\quad\quad \backslash\ (univii.\ elim\ (inj_f)\ \backslash\ imp.elim) \\
\quad\quad\quad \therefore x = y \\
\quad\quad]
\end{array}
$$

A.1.2.1.2. CASE: $x \in X_1, y \in X_2$. In this case, we arrive at a contradiction, namely we can prove that $f(x) \in Y_1$ and that $f(x) \notin Y_1$. From this contradiction, we will deduce that $x = y$. Note that the same holds for the third case; that is this contradiction is symmetric in x and y. We should be able to save a bit of work, if we parameterize the proof over x and y and instantiate it later accordingly to obtain a contradiction for the second *and* the third case.

$\langle$ Case: $x \in X_1, y \in X_2$. A.1.2.1.2 $\rangle \equiv$

$$
\begin{array}{l}
generic\_lemma_2 \ := [x, y : X \ ;\ main\_hyp \ : \ h\ (x) = h(y) \\
\quad\quad \vdash [\ local\_hyp_1 \ : x \in X_1;\ local\_hyp_2 \ : y \in X \setminus X_1 \\
\quad\quad\quad \vdash [\langle\ y \in X_2.\ \text{A.1.2.1.2.1} \rangle \\
\quad\quad\quad\quad ; \langle f(x) = g^{-1}(y).\ \text{A.1.2.1.2.2} \rangle \\
\quad\quad\quad\quad ; \langle f(x) \in Y_1.\ \text{A.1.2.1.2.3} \rangle \\
\quad\quad\quad\quad ; \langle f(x) \notin Y_1.\ \text{A.1.2.1.2.4} \rangle \\
\quad\quad\quad\quad \vdash \langle \text{Contradiction } (x \in X_1, y \in X_2).\ \text{A.1.2.1.2.5} \rangle \\
\quad\quad]\,]\,]
\end{array}
$$

A.1.2.1.2.1. That $y \in X_2$ follows immediately from the second local hypotheses.

$\langle\, y \in X_2$. A.1.2.1.2.1$\,\rangle \equiv$

$corr_2 \;:=\; Leibniz\ (sym\_eq(conjiv.\ elim\ \mathbf{.3}(decomposition)), local\_hyp_2) \therefore y \in X_2$

A.1.2.1.2.2. $\langle\, f(x) = g^{-1}(y)$. A.1.2.1.2.2$\,\rangle \equiv$

$lemma_1 \;:=\; main\_hyp$

$\quad\therefore h(x) = h(y)$

$\setminus\ Leibniz(mapsum.\ pos\ (local\_hyp_1))$

$\quad\therefore f(x) = h(y)$

$\setminus\ Leibniz(mapsum.\ neg\ (local\_hyp_2))$

$\quad\therefore f(x) = g^{-1}(y)$

A.1.2.1.2.3. $\langle\, f(x) \in Y_1$. A.1.2.1.2.3$\,\rangle \equiv$

$lemma_2 \;:=\; local\_hyp_1$

$\quad\therefore x \in X_1$

$\setminus\ mapext$

$\quad\therefore f(x) \in f^{\dagger}(X_1)$

$\setminus\ Leibniz(sym\_eq(conjiv.\ elim\ \mathbf{.1}(decomposition)))$

$\quad\therefore f(x) \in Y_1$

A.1.2.1.2.4. $\langle\, f(x) \notin Y_1$. A.1.2.1.2.4$\,\rangle \equiv$

$lemma_3 \;:=\; corr_2$

$\quad\therefore y \in X_2$

$\setminus\ mapext$

$\quad\therefore g^{-1}(y) \in (g^{-1})^{\dagger}(X_2)$

$\setminus\ Leibniz(injective\_lemma(inj_g, conjiv.\ elim\ \mathbf{.2}(decomposition)))$

$\quad\therefore g^{-1}(y) \in Y_2$

$\setminus\ Leibniz(sym\_eq(lemma_1))$

$\quad\therefore f(x) \in Y_2$

$\setminus\ Leibniz(conjiv.\ elim\ \mathbf{.4}(decomposition))$

$\quad\therefore f(x) \in Y \setminus (Y_1)$

$\setminus\ in\_def.\ up$

$\quad\therefore \neg(f(x) \in Y_1)$

A.1.2.1.2.5. $\langle$ Contradiction $(x \in X_1, y \in X_2)$. A.1.2.1.2.5$\,\rangle \equiv$

$refutation\ (lemma_3, lemma_2)$

$\quad\therefore$ False

$\setminus\ False\_elim$

$\quad\therefore x = y$

A.1.2.1.3. CASE: $x \in X_1, y \in X_2$. (Once more.) Here we just instantiate the lemma just proven.

$\langle$ Case: $x \in X_1, y \in X_2$. A.1.2.1.2$\,\rangle+ \equiv$

$; \mathit{lemma}_2 \ := [\ \mathit{local\_hyp}_1 \ : x \in X_1; \ \mathit{local\_hyp}_2 \ : y \in (X \setminus X_1)$
$\quad \vdash \mathit{generic\_lemma}_2\ (x, y, \mathit{main\_hyp}, \mathit{local\_hyp}_1, \mathit{local\_hyp}_2)$
$\quad\quad \therefore x = y$
$\quad]$

A.1.2.1.4. CASE: $x \in X_2, y \in X_1$. By symmetry, we arrive at a contradiction just as in the previous case.

$\langle$ Case: $x \in X_2, y \in X_1$. A.1.2.1.4 $\rangle \equiv$
$\mathit{lemma}_3 \ := [\ \mathit{local\_hyp}_1 \ : x \in (X \setminus X_1); \ \mathit{local\_hyp}_2 \ : y \in X_1$
$\quad \vdash \mathit{generic\_lemma}_2\ (y, x, \mathit{sym\_eq}(\mathit{main\_hyp}), \mathit{local\_hyp}_2, \mathit{local\_hyp}_1)$
$\quad\ \setminus \mathit{sym\_eq}$
$\quad\quad \therefore x = y$
$\quad]$

A.1.2.1.5. CASE: $x, y \in X_2$. In order to deduce that $x = y$ in this case, we make use of the fact, that an inverse of any map is injective. Note that in our formalization of the "inverse", we assume that $g(x) = y$ if $x = g^{-1}(y)$. Such a g^{-1}, defined on the range of g is always injective because g is functional. (The dual of injective is functional.)

$\langle$ Case: $x, y \in X_2$. A.1.2.1.5 $\rangle \equiv$
$\mathit{lemma}_4 \ := [\ \mathit{local\_hyp}_1 \ : x \in X \setminus X_1; \ \mathit{local\_hyp}_2 \ : y \in X \setminus X_1$
$\quad \vdash \mathit{main\_hyp}$
$\quad\quad \therefore h(x) = h(y)$
$\quad\ \setminus \mathit{Leibniz}(\mathit{mapsum}.\ \mathit{neg}\ (\mathit{local\_hyp}_1))$
$\quad\quad \therefore (g^{-1})(x) = h(y)$
$\quad\ \setminus \mathit{Leibniz}(\mathit{mapsum}.\ \mathit{neg}\ (\mathit{local\_hyp}_2))$
$\quad\quad \therefore g^{-1}(x) = g^{-1}(y)$
$\quad\ \setminus \mathit{inverse\_injective}$
$\quad\quad \therefore x = y$
$\quad]$

A.1.2.1.6. For each case we have derived $x = y$ thus $x = y$ holds in general under the assumption $h(x) = h(y)$.

$\langle$ QED, i.e., $x = y$. A.1.2.1.6 $\rangle \equiv$
$[\ \mathit{cases} \ := \ \mathit{conj}\ .\ \mathit{intro}\ (\mathit{complement}.\mathit{case\_distinction} \therefore x \in X_1 \vee x \in (X \setminus X_1)$
$\quad\quad , \mathit{complement}.\mathit{case\_distinction} \therefore y \in X_1 \vee y \in (X \setminus X_1))$
$\vdash \mathit{casesii}\ (\mathit{cases}, \mathit{lemma}_1, \mathit{lemma}_2, \mathit{lemma}_3, \mathit{lemma}_4) \therefore x = y$
$]$

A.1.2.3. To prove that h is surjective, we have to show that the union of the ranges of the two component functions is all of Y. Here, we make use of the fact, that $Y = Y_1 \cup Y_2$.

$\langle\, h$ is surjective. A.1.2.3 $\rangle \equiv$

$$
\begin{aligned}
\mathit{surj}_h \ := [&\langle\, \mathrm{ran}(f_{\,|X_1}) = Y_1.\ \text{A.1.2.3.1}\,\rangle \\
;&\langle\, \mathrm{ran}((g^{\text{-}1})_{\,|X\setminus X_1}) = Y_2.\ \text{A.1.2.3.2}\,\rangle \\
\vdash&\langle\, \text{QED } (h \text{ surjective}).\ \text{A.1.2.3.3}\,\rangle \\
]&
\end{aligned}
$$

A.1.2.3.1. $\langle\, \mathrm{ran}(f_{\,|X_1}) = Y_1.\ \text{A.1.2.3.1}\,\rangle \equiv$

$$
\begin{aligned}
\mathit{lemma}_1 \ := \ & \mathit{refl\_eq} \\
& \quad \therefore \mathrm{ran}(f_{\,|X_1}) = f^{\dagger}(X_1) \\
& \setminus \mathit{Leibniz}(\mathit{sym\_eq}(\mathit{conjiv.\ elim}\ \mathbf{.1}(\mathit{decomposition}))) \\
& \quad \therefore \mathrm{ran}(f_{\,|X_1}) = Y_1
\end{aligned}
$$

A.1.2.3.2. $\langle\, \mathrm{ran}((g^{\text{-}1})_{\,|X\setminus X_1}) = Y_2.\ \text{A.1.2.3.2}\,\rangle \equiv$

$$
\begin{aligned}
\mathit{lemma}_2 \ := \ & \mathit{refl\_eq} \\
& \quad \therefore \mathrm{ran}((g^{\text{-}1})_{\,|X\setminus X_1}) = (g^{\text{-}1})^{\dagger}(X \setminus X_1) \\
& \setminus \mathit{Leibniz}(\mathit{sym\_eq}(\mathit{conjiv.\ elim}\ \mathbf{.3}(\mathit{decomposition}))) \\
& \quad \therefore \mathrm{ran}((g^{\text{-}1})_{\,|X\setminus X_1}) = (g^{\text{-}1})^{\dagger}(X_2) \\
& \setminus \mathit{Leibniz}(\mathit{injective\_lemma}(\mathit{inj}_g, \mathit{conjiv.\ elim}\ \mathbf{.2}(\mathit{decomposition}))) \\
& \quad \therefore \mathrm{ran}((g^{\text{-}1})_{\,|X\setminus X_1}) = Y_2
\end{aligned}
$$

A.1.2.3.3. $\langle\, \text{QED } (h \text{ surjective}).\ \text{A.1.2.3.3}\,\rangle \equiv$

$$
\begin{aligned}
& \mathit{union\ .\ total} \\
& \quad \therefore \mathit{set\_total}(Y_1 \cup (Y \setminus Y_1)) \\
& \setminus \mathit{Leibniz}(\mathit{sym\_eq}(\mathit{conjiv.\ elim}\ \mathbf{.4}(\mathit{decomposition}))) \\
& \quad \therefore \mathit{set\_total}(Y_1 \cup Y_2) \\
& \setminus \mathit{Leibniz}(\mathit{sym\_eq}(\mathit{lemma}_1)) \\
& \setminus \mathit{Leibniz}(\mathit{sym\_eq}(\mathit{lemma}_2)) \\
& \quad \therefore \mathit{set\_total}(\mathrm{ran}(f_{\,|X_1}) \cup \mathrm{ran}((g^{\text{-}1})_{\,|(X\setminus X_1)})) \\
& \setminus \mathit{mapsum.\ surjective} \\
& \quad \therefore \mathit{surjective}(h)
\end{aligned}
$$

A.1.2.4. $\langle\,$QED (Schröder-Bernstein). A.1.2.4$\,\rangle \equiv$

$\mathit{isomorphic}\ (\mathit{inj}_h, \mathit{surj}_h) \therefore X \simeq Y$

A.1.4. We will now prove Banach's decomposition theorem which was so instrumental in proving the Schröder-Bernstein theorem.

Theorem (Banach decomposition). Let X and Y be sets and let $f\colon X \to Y$ and $g\colon Y \to X$ be maps. Then there exist disjoint subsets X_1 and X_2 of X and disjoint subsets Y_1 and Y_2 of Y such that $Y_1 = f(X_1)$, $g(Y_2) = X_2$, $X = X_1 \cup X_2$, and $Y = Y_1 \cup Y_2$.

Proof sketch. Consider the map $\Phi\colon \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$ defined on the powerset lattice of X by $\Phi(S) \triangleq X \setminus g(Y \setminus f(S))$. This is monotonic and has thus has by Knaster-Tarski a fixpoint X_1. $\square$

$\langle\,$Proof of Banach's decomposition theorem. A.1.4$\,\rangle \equiv$

$[X, Y ? \quad sort\,;\, f\ : [X \vdash Y];\ g\ : [Y \vdash X]$
$;\Phi \quad := [\, S : set\ (X) \vdash X \setminus g^{\dagger}(Y \setminus f^{\dagger}(S))\,]$
$;\ X_1 \quad := \bigcup(ext([\, S : set\ (X) \vdash S \subseteq \Phi(S)\,]))$
$\vdash [\langle$ Φ is monotonic. A.1.4.1 $\rangle$
$;\langle$ Φ has fixpoint X_1. A.1.4.2 $\rangle$
$\vdash\langle$ QED (Banach decomposition). A.1.4.3 $\rangle$
$]\,]\ \therefore \langle$ Type of Banach's decomposition theorem. A.1.1 $\rangle$

A.1.4.1. The monotonicity of Φ follows from the monotonicity of the functions Φ is composed of. More precisely, the composition (cut) of two monotonic functions or two antimonotonic functions is monotonic. The composition of a monotonic with an antimonotonic function is antimonotonic and vice-versa. Hence, since Φ is composed of two monotonic and two antimonotonic functions, Φ is monotonic. Formally, this line of reasoning is concisely expressed in Deva as follows.

$\langle$ Φ is monotonic. A.1.4.1 $\rangle \equiv$
$lemma_1 := composition\,.\,monanti \mathbin{\diamond\!\!>} composition\,.\,antimon \mathbin{\diamond\!\!>} composition\,.\,antianti$
$\quad ' \,(mapext\_monotonic \therefore monotonic(\subseteq, \subseteq, f^{\dagger}))$
$\quad ' \,(compl\_antimon \therefore antimon(\subseteq, \subseteq, Y\setminus))$
$\quad ' \,(mapext\_monotonic \therefore monotonic(\subseteq, \subseteq, g^{\dagger}))$
$\quad ' \,(compl\_antimon \therefore antimon(\subseteq, \subseteq, X\setminus))$
$\quad\quad \therefore monotonic(\subseteq, \subseteq, (f^{\dagger}) \mathbin{\diamond\!\!>} (Y\setminus) \mathbin{\diamond\!\!>} (g^{\dagger}) \mathbin{\diamond\!\!>} (X\setminus))$
$\quad\quad \therefore monotonic(\subseteq, \subseteq, \Phi)$

A.1.4.2. The subsets of X form a complete lattice with respect to subset inclusion, thus Knaster-Tarski's theorem is applicable to Φ.

$\langle$ Φ has fixpoint X_1. A.1.4.2 $\rangle \equiv$
import $Complete\_lattice\ (set(X), \subseteq, \bigcup, subset\_po, subset\_cl)$
$;\ lemma_2 := knaster\_tarski\ (\Phi, lemma_1) \therefore \Phi(X_1) = X_1$

A.1.4.3. $\langle$ QED (Banach decomposition). A.1.4.3 $\rangle \equiv$
$[\ Y_1 := f^{\dagger}\,(X_1);\ Y_2 := Y \setminus Y_1\,;\ X_2 := g^{\dagger}\,(Y_2)$
$\vdash conjiv\,.\,intro\,(refl\_eq \therefore Y_1 = f^{\dagger}(X_1), refl\_eq \therefore X_2 = g^{\dagger}(Y_2)$
$\quad\quad , \langle\, X_2 = X \setminus X_1.$ A.1.4.3.1 $\rangle, refl\_eq \therefore Y_2 = Y \setminus Y_1)$
$]\setminus\ exiv.\,intro$

A.1.4.3.1. $\langle\, X_2 = X \setminus X_1.$ A.1.4.3.1 $\rangle \equiv$
$refl\_teq$
$\quad \therefore \Phi(X_1) = X \setminus X_2$
$\setminus\ Leibniz(lemma_2)$
$\quad \therefore X_1 = X \setminus X_2$
$\setminus\ complement.\,galois\,.\,down$
$\quad \therefore X \setminus X_1 = X_2$
$\setminus\ sym\_eq$
$\quad \therefore X_2 = X \setminus X_1$

A.2 First-order intuitionistic logic

$\langle$ First order intuitionistic logic. A.2 $\rangle \equiv$
context *FOIL* $:=$
$[\![\langle$ Basic types. A.2.1 $\rangle$
; $\langle$ Signature of FOIL. A.2.2 $\rangle$
; $\langle$ Axioms of FOIL (equality). A.2.3 $\rangle$
; $\langle$ Derived laws of FOIL (equality). A.2.4 $\rangle$
; $\langle$ Overloaded equality (term and functional equality). A.2.5 $\rangle$
; $\langle$ Axioms of FOIL (propositional logic). A.2.6 $\rangle$
; $\langle$ Derived laws of FOIL (propositional logic). A.2.7 $\rangle$
; $\langle$ Axioms of FOIL (quantifiers). A.2.8 $\rangle$
; $\langle$ Derived quantifiers of FOIL. A.2.9 $\rangle$
$]\!]$

A.2.1. $\langle$ Basic types. A.2.1 $\rangle \equiv$
$prop, sort :$ **prim**

A.2.2. $\langle$ Signature of FOIL. A.2.2 $\rangle \equiv$

$$\begin{array}{ll} (\cdot) \wedge (\cdot) & : [prop; prop \vdash prop\,] \\ ;\ (\cdot) \vee (\cdot) & : [prop; prop \vdash prop\,] \\ ;\ (\cdot) \Rightarrow (\cdot) & : [prop; prop \vdash prop\,] \\ ; \mathsf{False} & :\ prop \\ ; \forall\,(\cdot) & : [\ s\ ?\ sort\ ; [s \vdash prop\,] \vdash prop\,] \\ ; \exists\,(\cdot) & : [\ s\ ?\ sort\ ; [s \vdash prop\,] \vdash prop\,] \end{array}$$

A.2.3. $\langle$ Axioms of FOIL (equality). A.2.3 $\rangle \equiv$

$$\begin{array}{lll} ;\ teq & : & [\ s\ ?\ sort\ ; s; s \vdash prop\,] \\ ;\ feq & : & [s, t\ ?\ \ sort\ ; [s \vdash t\,]; [s \vdash t\,] \vdash prop\,] \\ ;\ (\cdot) = (\cdot) & := & \mathbf{alt}\ \ [teq, feq] \end{array}$$

$$\begin{array}{lll} ;\ refl\_teq & : & [\ s\ ?\ sort\ ; x\ ?\ \ s\ \vdash x = x\,] \\ ;\ Leibniz\_teq & : & [\ s\ ?\ sort\ ; x, y\ ?\ \ s\ ; P\ ?\ [s \vdash prop\,] \vdash [x = y \vdash \left| \dfrac{P\ (x)}{P\ (y)} \right.]\,] \\ ;\ refl\_feq & : & [s, t\ ?\ \ sort\ ; f\ ?\ [s \vdash t\,] \vdash f = f\,] \\ ;\ Leibniz\_feq & : & [s, t\ ?\ \ sort\ ; f, g\ ?\ [s \vdash t\,]; P\ ?\ [[s \vdash t\,] \vdash prop\,] \vdash [f = g \vdash \left| \dfrac{P\ (f)}{P\ (g)} \right.]\,] \\ ;\ extensionality & : & [s, t\ ?\ \ sort\ ; f, g\ ?\ [s \vdash t\,] \vdash \left| \dfrac{[\ x\ ?\ s\ \vdash f(x) = g(x)\,]}{f = g} \right.] \end{array}$$

A.2.4. $\langle$ Derived laws of FOIL (equality). A.2.4 $\rangle \equiv$

$$sym\_teq \ := \ Leibniz\_teq\ (\mathbf{6} := refl\_teq) \therefore [\ s\ ?\ sort\ ; x, y\ ?\ \ s\ \vdash \left| \dfrac{x = y}{y = x} \right.]$$

$$; \ \mathit{trans\_teq} \ := (\mathit{sym\_teq} \diamond\!\!> \mathit{Leibniz\_teq}) \therefore [\, s \,?\, \mathit{sort} \,; x, y, z \,?\, \ s \ \vdash \frac{x = y; y = z}{x = z}]$$

$$; \ \mathit{sym\_feq} \ := \ \mathit{Leibniz\_feq}\ (\mathbf{7} := \mathit{refl\_feq}) \therefore [s, t \,?\, \ \mathit{sort} \,; f, g \,?\, [s \vdash t\,] \vdash \frac{f = g}{g = f}]$$

$$; \ \mathit{trans\_feq} \ := (\mathit{sym\_feq} \diamond\!\!> \mathit{Leibniz\_feq}) \therefore [s, t \,?\, \ \mathit{sort} \,; f, g, h \,?\, [s \vdash t\,] \vdash \frac{f = g; g = h}{f = h}]$$

A.2.5. $\langle$ Overloaded equality (term and functional equality). A.2.5 $\rangle \equiv$

$$\begin{aligned} & \mathit{refl\_eq} && := \mathbf{alt}\ [\mathit{refl\_teq}, \mathit{refl\_feq}] \\ ;\ & \mathit{sym\_eq} && := \mathbf{alt}\ [\mathit{sym\_teq}, \mathit{sym\_feq}] \\ ;\ & \mathit{trans\_eq} && := \mathbf{alt}\ [\mathit{trans\_teq}, \mathit{trans\_feq}] \\ ;\ & \mathit{Leibniz} && := \mathbf{alt}\ [\mathit{Leibniz\_teq}, \mathit{Leibniz\_feq}] \end{aligned}$$

A.2.6. $\langle$ Axioms of FOIL (propositional logic). A.2.6 $\rangle \equiv$

$$\begin{aligned} & \mathit{conj} && : [P, Q \,?\, \ \mathit{prop} \\ & && \quad \vdash \langle\!| \ \mathit{intro} \ := \frac{P; Q}{P \wedge Q}, \mathit{elim} \ := \langle\!| \frac{P \wedge Q}{P}, \frac{P \wedge Q}{Q} |\!\rangle |\!\rangle \\ & && \quad] \\ ;\ & \mathit{disj} && : [P, Q \,?\, \ \mathit{prop} \\ & && \quad \vdash \langle\!| \ \mathit{intro} \ := \langle\!| \frac{P}{P \vee Q}, \frac{Q}{P \vee Q} |\!\rangle \\ & && \qquad , \ \mathit{elim} \ := [\, R \,?\, \mathit{prop} \ \vdash \frac{P \vee Q; [P \vdash R]; [Q \vdash R]}{R}] \\ & && \qquad |\!\rangle \\ & && \quad] \\ ;\ & \mathit{imp} && : [P, Q \,?\, \ \mathit{prop} \\ & && \quad \vdash \langle\!| \ \mathit{intro} \ := \frac{[P \vdash Q\,]}{P \Rightarrow Q}, \mathit{elim} \ := \frac{P \Rightarrow Q}{[P \vdash Q\,]} |\!\rangle \\ & && \quad] \\ ;\ & \mathit{False\_elim} && : [\, P \,?\, \mathit{prop} \ \vdash [\mathsf{False} \vdash P\,]\,] \end{aligned}$$

A.2.7. $\langle$ Derived laws of FOIL (propositional logic). A.2.7 $\rangle \equiv$

$$\begin{aligned} & \mathit{conjiv} \ : [A, B, C, D \,?\, \ \mathit{prop} \\ & \quad \vdash \langle\!| \ \mathit{intro} \ := \frac{A; B; C; D}{A \wedge B \wedge C \wedge D} \\ & \quad\ , \ \mathit{elim} \ := \langle\!| \frac{A \wedge B \wedge C \wedge D}{A}, \frac{A \wedge B \wedge C \wedge D}{B} \\ & \qquad\qquad\quad , \frac{A \wedge B \wedge C \wedge D}{C}, \frac{A \wedge B \wedge C \wedge D}{D} \\ & \qquad\qquad\quad |\!\rangle \\ & \quad |\!\rangle \\ & \quad] \end{aligned}$$

$$; \ casesii \ : [A, B, C, D, E \ ? \ prop \ \vdash \left| \frac{(A \vee B) \wedge (C \vee D); \ [A; C \vdash E]; [A; D \vdash E]; [B; C \vdash E]; [B; D \vdash E]}{E} \right.]$$

A.2.8. $\langle$ Axioms of FOIL (quantifiers). A.2.8 $\rangle \equiv$

$; \ univ \ :$

$[\, s \ ? \ sort \, ; \ P \ ? \, [s \vdash prop \,]$

$\vdash \langle\!| \ intro \ := \left| \frac{[\, x : s \ \vdash P(x) \,]}{\forall \ P} \right., \ elim \ := \left| \frac{\forall \ P}{[\, x \ ? \ s \ \vdash P(x) \,]} \right. |\!\rangle$

$]$

$; \ ex \ :$

$[\, s \ ? \ sort \, ; \ P \ ? \, [s \vdash prop \,]$

$\vdash \langle\!| \ intro \ := \left| \frac{x \ ? \ s \, ; P(x)}{\exists \ P} \right., \ elim \ := [\, p \ ? \ prop \ \vdash \left| \frac{\exists \ P}{[[\, x \ ? \ s \, ; P(x) \vdash p \,] \vdash p \,]} \right.] |\!\rangle$

$]$

A.2.9. $\langle$ Derived quantifiers of FOIL. A.2.9 $\rangle \equiv$

$; \forall_2 \, (\cdot) \ := [s_1, s_2 \ ? \ sort \, ; \ P \ : [s_1; s_2 \vdash prop \,]$
$\vdash \forall [\, x_1 : s_1 \ \vdash \forall [\, x_2 : s_2 \ \vdash P(x_1, x_2) \,] \,]$
$]$

$; \ univii \ : \ [s_1, s_2 \ ? \ sort \, ; \ P \ ? \, [s_1; s_2 \vdash prop \,]$
$\vdash \langle\!| \ intro \ := [[\, x_1 : s_1 \, ; x_2 : \ s_2 \ \vdash P(x_1, x_2) \,] \vdash \forall_2 \ P \ ]$
$, \ elim \ := [\forall_2 P \vdash [\, x_1 \ ? \ s_1 \, ; x_2 \ ? \ s_2 \ \vdash P(x_1, x_2) \,] \,]$
$|\!\rangle$
$]$

$; \exists_4 \, (\cdot) \ :=$

$[s_1, s_2, s_3, s_4 \ ? \ sort \, ; \ P \ : [s_1; s_2; s_3; s_4 \vdash prop \,]$

$\vdash \exists [\, x_1 : s_1 \ \vdash \exists [\, x_2 : s_2 \ \vdash \exists [\, x_3 : s_3 \ \vdash \exists [\, x_4 : s_4 \ \vdash P(x_1, x_2, x_3, x_4) \,] \,] \,] \,]$

$]$

$; \ exiv \ :$

$[s_1, s_2, s_3, s_4 \ ? \ sort \, ; \ P \ ? \, [s_1; s_2; s_3; s_4 \vdash prop \,]$

$\vdash \langle\!| \ intro \ := \left| \frac{x_1 \ ? \ s_1 \, ; x_2 \ ? \ s_2 \, ; x_3 \ ? \ s_3 \, ; x_4 \ ? \ s_4 \, ; P(x_1, x_2, x_3, x_4)}{\exists_4 \, P} \right.$

$, \ elim \ := [\, p \ ? \ prop \ \vdash \left| \frac{\exists_4 \, P}{[[\, x_1 \ ? \ s_1 \, ; x_2 \ ? \ s_2 \, ; x_3 \ ? \ s_3 \, ; x_4 \ ? \ s_4 \, ; P(x_1, x_2, x_3, x_4) \vdash p \,] \vdash p \,]} \right.]$

$|\!\rangle$

$]$

A.3 First order predicate logic

$\langle$ First order logic. A.3 $\rangle \equiv$

context *FOPL* $:=$

$[\![$ **import** *FOIL*

; $\langle$ Derived laws of FOPL (negation and refutation). A.3.1 $\rangle$
; $\langle$ Axioms of FOPL (excluded middle). A.3.2 $\rangle$
$\rrbracket$

A.3.1. $\langle$ Derived laws of FOPL (negation and refutation). A.3.1 $\rangle \equiv$

$\neg(\cdot) \quad := [\, P : prop \vdash P \Rightarrow \mathsf{False}\,]$
$;\ refutation\ := \ imp\,.elim \therefore [\, P\ ?\ prop \vdash [\neg P; P \vdash \mathsf{False}\,]\,]$

A.3.2. $\langle$ Axioms of FOPL (excluded middle). A.3.2 $\rangle \equiv$

$excluded\_middle\ : [\, P\ ?\ prop \vdash P \vee \neg\, P\ ]$

A.4 Elementary set theory

$\langle$ Elementary set theory. A.4 $\rangle \equiv$
context $EST :=$
$\llbracket$ $\langle$ Definitions for elementary set theory. A.4.1 $\rangle$
$\rrbracket$

A.4.1. $\langle$ Definitions for elementary set theory. A.4.1 $\rangle \equiv$

$set\ : [sort \vdash sort\,]$
$;\ ext\ : [\mathcal{U}\ ?\ sort \vdash [[\mathcal{U} \vdash prop\,] \vdash set(\mathcal{U})\,]\,]$

A.4.2. Elements.

$\langle$ Definitions for elementary set theory. A.4.1 $\rangle + \equiv$
$;\ (\cdot) \in (\cdot)\ : [\mathcal{U}\ ?\ sort \vdash [\mathcal{U}; set(\mathcal{U}) \vdash prop\,]\,]$
$;\ in\_def\ \ : [\mathcal{U}\ ?\ sort\,; x\ ?\ \mathcal{U}; P\ ?\ [\mathcal{U} \vdash prop\,] \vdash [P(x) \models x \in ext(P)\,]\,]$

A.4.3. Subset ordering and set equality.

$\langle$ Definitions for elementary set theory. A.4.1 $\rangle + \equiv$
$;\ subsetp\ := [\mathcal{U}\ ?\ sort \vdash [A, B : set\ (\mathcal{U}) \vdash [\, x : \mathcal{U} \vdash (x \in A) \Rightarrow (x \in B)\,]\,]\,]$
$;\ (\cdot) \subseteq (\cdot)\ := [\mathcal{U}\ ?\ sort \vdash [A, B : set\ (\mathcal{U}) \vdash \forall(subsetp(A, B))\,]\,]$
$;\ seteq\ \ :\ \ [\mathcal{U}\ ?\ sort\,; A, B : set\ (\mathcal{U}) \vdash \left|\dfrac{(A \subseteq B) \wedge (B \subseteq A)}{A = B}\right.]$

A.4.4. Union.

$\langle$ Definitions for elementary set theory. A.4.1 $\rangle + \equiv$
$;\ unionp\ := [\mathcal{U}\ ?\ sort \vdash [A, B : set\ (\mathcal{U}) \vdash [\, x : \mathcal{U} \vdash (x \in A) \vee (x \in B)\,]\,]\,]$
$;\ (\cdot) \cup (\cdot)\ := \ unionp \diamond\!\!> ext$
$;\ Unionp\ :=$
$[\mathcal{U}\ ?\ sort \vdash [\mathcal{A} : set\ (set(\mathcal{U})) \vdash [\, x : \mathcal{U} \vdash \exists([\, A : set\ (\mathcal{U}) \vdash (A \in \mathcal{A}) \wedge (x \in A)\,])\,]\,]\,]$
$;\ \bigcup(\cdot)\ \ := \ Unionp \diamond\!\!> ext$

A.4.5. Complement.

$\langle$ Definitions for elementary set theory. A.4.1 $\rangle + \equiv$

$$
\begin{array}{ll}
; \; complp & := [\mathcal{U} : sort \vdash [\, A : set\,(\mathcal{U}) \vdash [\, x : \mathcal{U} \vdash \neg(x \in A)\,]\,]\,] \\
; \; (\cdot) \setminus (\cdot) & := [\mathcal{U} : sort \vdash [\, A : set\,(\mathcal{U}) \vdash ext(complp(\mathcal{U}, A))\,]\,]
\end{array}
$$

A.4.6. Properties.

$\langle$ Definitions for elementary set theory. A.4.1 $\rangle + \equiv$

$$
\begin{array}{lll}
; \; set\_total & := [\mathcal{U} \,?\, sort \,; A : set\,(\mathcal{U}) \vdash \forall[\, x : \mathcal{U} \vdash x \in A\,]\,] & \\
; \; set\_empty & := [\mathcal{U} \,?\, sort \,; A : set\,(\mathcal{U}) \vdash \forall[\, x : \mathcal{U} \vdash \neg x \in A\,]\,] & \\
; \; union & : & [\mathcal{U} \,?\, sort \\
& & \vdash \langle\!| \; commutative \; := [A, B \,?\, set\,(\mathcal{U}) \vdash A \cup B = B \cup A\,] \\
& & \quad , \; total \; := [\, A \,?\, set\,(\mathcal{U}) \vdash set\_total(A \cup (\mathcal{U} \setminus A))\,] \\
& & \quad |\!\rangle \\
& &] \\
; \; complement & : & [\mathcal{U} \,?\, sort \\
& & \vdash \langle\!| \; case\_distinction \; := [\, A \,?\, set\,(\mathcal{U}); x \,?\, \mathcal{U} \vdash x \in A \vee x \in (\mathcal{U} \setminus A)\,] \\
& & \quad , \; galois \; := [A, B \,?\, set\,(\mathcal{U}) \vdash [A = \mathcal{U} \setminus B \models \mathcal{U} \setminus A = B\,]\,] \\
& & \quad |\!\rangle \\
& &] \\
; \; subset\_po & : & [\mathcal{U} \,?\, sort \,; A, B, C \,?\, set\,(\mathcal{U}) \\
& & \vdash \langle\!| \; refl \; := A \subseteq A \\
& & \quad , \; anti\_sym \; := [A \subseteq B \wedge B \subseteq A \vdash A = B\,] \\
& & \quad , \; trans \; := [A \subseteq B \wedge B \subseteq C \vdash A \subseteq C\,] \\
& & \quad |\!\rangle \\
& &] \\
; \; subset\_cl & : & [\mathcal{U} \,?\, sort \,; \mathcal{A} \,?\, set\,(set(\mathcal{U}));\; B \,?\, set\,(\mathcal{U}) \\
& & \vdash \dfrac{[\, A \,?\, set\,(\mathcal{U}); A \in \mathcal{A} \vdash A \subseteq B\,]}{\bigcup \mathcal{A} \subseteq B} \\
& &]
\end{array}
$$

A.5 A theory of maps

$\langle$ A theory of maps. A.5 $\rangle \equiv$

context $ATM :=$

$[\![$ $\langle$ Definitions for a theory of maps. A.5.1 $\rangle$

$]\!]$

A.5.1. Range.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle \equiv$

$$
\begin{array}{l}
\mathrm{ran} := [X, Y \,?\, sort \,; f : [X \vdash Y\,] \\
\qquad \vdash ext\,([\, y : Y \vdash \exists([\, x : X \vdash f(x) = y\,])\,]) \\
\qquad]
\end{array}
$$

A.5.2. Restriction.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; (\cdot)_{|(\cdot)} \quad : [X, Y \,?\, \mathit{sort}\,; [X \vdash Y]; \mathit{set}(X) \vdash [X \vdash Y]]$

$; \mathit{restriction} : [X, Y \,?\, \mathit{sort}\,; f \,?\, [X \vdash Y]; A \,?\, \mathit{set}\,(X); a \,?\, X \vdash \left| \dfrac{a \in A}{(f_{\,|A})(a) = f(a)} \right.]$

A.5.3. Extension to sets.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; (\cdot)^{\dagger} \quad := [X, Y \,?\, \mathit{sort}\,; f : [X \vdash Y] \vdash [A : \mathit{set}\,(X) \vdash \mathrm{ran}(f_{\,|A})]]$

$; \mathit{mapext} : \quad [X, Y \,?\, \mathit{sort}\,; f \,?\, [X \vdash Y];\ x \,?\, X\,;\ A \,?\, \mathit{set}\,(X)$

$\vdash [x \in A \vdash f(x) \in f^{\dagger}(A)]$

$]$

A.5.4. Injective maps and inverses.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; \mathit{injective} \quad := [X, Y \,?\, \mathit{sort}\,; f : [X \vdash Y] \vdash \forall_2[x, y :\, X \vdash f(x) = f(y) \Rightarrow x = y]]$

$; (\cdot)^{-1} \quad : \quad [X, Y \,?\, \mathit{sort} \vdash [[X \vdash Y] \vdash [Y \vdash X]]]$

$; \mathit{inverse} \quad : \quad [X, Y \,?\, \mathit{sort}\,; f \,?\, [X \vdash Y]; x \,?\, X\,; y \,?\, Y \vdash [x = f^{-1}(y) \vdash f(x) = y]]$

$; \mathit{inverse\_injective} : \quad [X, Y \,?\, \mathit{sort}\,; f \,?\, [X \vdash Y]; x, y \,?\, Y \vdash [f^{-1}(x) = f^{-1}(y) \vdash x = y]]$

$; \mathit{injective\_lemma} : \quad [X, Y \,?\, \mathit{sort}\,; f \,?\, [X \vdash Y];\ A \,?\, \mathit{set}\,(X);\ B \,?\, \mathit{set}\,(Y)$

$\vdash \left| \dfrac{\mathit{injective}\,(f); B = f^{\dagger}(A)}{(f^{-1})^{\dagger}(B) = A} \right.$

$]$

A.5.5. Surjective maps.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; \mathit{surjective} := [X, Y \,?\, \mathit{sort} \vdash [f : [X \vdash Y] \vdash \mathit{set\_total}(\mathrm{ran}(f))]]$

A.5.6. Isomorphic sorts.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; (\cdot) \simeq (\cdot) \quad : [X, Y :\, \mathit{sort} \vdash \mathit{prop}]$

$; \mathit{isomorphic} : [X, Y \,?\, \mathit{sort}\,; h \,?\, [X \vdash Y] \vdash \left| \dfrac{\mathit{injective}\,(h); \mathit{surjective}(h)}{X \simeq Y} \right.]$

A.5.7. Sum.

$\langle$ Definitions for a theory of maps. A.5.1 $\rangle + \equiv$

$; (\cdot) \oplus_{(\cdot)} (\cdot) : [X, Y \,?\, \mathit{sort} \vdash [[X \vdash Y]; [X \vdash Y]; \mathit{set}(X) \vdash [X \vdash Y]]]$

$$
\begin{array}{lll}
; \ \mathit{mapsum} & : [X, Y\ ?\ \mathit{sort}\ ; f, g\ ?\ [X \vdash Y];\ A\ ?\ \mathit{set}\,(X);\ x\ ?\ X & \\
& \vdash \langle\!| \ \mathit{pos} & := [x \in A \vdash (f \oplus_A g)(x) = f(x)] \\
& \quad , \mathit{neg} & := [x \in (X \setminus A) \vdash (f \oplus_A g)(x) = g(x)] \\
& \quad , \mathit{surjective} & := [\ \mathit{set\_total}\,(\mathrm{ran}(f_{\,|A}) \cup \mathrm{ran}(g_{\,|(X \setminus A)})) \\
& & \quad \vdash \mathit{surjective}\,(f \oplus_A g) \\
& & \quad] \\
& \quad |\!\rangle & \\
&] &
\end{array}
$$

A.5.8. Monotonicity.

⟨ Definitions for a theory of maps. A.5.1 ⟩+ ≡

$$
\begin{array}{ll}
; \ \mathit{monotonic} & := [X, Y\ ?\ \mathit{sort}\ ;\ R\ : [X; X \vdash \mathit{prop}];\ Q\ : [Y; Y \vdash \mathit{prop}];\ f\ : [X \vdash Y] \\
& \quad \vdash [a, b\ ?\ X\ ; R(a,b) \vdash Q(f(a), f(b))] \\
& \quad] \\
; \ \mathit{antimon} & := [X, Y\ ?\ \mathit{sort}\ ;\ R\ : [X; X \vdash \mathit{prop}];\ Q\ : [Y; Y \vdash \mathit{prop}];\ f\ : [X \vdash Y] \\
& \quad \vdash [a, b\ ?\ X\ ; R(a,b) \vdash Q(f(b), f(a))] \\
& \quad]
\end{array}
$$

A.5.9. The composition of monotonic maps is monotonic.

⟨ Definitions for a theory of maps. A.5.1 ⟩+ ≡

$$
\begin{array}{l}
; \ \mathit{composition}\ : \\
\ [X, Y, Z\ ?\ \mathit{sort} \\
\ ; R \qquad ?\ [X; X \vdash \mathit{prop}];\ \ Q\ \ ?\ [Y; Y \vdash \mathit{prop}];\ \ P\ \ ?\ [Z; Z \vdash \mathit{prop}] \\
\ ; f \qquad ?\ [X \vdash Y]; \qquad g\ \ ?\ [Y \vdash Z] \\
\ \vdash \langle\!|\ \mathit{monotonic}\ := [\ \mathit{monotonic}\,(R, Q, f);\ \mathit{monotonic}(Q, P, g) \vdash \mathit{monotonic}(R, P, f \diamond\!\!> g)] \\
\quad , \mathit{antimon}\ \ \ := [\ \mathit{antimon}\,(R, Q, f);\ \mathit{monotonic}(Q, P, g) \vdash \mathit{antimon}(R, P, f \diamond\!\!> g)] \\
\quad , \mathit{monanti}\ \ \ := [\ \mathit{monotonic}\,(R, Q, f);\ \mathit{antimon}(Q, P, g) \vdash \mathit{antimon}(R, P, f \diamond\!\!> g)] \\
\quad , \mathit{antianti}\ \ \ := [\ \mathit{antimon}\,(R, Q, f);\ \mathit{antimon}(Q, P, g) \vdash \mathit{monotonic}(R, P, f \diamond\!\!> g)] \\
\quad |\!\rangle \\
\ ]
\end{array}
$$

A.5.10. The extension of a map is monotonic with respect to set inclusion.

⟨ Definitions for a theory of maps. A.5.1 ⟩+ ≡

$$; \ \mathit{mapext\_monotonic}\ : [X, Y\ ?\ \mathit{sort}\ ; f\ ?\ [X \vdash Y] \vdash \mathit{monotonic}(\subseteq, \subseteq, f^{\dagger})]$$

A.5.11. The complement is an antimonotonic function.

⟨ Definitions for a theory of maps. A.5.1 ⟩+ ≡

$$; \ \mathit{compl\_antimon}\ : [\ X\ ?\ \mathit{sort}\ \vdash \mathit{antimon}(\subseteq, \subseteq, X\setminus)]$$

A.6 Putting it all together

context *Library* $:=$

$[\![$ ⟨ First order intuitionistic logic. A.2 ⟩

; **import** ⟨ First order logic. A.3 ⟩

; **import** ⟨ Elementary set theory. A.4 ⟩

; **import** ⟨ A theory of maps. A.5 ⟩

; **context** *Complete_lattice* $:=$

$[\![$ ⟨ Complete lattice. 4.1 ⟩

; *knaster_tarski* $:=$ ⟨ Proof of the Knaster-Tarski theorem. 4.3 ⟩

$]\!]$

; *banach* $:=$ ⟨ Proof of Banach's decomposition theorem. A.1.4 ⟩

; *sb* $:=$ ⟨ Proof of the Schröder-Bernstein theorem. A.1.2 ⟩

$]\!]$

A.7 Table of Deva sections

⟨ $X_2 = X \setminus X_1$. A.1.4.3.1 ⟩ This code is used in section A.1.4.3.
⟨ $\mathrm{ran}((g^{-1})_{|X \setminus X_1}) = Y_2$. A.1.2.3.2 ⟩ This code is used in section A.1.2.3.
⟨ $\mathrm{ran}(f_{|X_1}) = Y_1$. A.1.2.3.1 ⟩ This code is used in section A.1.2.3.
⟨ $f(x) = g^{-1}(y)$. A.1.2.1.2.2 ⟩ This code is used in section A.1.2.1.2.
⟨ $f(x) \in Y_1$. A.1.2.1.2.3 ⟩ This code is used in section A.1.2.1.2.
⟨ $f(x) \notin Y_1$. A.1.2.1.2.4 ⟩ This code is used in section A.1.2.1.2.
⟨ h is injective. A.1.2.1 ⟩ This code is used in section A.1.2.
⟨ h is surjective. A.1.2.3 ⟩ This code is used in section A.1.2.
⟨ $y \in X_2$. A.1.2.1.2.1 ⟩ This code is used in section A.1.2.1.2.
⟨ A theory of maps. A.5 ⟩ This code is used in section A.6.
⟨ Axioms of FOIL (equality). A.2.3 ⟩ This code is used in section A.2.
⟨ Axioms of FOIL (propositional logic). A.2.6 ⟩ This code is used in section A.2.
⟨ Axioms of FOIL (quantifiers). A.2.8 ⟩ This code is used in section A.2.
⟨ Axioms of FOPL (excluded middle). A.3.2 ⟩ This code is used in section A.3.
⟨ Basic types. A.2.1 ⟩ This code is used in section A.2.
⟨ Case: $x, y \in X_1$. A.1.2.1.1 ⟩ This code is used in section A.1.2.1.
⟨ Case: $x, y \in X_2$. A.1.2.1.5 ⟩ This code is used in section A.1.2.1.
⟨ Case: $x \in X_1, y \in X_2$. A.1.2.1.2, A.1.2.1.3 ⟩ This code is used in section A.1.2.1.
⟨ Case: $x \in X_2, y \in X_1$. A.1.2.1.4 ⟩ This code is used in section A.1.2.1.
⟨ Complete lattice. 4.1, 4.2 ⟩ This code is used in section A.6.
⟨ Contradiction ($x \in X_1, y \in X_2$). A.1.2.1.2.5 ⟩ This code is used in section A.1.2.1.2.
⟨ Definitions for a theory of maps. A.5.1, A.5.11, A.5.10, A.5.9, A.5.8, A.5.7, A.5.6, A.5.5, A.5.4, A.5.3, A.5.2 ⟩ This code is used in section A.5.
⟨ Definitions for elementary set theory. A.4.1, A.4.6, A.4.5, A.4.4, A.4.3, A.4.2 ⟩ This code is used in section A.4.
⟨ Derived laws of FOIL (equality). A.2.4 ⟩ This code is used in section A.2.
⟨ Derived laws of FOIL (propositional logic). A.2.7 ⟩ This code is used in section A.2.
⟨ Derived laws of FOPL (negation and refutation). A.3.1 ⟩ This code is used in section A.3.
⟨ Derived quantifiers of FOIL. A.2.9 ⟩ This code is used in section A.2.
⟨ Elementary set theory. A.4 ⟩ This code is used in section A.6.
⟨ First order intuitionistic logic. A.2 ⟩ This code is used in section A.6.
⟨ First order logic. A.3 ⟩ This code is used in section A.6.

⟨ Overloaded equality (term and functional equality). A.2.5 ⟩ This code is used in section A.2.
⟨ Proof of Banach's decomposition theorem. A.1.4 ⟩ This code is used in section A.6.
⟨ Proof of the Knaster-Tarski theorem. 4.3 ⟩ This code is used in section A.6.
⟨ Proof of the Schröder-Bernstein theorem. A.1.2 ⟩ This code is used in section A.6.
⟨ QED (h surjective). A.1.2.3.3 ⟩ This code is used in section A.1.2.3.
⟨ QED (Banach decomposition). A.1.4.3 ⟩ This code is used in section A.1.4.
⟨ QED (Knaster-Tarski). 4.3.4 ⟩ This code is used in section 4.3.
⟨ QED (Schröder-Bernstein). A.1.2.4 ⟩ This code is used in section A.1.2.
⟨ QED, i.e., $x = y$. A.1.2.1.6 ⟩ This code is used in section A.1.2.1.
⟨ Signature of FOIL. A.2.2 ⟩ This code is used in section A.2.
⟨ Type of Banach's decomposition theorem. A.1.1 ⟩ This code is used in section A.1.4.
⟨ $\Phi(\sqcup M)$ belongs to M. 4.3.2 ⟩ This code is used in section 4.3.
⟨ $\Phi(\sqcup M)$ is atmost $\sqcup M$. 4.3.3 ⟩ This code is used in section 4.3.
⟨ Φ has fixpoint X_1. A.1.4.2 ⟩ This code is used in section A.1.4.
⟨ Φ is monotonic. A.1.4.1 ⟩ This code is used in section A.1.4.
⟨ $\sqcup M$ is atmost $\Phi(\sqcup M)$. 4.3.1 ⟩ This code is used in section 4.3.

A.8 Index of variables

$\wedge$: 4, 4.1, 4.3.1, A.1.1, A.1.2, *A.2.2*, A.2.6, A.2.7, A.4.3, A.4.4, A.4.6.
$\sqsubseteq$: 4.1, 4.2, 4.3, 4.3.1, 4.3.2, 4.3.3.
$=$: 4.1, 4.2, 4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4, A.1.1, A.1.2, A.1.2.1, A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.1, A.1.2.1.2.2, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.1.2.5, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5, A.1.2.1.6, A.1.2.3, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.1.2.4, A.1.4, A.1.4.1, A.1.4.2, A.1.4.3, A.1.4.3.1, A.2, A.2.1, A.2.2, *A.2.3*, A.2.4, A.2.5, A.2.6, A.2.7, A.2.8, A.2.9, A.3, A.3.1, A.3.2, A.4, A.4.1, A.4.2, A.4.3, A.4.4, A.4.5, A.4.6, A.5, A.5.1, A.5.2, A.5.3, A.5.4, A.5.5, A.5.6, A.5.7, A.5.8, A.5.9, A.5.10, A.5.11.
$\Rightarrow$: A.1.2.1, *A.2.2*, A.2.6, A.3.1, A.4.3, A.5.4.
$\vee$: A.1.2.1.6, *A.2.2*, A.2.6, A.2.7, A.3.2, A.4.4, A.4.6.
$\mathcal{A}$: A.4.4, A.4.6.
$\bigcup$: A.1.4, A.1.4.2, *A.4.4*, A.4.6.
$\setminus$: A.1.1, A.1.2, A.1.2.1.2, A.1.2.1.2.4, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5, A.1.2.1.6, A.1.2.3.2, A.1.2.3.3, A.1.4, A.1.4.1, A.1.4.3, A.1.4.3.1, *A.4.5*, A.4.6, A.5.7, A.5.11.
$\cup$: A.1.2.3.3, *A.4.4*, A.4.6, A.5.7.
$\in$: 4.1, 4.2, 4.3.1, 4.3.2, A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.1, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5, A.1.2.1.6, *A.4.2*, A.4.3, A.4.4, A.4.5, A.4.6, A.5.2, A.5.3, A.5.7.
$^{-1}$: A.1.2, A.1.2.1.2.2, A.1.2.1.2.4, A.1.2.1.5, A.1.2.3.2, A.1.2.3.3, *A.5.4*.
$\simeq$: A.1.2.4, *A.5.6*.
$\dagger$: A.1.1, A.1.2, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.3.1, A.1.2.3.2, A.1.4, A.1.4.1, A.1.4.3, *A.5.3*, A.5.4, A.5.10.
$\oplus_{(\cdot)}(\cdot)$: A.1.2, *A.5.7*.
ran: A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, *A.5.1*, A.5.3, A.5.5, A.5.7.
$|_{(\cdot)}$: A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, *A.5.2*, A.5.3, A.5.7.
$\subseteq$: A.1.4, A.1.4.1, A.1.4.2, *A.4.3*, A.4.6, A.5.10, A.5.11.
$\neg$: A.1.2.1.2.4, *A.3.1*, A.3.2, A.4.5, A.4.6.
$\forall$: *A.2.2*, A.2.8, A.2.9, A.4.3, A.4.6.
$\forall_2$: *A.2.9*, A.5.4.
anti_sym: 4.1, 4.3.4, A.4.6.
antianti: A.1.4.1, A.5.9.
antimon: A.1.4.1, *A.5.8*, A.5.9, A.5.11.
ATM: *A.5*.
banach: A.1.2, *A.6*.
case_distinction: A.1.2.1.6, A.4.6.
cases: A.1.2.1.6.
casesii: A.1.2.1.6, *A.2.7*.
commutative: A.4.6.
compl_antimon: A.1.4.1, *A.5.11*.
complement: A.1.2.1.6, A.1.4.3.1, *A.4.6*.

complete_lattice: 4.1, 4.2, 4.3.1.
Complete_lattice: A.1.4.2, *A.6*.
complp: *A.4.5*.
composition: A.1.4.1, *A.5.9*.
conj: 4.3.1, 4.3.4, A.1.2.1.6, *A.2.6*.
conjiv: A.1.2.1.2.1, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.1.4.3, *A.2.7*.
corr$_2$: A.1.2.1.2.1, A.1.2.1.2.4.
decomposition: A.1.2, A.1.2.1.2.1, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3.
disj: *A.2.6*.
down: 4.3.1, 4.3.2, A.1.4.3.1.
elim: 4, A.1.2, A.1.2.1.1, A.1.2.1.2.1, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.2.6, A.2.7, A.2.8, A.2.9, A.3.1.
EST: *A.4*.
ex: *A.2.8*.
$\exists$: *A.2.2*, A.2.8, A.2.9, A.4.4, A.5.1.
excluded_middle: *A.3.2*.
exiv: A.1.2, A.1.4.3, *A.2.9*.
$\exists_4$: A.1.1, *A.2.9*.
ext: 4.1, 4.3, A.1.4, *A.4.1*, A.4.2, A.4.4, A.4.5, A.5.1.
extensionality: *A.2.3*.
False: A.1.2.1.2.5, *A.2.2*, A.2.6, A.3.1.
False_elim: A.1.2.1.2.5, *A.2.6*.
feq: *A.2.3*.
FOIL: *A.2*, A.3.
FOPL: *A.3*.
galois: A.1.4.3.1, A.4.6.
generic_lemma$_2$: A.1.2.1.2, A.1.2.1.3, A.1.2.1.4.
hyp: 4.3.1.
imp: A.1.2.1, A.1.2.1.1, *A.2.6*, A.3.1.
in_def: 4.3.1, 4.3.2, A.1.2.1.2.4, *A.4.2*.
inj$_f$: A.1.2, A.1.2.1.1.
inj$_g$: A.1.2, A.1.2.1.2.4, A.1.2.3.2.
inj$_h$: A.1.2.1, A.1.2.4.
injective: A.1.2, A.1.2.1, *A.5.4*, A.5.6.
injective_lemma: A.1.2.1.2.4, A.1.2.3.2, *A.5.4*.
intro: 4, 4.3.1, 4.3.4, A.1.2.1, A.1.2.1.6, A.1.4.3, A.2.6, A.2.7, A.2.8, A.2.9.
inverse: *A.5.4*.
inverse_injective: A.1.2.1.5, *A.5.4*.
isomorphic: A.1.2.4, *A.5.6*.
knaster_tarski: A.1.4.2, *A.6*.
Leibniz: A.1.2.1.1, A.1.2.1.2.1, A.1.2.1.2.2, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.1.5, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.1.4.3.1, *A.2.5*.
Leibniz_feq: *A.2.3*, A.2.4, A.2.5.
Leibniz_teq: *A.2.3*, A.2.4, A.2.5.
lemma$_1$: 4.3.1, 4.3.2, 4.3.4, A.1.2.1.1, A.1.2.1.2.2, A.1.2.1.2.4, A.1.2.1.6, A.1.2.3.1, A.1.2.3.3, A.1.4.1, A.1.4.2.
lemma$_2$: 4.3.2, 4.3.3, A.1.2.1.2.3, A.1.2.1.2.5, A.1.2.1.3, A.1.2.1.6, A.1.2.3.2, A.1.2.3.3, A.1.4.2, A.1.4.3.1.
lemma$_3$: 4.3.3, 4.3.4, A.1.2.1.2.4, A.1.2.1.2.5, A.1.2.1.4, A.1.2.1.6.
lemma$_4$: A.1.2.1.5, A.1.2.1.6.
Library: *A.6*.
local_hyp$_1$: A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.2, A.1.2.1.2.3, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5.
local_hyp$_2$: A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.1, A.1.2.1.2.2, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5.
$\sqcup$: 4.1, 4.2, 4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4.
lub_ub: 4.2, 4.3.1, 4.3.3.
main_hyp: A.1.2.1, A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.2, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5.
mapext: A.1.2.1.2.3, A.1.2.1.2.4, *A.5.3*.
mapext_monotonic: A.1.4.1, *A.5.10*.
mapsum: A.1.2.1.1, A.1.2.1.2.2, A.1.2.1.5, A.1.2.3.3, *A.5.7*.
monanti: A.1.4.1, A.5.9.
monotonic: 4.3, 4.3.1, 4.3.2, A.1.4.1, *A.5.8*, A.5.9, A.5.10.
neg: A.1.2.1.2.2, A.1.2.1.5, A.5.7.
partial_order: 4.1, 4.2, 4.3.1, 4.3.4.
Φ: 4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4, A.1.4, A.1.4.1, A.1.4.2, A.1.4.3.1.
pos: A.1.2.1.1, A.1.2.1.2.2, A.5.7.
prop: 4, 4.1, *A.2.1*, A.2.2, A.2.3, A.2.6, A.2.7, A.2.8, A.2.9, A.3.1, A.3.2, A.4.1, A.4.2, A.5.6, A.5.8, A.5.9.
refl: 4.1, 4.2, A.4.6.
refl_eq: A.1.2.3.1, A.1.2.3.2, A.1.4.3, *A.2.5*.
refl_feq: *A.2.3*, A.2.4, A.2.5.
refl_teq: A.1.4.3.1, *A.2.3*, A.2.4, A.2.5.
refutation: A.1.2.1.2.5, *A.3.1*.
restriction: *A.5.2*.
s_1: A.2.9.
s_2: A.2.9.
s_3: A.2.9.
s_4: A.2.9.
sb: *A.6*.

set: 4.1, 4.2, A.1.1, A.1.2, A.1.4, A.1.4.2, *A.4.1*, A.4.2, A.4.3, A.4.4, A.4.5, A.4.6, A.5.2, A.5.3, A.5.4, A.5.7.
set_empty: *A.4.6*.
set_total: A.1.2.3.3, *A.4.6*, A.5.5, A.5.7.
seteq: *A.4.3*.
sort: 4.1, A.1.1, A.1.2, A.1.4, *A.2.1*, A.2.2, A.2.3, A.2.4, A.2.8, A.2.9, A.4.1, A.4.2, A.4.3, A.4.4, A.4.5, A.4.6, A.5.1, A.5.2, A.5.3, A.5.4, A.5.5, A.5.6, A.5.7, A.5.8, A.5.9, A.5.10, A.5.11.
subset_cl: A.1.4.2, *A.4.6*.
subset_po: A.1.4.2, *A.4.6*.
subsetp: *A.4.3*.
$surj_h$: A.1.2.3, A.1.2.4.
surjective: A.1.2.3.3, *A.5.5*, A.5.6, A.5.7.
sym_eq: A.1.2.1.2.1, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.1.4, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.1.4.3.1, *A.2.5*.
sym_feq: *A.2.4*, A.2.5.
sym_teq: *A.2.4*, A.2.5.
teq: *A.2.3*.
total: A.1.2.3.3, A.4.6.
trans: 4.1, 4.3.1, A.4.6.
trans_eq: *A.2.5*.
trans_feq: *A.2.4*, A.2.5.
trans_teq: *A.2.4*, A.2.5.
$\mathcal{U}$: *4.1*.
union: A.1.2.3.3, *A.4.6*.
Unionp: *A.4.4*.
unionp: *A.4.4*.
univ: *A.2.8*.
univii: A.1.2.1, A.1.2.1.1, *A.2.9*.
up: 4.2, 4.3.1, A.1.2.1.2.4.
X_1: A.1.1, A.1.2, A.1.2.1.1, A.1.2.1.2, A.1.2.1.2.3, A.1.2.1.3, A.1.2.1.4, A.1.2.1.5, A.1.2.1.6, A.1.2.3.1, A.1.2.3.2, A.1.2.3.3, A.1.4, A.1.4.2, A.1.4.3, A.1.4.3.1.
x_1: A.2.9.
X_2: A.1.1, A.1.2, A.1.2.1.2.1, A.1.2.1.2.4, A.1.2.3.2, A.1.4.3, A.1.4.3.1.
x_2: A.2.9.
x_3: A.2.9.
x_4: A.2.9.
Y_1: A.1.1, A.1.2, A.1.2.1.2.3, A.1.2.1.2.4, A.1.2.3.1, A.1.2.3.3, A.1.4.3.
Y_2: A.1.1, A.1.2, A.1.2.1.2.4, A.1.2.3.2, A.1.2.3.3, A.1.4.3.

B Development of a revision management system

B.1 Specifications and refinements

B.1.1 Introduction and overview

To foster clarity and understanding of the technical issues the development is described in an incremental style. After introducing some basic sorts and constants, a simple (but not simplistic) abstract model of a revision control system is specified. All further parts of the development are based on this abstract specification. A central development issue concerns the efficient storage of the various revisions. Therefore the abstract specification is reified twice. The first data reification introduces the so-called "delta-technique" which is based on the realization that subsequent revisions, while often huge files, usually do not differ very much. The idea then is to store just the differences (as tables of lines) between revisions, instead of the revisions themselves. The second data reification then summarizes the description of all these differences into a single global array, to speed up computation time. The final part of the development describes two extensions of the revision control system: the introduction of user-held locks on revisions and the generalisation of the system operations towards robustness. This leads to the following overall structure of the development.

$\langle$ The devlopment of a revision management system. B.1.1 $\rangle \equiv$
$[\![$ $\langle$ Basic sorts and constants. B.1.2 $\rangle$
; $\langle$ Abstract specification of the kernel system. B.1.3 $\rangle$
; $\langle$ 1st. data reification (delta technique). B.1.4 $\rangle$
; $\langle$ 2nd. data reification (global array). B.1.5 $\rangle$
; $\langle$ Extension by user-held locks. B.1.6 $\rangle$
; $\langle$ Extension to robust operations. B.1.7 $\rangle$
$]\!]$

B.1.2 Basic sorts and constants

Rid
is the set of revision identifiers. Files, imported from the library, in general are sequences of lines which are themselves sequences of characters.

Finally, there is a maximum on the number of revisions allowed.

$\langle$ Basic sorts and constants. B.1.2 $\rangle \equiv$
$[\![\ Rid \quad : sort$
$;\ RevMax \quad : nat$
$;\ RevMax_{pos} \ : RevMax \geq 1$
$]\!]$

B.1.3 Abstract specification of the kernel system

The kernel system is given by its state and invariant, its individual operations, their combination into a module, and the proof of their validity.

$\langle$ Abstract specification of the kernel system. B.1.3 $\rangle \equiv$
$[\![$ $\langle$ Kernel state and invariant. B.1.3.1 $\rangle$
; $\langle$ Kernel operations. B.1.3.2 $\rangle$

; ⟨ Module assembly. B.1.3.7 ⟩
; ⟨ Validity of the kernel specification. B.1.3.8 ⟩
$]\!]$

B.1.3.1. The state of the kernel system consists of an association mapping from revision identifiers to actual revisions and of a tree of revision identifiers. Constructors and destructors for this state are defined as usual. The invariant captures the following requirements:

- The revision identifiers in the revision tree correspond to those in the association map,
- no revision identifiers occurs twice in the revision tree, and
- the number of revisions is less than $RevMax$.

⟨ Kernel state and invariant. B.1.3.1 ⟩ $\equiv$

$$
\begin{array}{lll}
[\![\ K_{st} & := (Rid \xrightarrow{m} File) \otimes tree(Rid) & \\
;\ K & := \langle\!| \ mk & := (\mapsto) \therefore [(Rid \xrightarrow{m} File); tree(Rid) \vdash K_{st}\,] \\
 & \quad ,\ cont & := [\, rg : K_{st} \vdash sel_1(rg)\,] \\
 & \quad ,\ dep & := [\, rg : K_{st} \vdash sel_2(rg)\,] \\
 & \quad |\!\rangle & \\
;\ K_{inv} & := [\, rg\ :\ K_{st} & \\
 & \quad \vdash \mathbf{dom}\, K\,\textbf{.}\,cont\ (rg) = info(K\textbf{.}\ dep\ (rg)) & \\
 & \qquad \wedge\ nodup(K\textbf{.}\ dep\ (rg)) & \\
 & \qquad \wedge\ RevMax \geq \mathbf{card}(\mathbf{dom}\, K\,\textbf{.}\,cont\ (rg)) & \\
 & \qquad \wedge\ \forall[\, r : Rid\ \vdash r \in \mathbf{dom}\, K\,\textbf{.}\,cont\ (rg) \Rightarrow wff_F(K\textbf{.}\ cont\ (rg) \nabla r)\,] & \\
 & \quad] & \\
]\!] & &
\end{array}
$$

B.1.3.2. There are four operations in the kernel system. Their precise behaviour is described below.

⟨ Kernel operations. B.1.3.2 ⟩ $\equiv$

$$
\begin{array}{lll}
K_{op} := \langle\!| & RESET & := \langle \text{Reset the system. B.1.3.3} \rangle \\
, & OPEN & := \langle \text{Open the system with a root file. B.1.3.4} \rangle \\
, & CHECKIN & := \langle \text{Check in a file. B.1.3.5} \rangle \\
, & CHECKOUT & := \langle \text{Check out a file. B.1.3.6} \rangle \\
|\!\rangle & &
\end{array}
$$

B.1.3.3. The system is reset by setting the state components to the empty mapping and the empty tree. There are neither input nor output parameters.

⟨ Reset the system. B.1.3.3 ⟩ $\equiv$

$[\_\_ : \mathbf{void}\, ; \overleftarrow{rg} : K_{st}$
$\vdash \langle\!| \; pre \;\; := \; true$
$\quad , post \;\; := [\_\_ : \mathbf{void}\, ; rg : K_{st}$
$\qquad\qquad \vdash rg = K\,.\, mk\; (\langle\rangle, \tau)$
$\qquad\qquad]$
$\quad |\!\rangle$
$]$
$\quad \therefore op_{st}(K_{st})$

B.1.3.4. A new revision control system is opened with a revision and a revision identifier. Note that the below specification requires the system to have been previously empty.

$\langle$ Open the system with a root file. B.1.3.4 $\rangle \equiv$
$[\, in \; : (File \otimes Rid); \overleftarrow{rg} : K_{st}$
$\vdash [\, f \;\; := sel_1(in); \;\; r \;\; := sel_2(in)$
$\quad \vdash \langle\!| \; pre \;\; := \mathbf{dom}\, K\,.\, cont\; (\overleftarrow{rg}) = \{\,\} \wedge \mathit{wff}_F(f)$
$\qquad , post \;\; := [\_\_ : \mathbf{void}\, ; rg : K_{st}$
$\qquad\qquad \vdash rg = K\,.\, mk\; (\langle r \mapsto f\rangle, node(r, \langle\rangle))$
$\qquad\qquad]$
$\qquad |\!\rangle$
$\quad]$
$]$
$\quad \therefore op_{in}(File \otimes Rid, K_{st})$

B.1.3.5. Given a non-empty system, a new revision, i.e. a file, can be checked in by indicating the revision identifier of the previous revision and providing a new revision identifier.

$\langle$ Check in a file. B.1.3.5 $\rangle \equiv$
$[\, in \; : \; File \otimes (Rid \otimes Rid); \overleftarrow{rg} : K_{st}$
$\vdash [\, f \qquad := sel_1(in)$
$\quad ; \; prev \;\; := sel_1(sel_2(in))$
$\quad ; \; new \;\; := sel_2(sel_2(in))$
$\quad \vdash \langle\!| \; pre \;\; := prev \in \mathbf{dom}\, K\,.\, cont\; (\overleftarrow{rg})$
$\qquad\qquad \wedge\; new \notin \mathbf{dom}\, K\,.\, cont\; (\overleftarrow{rg})$
$\qquad\qquad \wedge\; RevMax\; ¿\; \mathbf{card}(\mathbf{dom}\, K\,.\, cont\; (\overleftarrow{rg}))$
$\qquad\qquad \wedge\; \mathit{wff}_F(f)$
$\qquad , post \;\; := [\_\_ : \mathbf{void}\, ; rg : K_{st}$
$\qquad\qquad \vdash rg = K\,.\, mk\; ((new \mapsto f) \odot K\,.\, cont\; (\overleftarrow{rg})$
$\qquad\qquad\qquad\qquad , insert(prev, new, K\,.\, dep\; (\overleftarrow{rg})))$
$\qquad\qquad]$
$\qquad |\!\rangle$
$\quad]$
$]$

$\therefore op_{in}(File \otimes (Rid \otimes Rid), K_{st})$

B.1.3.6. A revision can be checked through its revision identifier. This operation does not change the state of the system.

$\langle$ Check out a file. B.1.3.6 $\rangle \equiv$

$$
\begin{array}{l}
[\, r \,:\, Rid \,;\, \overleftarrow{rg} \,:\, K_{st} \\
\vdash \langle\!| \, pre \;\; := r \in \mathbf{dom}\, K\,.\,cont\,(\overleftarrow{rg}) \\
\quad , \, post \; := [\, f \,:\, File \,;\, rg : \, K_{st} \\
\qquad\qquad\quad \vdash f = K.\, cont\,(\overleftarrow{rg}) \nabla\, r \wedge rg = \overleftarrow{rg} \\
\qquad\qquad\quad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op(Rid, File, K_{st})
\end{array}
$$

B.1.3.7. The invariant and the operations are combined into a module.

$\langle$ Module assembly. B.1.3.7 $\rangle \equiv$

$$
\begin{array}{l}
[\![\, K_{mod} \; := \langle\!| \, inv \;\; := \;\; K_{inv} \\
\qquad\qquad\quad , \, ops \;\; := \langle K_{op}.RESET \rangle \odot \langle K_{op}.OPEN \rangle \\
\qquad\qquad\qquad\qquad \odot \langle K_{op}.CHECKIN \rangle \odot \langle K_{op}.CHECKOUT \rangle \\
\qquad\qquad\quad |\!\rangle \\
]\!]
\end{array}
$$

B.1.3.8. The proof obligations are proven separately for each operation and then combined to yield the validity of the module.

$\langle$ Validity of the kernel specification. B.1.3.8 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\; RESET_{val} & := & \langle \text{ Proof of } val\_op\,(K_{op}.RESET, K_{mod}.inv).\ \text{B.2.1} \rangle \\
 & & \quad \therefore val\_op(K_{op}.RESET, K_{mod}.inv) \\
; \; OPEN_{val} & := & \langle \text{ Proof of } val\_op\,(K_{op}.OPEN, K_{mod}.inv).\ \text{B.2.2} \rangle \\
 & & \quad \therefore val\_op(K_{op}.OPEN, K_{mod}.inv) \\
; \; CHECKOUT_{val} & := & \langle \text{ Proof of } val\_op\,(K_{op}.CHECKOUT, K_{mod}.inv).\ \text{B.2.3} \rangle \\
 & & \quad \therefore val\_op(K_{op}.CHECKOUT, K_{mod}.inv) \\
; \; CHECKIN_{val} & := & \langle \text{ Proof of } val\_op\,(K_{op}.CHECKIN, K_{mod}.inv).\ \text{B.2.4} \rangle \\
 & & \quad \therefore val\_op(K_{op}.CHECKIN, K_{mod}.inv) \\
; \; K\_valid & := & \langle\!| \, RESET_{val}\,,\, OPEN_{val}\,,\, CHECKIN_{val}\,,\, CHECKOUT_{val} \,|\!\rangle \\
 & & \;\; \setminus \, val\_assembly_4 \\
 & & \quad \therefore mod\_valid(K_{mod}) \\
]\!]
\end{array}
$$

B.1.4 Data reification to line-based deltas

The first data reification introduces the so-called "delta-technique" which is based on the realization that subsequent revisions, while often huge files, usually do not differ very much. The idea then is to store just the differences (as tables of lines) between revisions, instead of the revisions themselves.

The description of this development is of the same overall structure as the previous step, except that, in addition to the usual components, a retrieve function is specified according to which the data reification is shown to be valid.

⟨ 1st. data reification (delta technique). B.1.4 ⟩ $\equiv$
$[\![$ ⟨ State and invariant (delta technique). B.1.4.1 ⟩
; ⟨ Retrieve function (delta technique $\rightarrow$ files). B.1.4.2 ⟩
; ⟨ Operations (delta technique). B.1.4.3 ⟩
; ⟨ Module assembly (delta technique). B.1.4.8 ⟩
; ⟨ Validity of the specification (delta technique). B.1.4.9 ⟩
; ⟨ Validity of the data reification (delta technique). B.1.4.10 ⟩
$]\!]$

B.1.4.1. As a major difference to the previous section files are now described as line-based deltas, i.e. sequences of modification commands. The invariant requires in addition that all these deltas define meaningful modifications. The reader is referred to the context *Deltas* for the definition of the predicate *ok_delta*.

⟨ State and invariant (delta technique). B.1.4.1 ⟩ $\equiv$

$$
\begin{array}{l}
[\![\ D_{st} \ := (Rid \xrightarrow{m} \Delta(Line)) \otimes tree(Rid) \\
;\ D \quad := \{\!|\ mk \ := (\mapsto) \therefore [(Rid \xrightarrow{m} \Delta(Line)); tree(Rid) \vdash D_{st}\,] \\
\qquad\quad ,\ cont \ := [\ rg : D_{st} \ \vdash sel_1(rg)\,] \\
\qquad\quad ,\ dep \ := [\ rg : D_{st} \ \vdash sel_2(rg)\,] \\
\qquad\quad |\!\} \\
;\ D_{inv} \ := [\ rg \ : \ D_{st} \\
\qquad\quad \vdash \mathbf{dom}\, D\,.\,cont\,(rg) = info(D.\,dep\,(rg)) \\
\qquad\quad\ \wedge\ nodup(D.\,dep\,(rg)) \\
\qquad\quad\ \wedge\ RevMax \geq \mathbf{card}(\mathbf{dom}\, D\,.\,cont\,(rg)) \\
\qquad\quad\ \wedge\ \forall[\ r \ : \ Rid\,;\ del \ := \ D\,.\,cont\,(rg) \nabla\, r \\
\qquad\qquad \vdash r \in \mathbf{dom}\, D\,.\,cont\,(rg) \Rightarrow (wff_{\Delta}(del) \wedge wff_{F}(changed(del))) \\
\qquad\qquad] \\
\qquad\quad] \\
]\!]
\end{array}
$$

B.1.4.2. The retrieve function is defined in two steps: first, an auxiliary function ($retr\_rev_D$) is defined which constructs the actual file associated to a revision identifier r by applying all the deltas from the root of the revision tree to r. Second, in the actual retrieve function, this function is restricted to the available revisions. Note that the second component of the state remains unchanged in this data reification.

⟨ Retrieve function (delta technique $\rightarrow$ files). B.1.4.2 ⟩ $\equiv$

$$
\begin{array}{l}
[\![\ retr\_rev_D \ := [\ s \ ?\ sort\,;\ co \ : (Rid \xrightarrow{m} \Delta(s));\ dp \ : \ tree\,(Rid);\ r \ : \ Rid \\
\qquad\qquad \vdash \langle\rangle \oplus_s (co * init\_path(r, dp)) \\
\qquad\qquad]
\end{array}
$$

$$
\begin{array}{ll}
;\ retr_D & := [\ rg\ :\ D_{st}\ ;\ retr\_abs\ :=\ retr\_rev_D\ (D\textbf{.}\ cont\ (rg), D\textbf{.}\ dep\ (rg)) \\
& \quad \vdash K\ \textbf{.}\ mk\ (atm(retr\_abs, \mathbf{dom}\ D\ \textbf{.}\ cont\ (rg)), D\textbf{.}\ dep\ (rg)) \\
& \quad] \\
\rrbracket &
\end{array}
$$

B.1.4.3. For each abstract operation, there exists a corresponding concrete operation.

⟨ Operations (delta technique). B.1.4.3 ⟩ ≡

$$
\begin{array}{lll}
D_{op}\ := \langle\!| & RESET & := \langle \text{Reset the system (delta technique). B.1.4.4} \rangle \\
, & OPEN & := \langle \text{Open the system with a root file (delta technique). B.1.4.5} \rangle \\
, & CHECKIN & := \langle \text{Check in a file (delta technique). B.1.4.6} \rangle \\
, & CHECKOUT & := \langle \text{Check out a file (delta technique). B.1.4.7} \rangle \\
|\!\rangle & &
\end{array}
$$

B.1.4.4. The system is reset by setting the state components to the empty mapping and the empty tree. There are neither input nor output parameters.

⟨ Reset the system (delta technique). B.1.4.4 ⟩ ≡

$$
\begin{array}{l}
[\_\!\_ : \mathbf{void}\,;\overleftarrow{rg}\ :\ D_{st} \\
\vdash \langle\!|\ pre\ \ := \ true \\
\quad ,\ post\ := [\_\!\_ : \mathbf{void}\,; rg :\ D_{st} \\
\qquad\qquad \vdash rg = D\textbf{.}\ mk\ (\langle\rangle, \tau) \\
\qquad\qquad] \\
\quad |\!\rangle \\
] \\
\quad \therefore\ op_{st}(D_{st})
\end{array}
$$

B.1.4.5. A new revision control system is opened with a revision and a revision identifier. In contrast to the abstract specification, the revision is represented as a delta. In this simple case, the delta consists of a single delta-unit which describes the insertion of the new revision before the first line of the empty sequences (which is used as starting point).

⟨ Open the system with a root file (delta technique). B.1.4.5 ⟩ ≡

$$
\begin{array}{l}
[\ in\ :\ File \otimes Rid\ ;\overleftarrow{rg}\ :\ D_{st} \\
\vdash [\ f\ := sel_1(in);\ newr\ := sel_2(in) \\
\quad \vdash \langle\!|\ pre\ \ := \mathbf{dom}\ D\ \textbf{.}\ cont\ (\overleftarrow{rg}) = \{\,\} \wedge wff_F(f) \\
\qquad ,\ post\ := [\_\!\_ : \mathbf{void}\,; rg :\ D_{st} \\
\qquad\qquad \vdash rg = D\textbf{.}\ mk\ (\langle newr \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle, node(newr, \langle\rangle)) \\
\qquad\qquad] \\
\qquad |\!\rangle \\
\quad] \\
] \\
\quad \therefore\ op_{in}(File \otimes Rid, D_{st})
\end{array}
$$

B.1.4.6. The check in operation has to compute the difference between the previous revision and the new revision. In order to compute this difference, the previous revision has to be constructed first. Otherwise the specification corresponds to the abstract one.

⟨ Check in a file (delta technique). B.1.4.6 ⟩ ≡

$$
\begin{array}{l}
[\; in : File \otimes (Rid \otimes Rid);\ \overleftarrow{rg} : D_{st} \\
\vdash [\; f := sel_1(in);\ prev := sel_1(sel_2(in));\ new := sel_2(sel_2(in)) \\
\quad \vdash \langle\!| \; pre := prev \in \mathbf{dom}\, D\,.\, cont\,(\overleftarrow{rg}) \\
\qquad\qquad\qquad \wedge \neg new \in \mathbf{dom}\, D\,.\, cont\,(\overleftarrow{rg}) \\
\qquad\qquad\qquad \wedge\ RevMax \text{ ¿ } \mathbf{card}(\mathbf{dom}\, D\,.\, cont\,(\overleftarrow{rg})) \\
\qquad\qquad\qquad \wedge\ \mathit{wff}_F(f) \\
\qquad ,\ post := [\, \_ : \mathbf{void};\ rg : D_{st} \\
\qquad\qquad\qquad \vdash [\; del := \mathit{diff}\,(retr\_rev_D(D.\, cont\,(\overleftarrow{rg}), D.\, dep\,(\overleftarrow{rg}), prev), f) \\
\qquad\qquad\qquad\quad \vdash rg = D.\, mk\,((new \mapsto del) \odot D.\, cont\,(\overleftarrow{rg}), insert(prev, new, D.\, dep\,(\overleftarrow{rg}))) \\
\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad] \\
\qquad |\!\rangle \\
\quad] \\
] \\
\quad \therefore\ op_{in}(File \otimes (Rid \otimes Rid), D_{st})
\end{array}
$$

B.1.4.7. The check out operation computes the actual revision from the deltas based on the information present in the current revision tree.

⟨ Check out a file (delta technique). B.1.4.7 ⟩ ≡

$$
\begin{array}{l}
[\; r : Rid;\ \overleftarrow{rg} : D_{st} \\
\vdash \langle\!| \; pre := r \in \mathbf{dom}\, D\,.\, cont\,(\overleftarrow{rg}) \\
\quad ,\ post := [\; f : File;\ rg : D_{st} \\
\qquad\qquad\quad \vdash f = retr\_rev_D(D.\, cont\,(\overleftarrow{rg}), D.\, dep\,(\overleftarrow{rg}), r) \\
\qquad\qquad\qquad \wedge\ rg = \overleftarrow{rg} \\
\qquad\qquad\quad] \\
\quad |\!\rangle \\
] \\
\quad \therefore\ op(Rid, File, D_{st})
\end{array}
$$

B.1.4.8. The module assembly proceeds as usual.

⟨ Module assembly (delta technique). B.1.4.8 ⟩ ≡

$$
\begin{array}{l}
D_{mod} := \langle\!| \; inv := D_{inv} \\
\qquad\quad ,\ ops := \langle D_{op}.RESET \rangle \odot \langle D_{op}.OPEN \rangle \\
\qquad\qquad\qquad \odot \langle D_{op}.CHECKIN \rangle \odot \langle D_{op}.CHECKOUT \rangle \\
\qquad\quad |\!\rangle
\end{array}
$$

B.1.4.9.

$\langle$ Validity of the specification (delta technique). B.1.4.9 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\ \mathit{wff\_lemma} & : & [\ rg\ ?\ D_{st}\ ;\ r\ ?\ Rid \\
 & & \vdash [\ D_{inv}\ (rg) \\
 & & \quad ;\ r \in \mathbf{dom}\ D\ .\ cont\ (rg) \\
 & & \quad \vdash \mathit{wff}_F(retr\_rev_D(D.\ cont\ (rg), D.\ dep\ (rg), r)) \\
 & & \quad] \\
 & &] \\
;\ D\_RESET_{val} & := & \langle \text{Proof of } val\_op\ (D_{op}.RESET, D_{mod}.inv).\ \text{B.3.1} \rangle \\
 & & \quad \therefore val\_op(D_{op}.RESET, D_{mod}.inv) \\
;\ D\_OPEN_{val} & := & \langle \text{Proof of } val\_op\ (D_{op}.OPEN, D_{mod}.inv).\ \text{B.3.2} \rangle \\
 & & \quad \therefore val\_op(D_{op}.OPEN, D_{mod}.inv) \\
;\ D\_CHECKIN_{val} & := & \langle \text{Proof of } val\_op\ (D_{op}.CHECKIN, D_{mod}.inv).\ \text{B.3.4} \rangle \\
 & & \quad \therefore val\_op(D_{op}.CHECKIN, D_{mod}.inv) \\
;\ D\_CHECKOUT_{val} & := & \langle \text{Proof of } val\_op\ (D_{op}.CHECKOUT, D_{mod}.inv).\ \text{B.3.3} \rangle \\
 & & \quad \therefore val\_op(D_{op}.CHECKOUT, D_{mod}.inv) \\
;\ D\_valid & := & \langle\!| \ D\_RESET_{val}\ ,\ D\_OPEN_{val}\ ,\ D\_CHECKIN_{val}\ ,\ D\_CHECKOUT_{val} \ |\!\rangle \\
 & & \ \backslash\ val\_assembly_4 \\
 & & \quad \therefore mod\_valid(D_{mod}) \\
]\!] & &
\end{array}
$$

B.1.4.10. The proof of validity for the actual reification assumes the existence of an inverse of the retrieve function $retr_D$. This inverse amounts essentially to an algorithm that transform an abstract revision group into an equivalent one using the delta technique. While the existence of such an inverse is intuitively plausible, it would repeatedly checks in files using the concrete check in operation, its formal construction is not attempted here, because of the complexity of this task.

$\langle$ Validity of the data reification (delta technique). B.1.4.10 $\rangle \equiv$

$$
\begin{array}{l}
[\![\langle \text{Assumed inverse of retrieve function. B.1.4.11} \rangle \\
;\langle \text{Projection lemmas. B.4.1} \rangle \\
;\langle \text{Retrieve lemma. B.4.1.1} \rangle \\
;\ val\_retr_D\ := \langle \text{Proof of } val\_retr\ (D_{mod}.inv, K_{mod}.inv, retr_D).\ \text{B.4.2} \rangle \\
\qquad\qquad \therefore val\_retr(D_{mod}.inv, K_{mod}.inv, retr_D) \\
;\langle \text{Proof of the operation reification conditions B.1.4.12} \rangle \\
;\ reif_D \quad := \langle\!| \ module \quad := \langle\!| \ concrete\ :=\ D\_valid \\
\qquad\qquad\qquad\qquad\qquad\quad ,\ abstract\ :=\ K\_valid \\
\qquad\qquad\qquad\qquad\qquad\quad |\!\rangle \\
\qquad\qquad ,\ retrieval \quad :=\ val\_retr_D \\
\qquad\qquad ,\ reification\ :=\ D_{reif} \\
\qquad\qquad |\!\rangle \\
\qquad\qquad\quad \therefore D_{mod} \sqsubseteq_{retr_D} K_{mod} \\
]\!]
\end{array}
$$

B.1.4.11.

$\langle$ Assumed inverse of retrieve function. B.1.4.11 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\; convert & : [K_{st} \vdash D_{st}\,] \\
; \; prop\_convert & : [\; rg \;?\; K_{st} \\
& \quad \vdash \langle\!| \; invar \quad := [\, K_{mod} \,.\, inv\,(rg) \vdash D_{mod} .\, inv\,(convert(rg))\,] \\
& \quad\quad , \; inverse \; := \; retr_D\,(convert(rg)) = rg \\
& \quad\quad |\!\rangle \\
& \quad] \\
]\!]
\end{array}
$$

B.1.4.12.

$\langle$ Proof of the operation reification conditions B.1.4.12 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\; D\_RESET_{reif} & := \langle \text{Proof of operation reification } (RESET). \text{ B.4.3} \rangle \\
& \quad\quad \therefore D_{op}.RESET \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\, RESET \\
; \; D\_OPEN_{reif} & := \langle \text{Proof of operation reification } (OPEN). \text{ B.4.4} \rangle \\
& \quad\quad \therefore D_{op}.OPEN \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\, OPEN \\
; \; D\_CHECKIN_{reif} & := \langle \text{Proof of operation reification } (CHECKIN). \text{ B.4.6} \rangle \\
& \quad\quad \therefore D_{op}.CHECKIN \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\, CHECKIN \\
; \; D\_CHECKOUT_{reif} & := \langle \text{Proof of operation reification } (CHECKOUT). \text{ B.4.5} \rangle \\
& \quad\quad \therefore D_{op}.CHECKOUT \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\, CHECKOUT \\
; \; D_{reif} & := \langle\!|\langle\!|\langle\!| \; def\_oplist\_reif \,.\, sing \,.\, up\,(D\_RESET_{reif}) \\
& \quad\quad\quad \therefore \langle D_{op}.RESET \rangle \sqsubseteq^{oplist}_{D_{mod}.inv, retr_D} \langle K_{op}.RESET \rangle \\
& \quad\quad , \; def\_oplist\_reif \,.\, sing \,.\, up\,(D\_OPEN_{reif}) \\
& \quad\quad\quad \therefore \langle D_{op}.OPEN \rangle \sqsubseteq^{oplist}_{D_{mod}.inv, retr_D} \langle K_{op}.OPEN \rangle \\
& \quad\quad |\!\rangle \\
& \quad\quad {}^{\backslash}\, and.\, in \,{}^{\backslash}\, def\_oplist\_reif \,.\, join \,.\, up \\
& \quad , \; def\_oplist\_reif \,.\, sing \,.\, up\,(D\_CHECKIN_{reif}) \\
& \quad\quad\quad \therefore \langle D_{op}.CHECKIN \rangle \sqsubseteq^{oplist}_{D_{mod}.inv, retr_D} \langle K_{op}.CHECKIN \rangle \\
& \quad |\!\rangle \\
& \quad {}^{\backslash}\, and.\, in \,{}^{\backslash}\, def\_oplist\_reif \,.\, join \,.\, up \\
& , \; def\_oplist\_reif \,.\, sing \,.\, up\,(D\_CHECKOUT_{reif}) \\
& \quad\quad \therefore \langle D_{op}.CHECKOUT \rangle \sqsubseteq^{oplist}_{D_{mod}.inv, retr_D} \langle K_{op}.CHECKOUT \rangle \\
& |\!\rangle \\
& {}^{\backslash}\, and.\, in \,{}^{\backslash}\, def\_oplist\_reif \,.\, join \,.\, up \\
& \quad \therefore D_{mod}.ops \sqsubseteq^{oplist}_{D_{mod}.inv, retr_D} K_{mod}.\, ops \\
]\!]
\end{array}
$$

B.1.5 Data reification to global array

The second data reification summarizes the description of all file differences into a single global array, to speed up computation time. On the other hand, because of the somewhat technical indexing operations, the specification becomes somewhat low-level and very hard to read.

$\langle$ 2nd. data reification (global array). B.1.5 $\rangle \equiv$

$[\![\langle$ State and invariant (global array). B.1.5.1 $\rangle$

; $\langle$ Retrieve function (global array $\rightarrow$ delta technique). B.1.5.3 $\rangle$

; $\langle$ Operations (global array). B.1.5.4 $\rangle$

; $\langle$ Module assembly (global array). B.1.5.9 $\rangle$

; $\langle$ Validity of the specification (global array). B.1.5.10 $\rangle$

; $\langle$ Validity of the data reification (global array). B.1.5.11 $\rangle$

$]\!]$

B.1.5.1.

$\langle$ State and invariant (global array). B.1.5.1 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\ A_{st} & := (Rid \xrightarrow{m} \Delta(nat) \otimes tree(Rid)) \otimes File & \\
;\ A & := \langle\!| \ mk & := [\ cont\ : (Rid \xrightarrow{m} \Delta(nat));\ dep\ :\ tree\ (Rid);\ arr\ :\ File \\
 & & \quad \vdash (cont \mapsto dep) \mapsto arr \\
 & & \quad] \\
 & \quad ,\ cont & := [\ rg : A_{st}\ \vdash sel_1(sel_1(rg))\ ] \\
 & \quad ,\ dep & := [\ rg : A_{st}\ \vdash sel_2(sel_1(rg))\ ] \\
 & \quad ,\ arr & := [\ rg : A_{st}\ \vdash sel_2(rg)\ ] \\
 & \quad |\!\rangle & \\
;\ A_{inv} & := \langle \text{Invariant (global array). B.1.5.2} \rangle & \\
]\!] & &
\end{array}
$$

B.1.5.2.

$\langle$ Invariant (global array). B.1.5.2 $\rangle \equiv$

$$
\begin{array}{l}
[\ rg\ :\ A_{st} \\
\vdash \mathbf{dom}\ A\ .\ cont\ (rg) = info(A.\ dep\ (rg)) \\
\quad \wedge\ nodup(A.\ dep\ (rg)) \\
\quad \wedge\ RevMax \geq \mathbf{card}(\mathbf{dom}\ A\ .\ cont\ (rg)) \\
\quad \wedge\ (\forall [\ r\ :\ Rid \qquad\qquad\qquad) \\
\qquad\quad \vdash r \in \mathbf{dom}\ A\ .\ cont\ (rg) \\
\qquad\qquad \Rightarrow wff_{\Delta}(A.\ cont\ (rg) \nabla r) \\
\qquad\quad] \\
\quad \wedge\ \mathbf{dom}\ sam\ (A.\ arr\ (rg)) \\
\quad = \bigcup([\ d : \Delta(nat) \vdash \text{elemsflatten}(ins * d)\ ] *\ \text{rng}\ \ A\ .\ cont\ (rg)) \\
]
\end{array}
$$

B.1.5.3.

$\langle$ Retrieve function (global array $\rightarrow$ delta technique). B.1.5.3 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ retr\_rev_A\ := [\ co\ : (Rid \xrightarrow{m} \Delta(nat));\ dp\ :\ tree\ (Rid);\ arr\ :\ File\ ;\ r\ :\ Rid \\
\qquad\qquad\qquad \vdash sam\ (arr) * retr\_rev_D(co, dp, r) \\
\qquad\qquad\qquad]
\end{array}
$$

$$
\begin{array}{l}
;\ retr_A \quad := [\ rg \quad : \quad A_{st} \\
\qquad\qquad ;\ retr\_abs\ := [\ r\ :\ Rid \\
\qquad\qquad\qquad\qquad \vdash apply\_to\_inserts\ (sam(A.\ arr\ (rg)), A.\ cont\ (rg) \nabla r) \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad \vdash D\ .\ mk\ (atm(retr\_abs, \mathbf{dom}\ A\ .\ cont\ (rg)), A.\ dep\ (rg)) \\
\qquad\qquad] \\
\rrbracket
\end{array}
$$

B.1.5.4.

$\langle$ Operations (global array). B.1.5.4 $\rangle \equiv$

$$
\begin{array}{l}
A_{op}\ := \langle\!| \ RESET \quad := \langle \text{Reset the system (global array). B.1.5.5} \rangle \\
\qquad ,\ OPEN \quad := \langle \text{Open the system with a root file (global array). B.1.5.6} \rangle \\
\qquad ,\ CHECKIN \quad := \langle \text{Check in a file (global array). B.1.5.7} \rangle \\
\qquad ,\ CHECKOUT\ := \langle \text{Check out a file (global array). B.1.5.8} \rangle \\
\qquad |\!\rangle
\end{array}
$$

B.1.5.5.

$\langle$ Reset the system (global array). B.1.5.5 $\rangle \equiv$

$$
\begin{array}{l}
[\_ : \mathbf{void}\ ; \overleftarrow{rg}\ :\ A_{st} \\
\vdash \langle\!|\ pre \quad :=\ true \\
\quad ,\ post\ := [\_ : \mathbf{void}\ ; rg\ :\ A_{st} \\
\qquad\qquad \vdash rg = A.\ mk\ (\langle\rangle, \tau, \langle\rangle) \\
\qquad\qquad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op_{st}(A_{st})
\end{array}
$$

B.1.5.6.

$\langle$ Open the system with a root file (global array). B.1.5.6 $\rangle \equiv$

$$
\begin{array}{l}
[\ in\ :\ File \otimes Rid\ ; \overleftarrow{rg}\ :\ A_{st} \\
\vdash [\ f\ := sel_1(in);\ r\ := sel_2(in) \\
\quad \vdash \langle\!|\ pre \quad := \mathbf{dom}\ A\ .\ cont\ (\overleftarrow{rg}) = \{\ \} \\
\qquad ,\ post\ := [\_ : \mathbf{void}\ ; rg\ :\ A_{st} \\
\qquad\qquad \vdash rg \\
\qquad\qquad\quad = A.\ mk\ (\langle r \mapsto \langle\ \langle\ 1, count\_up(0, \mathrm{len} f\ \rangle_{\Delta_u}, 0)\rangle\ \rangle, node(r, \langle\rangle), f) \\
\qquad\qquad] \\
\qquad |\!\rangle \\
\quad] \\
] \\
\quad \therefore op_{in}(File \otimes Rid, A_{st})
\end{array}
$$

B.1.5.7.

$\langle$ Check in a file (global array). B.1.5.7 $\rangle \equiv$

$$
\begin{array}{l}
[\, in \;:\; File \otimes (Rid \otimes Rid); \overleftarrow{rg} \;:\; A_{st} \\
\vdash [\, f \;:= sel_1(in);\; prev \;:= sel_1(sel_2(in));\; new \;:= sel_2(sel_2(in)) \\
\quad \vdash \langle\!| \; pre \;\; := prev \in \mathbf{dom}\, A \,.\, cont\,(\overleftarrow{rg}) \\
\qquad\qquad\qquad \wedge \neg(new \in \mathbf{dom}\, A \,.\, cont\,(\overleftarrow{rg})) \\
\qquad\qquad\qquad \wedge\, \mathbf{card}(\mathbf{dom}\, A \,.\, cont\,(\overleftarrow{rg}))\; ¡\; RevMax \\
\qquad , post \;:= [\_ : \mathbf{void}\,; rg :\; A_{st} \\
\qquad\qquad\qquad \vdash [\;\; del \qquad := \; diff \;(retr\_rev_A(A.\, cont\,(\overleftarrow{rg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad A \,.\, dep\,(\overleftarrow{rg}), A.\, arr\,(\overleftarrow{rg}), prev) \\
\qquad\qquad\qquad\qquad\qquad\quad ,f) \\
\qquad\qquad\qquad\;\; ;\; del_{num} \;:= number_{\Delta}(\mathrm{len}\, A \,.\, arr\,(\overleftarrow{rg}), del) \\
\qquad\qquad\qquad\;\; \vdash rg = A.\, mk\,((new \mapsto del_{num}) \odot A.\, cont\,(\overleftarrow{rg}) \\
\qquad\qquad\qquad\quad , insert(prev, new, A.\, dep\,(\overleftarrow{rg})) \\
\qquad\qquad\qquad\quad , A.\, arr\,(\overleftarrow{rg}) +\!\!+ \mathrm{flatten}(ins * del) \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad\qquad\;\;] \\
\qquad\qquad\qquad] \\
\qquad |\!\rangle \\
\quad] \\
] \\
\quad \therefore op_{in}(File \otimes (Rid \otimes Rid), A_{st})
\end{array}
$$

B.1.5.8.

$\langle$ Check out a file (global array). B.1.5.8 $\rangle \equiv$

$$
\begin{array}{l}
[\, r \;:\; Rid\,; \overleftarrow{rg} \;:\; A_{st} \\
\vdash \langle\!| \; pre \;\; := r \in \mathbf{dom}\, A \,.\, cont\,(\overleftarrow{rg}) \\
\quad , post \;:= [\, f \;:\; File\,; rg :\; A_{st} \\
\qquad\qquad\quad \vdash f = retr\_rev_A(A.\, cont\,(\overleftarrow{rg}), \\
\qquad\qquad\qquad A \,.\, dep\,(\overleftarrow{rg}), A.\, arr\,(\overleftarrow{rg}), r) \\
\qquad\qquad\qquad \wedge\, rg = \overleftarrow{rg} \\
\qquad\qquad\quad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op(Rid, File, A_{st})
\end{array}
$$

B.1.5.9.

$\langle$ Module assembly (global array). B.1.5.9 $\rangle \equiv$

$$
\begin{array}{l}
A_{mod} \;:= \langle\!| \; inv \;\; := \; A_{inv} \\
\qquad\qquad , ops \;:= \langle A_{op}.RESET\rangle \odot \langle A_{op}.OPEN\rangle \\
\qquad\qquad\qquad\qquad \odot \langle A_{op}.CHECKIN\rangle \odot \langle A_{op}.CHECKOUT\rangle \\
\qquad\qquad |\!\rangle
\end{array}
$$

B.1.5.10.

⟨ Validity of the specification (global array). B.1.5.10 ⟩ ≡

$$
\begin{array}{lll}
[\![\ A\_RESET_{val} & : & val\_op\ (A_{op}.RESET, A_{mod}.inv) \\
;\ A\_OPEN_{val} & : & val\_op\ (A_{op}.OPEN, A_{mod}.inv) \\
;\ A\_CHECKIN_{val} & : & val\_op\ (A_{op}.CHECKIN, A_{mod}.inv) \\
;\ A\_CHECKOUT_{val} & : & val\_op\ (A_{op}.CHECKOUT, A_{mod}.inv) \\
;\ A\_valid & := & \langle\!| A\_RESET_{val}, A\_OPEN_{val}, A\_CHECKIN_{val}, A\_CHECKOUT_{val} |\!\rangle \\
& & \quad \setminus val\_assembly_4 \\
& & \qquad \therefore mod\_valid(A_{mod}) \\
]\!] & &
\end{array}
$$

B.1.5.11.

⟨ Validity of the data reification (global array). B.1.5.11 ⟩ ≡

$reif_A\ : A_{mod} \sqsubseteq_{retr_A} D_{mod}$

B.1.6 Extension by user-held locks

The kernel system does not take into account a multi-user environment. In such an environment, several users may simultaneously check out and check in revisions. This obviously requires some sort of coordination in order to prevent an inconsistent state of the project. A simple solution is to introduce locks which a user may own for (one or more) revisions and to require that a user must own a lock for a revision in order to check it in. This is described in detail below.

The overall presentation structure is the same as that of the kernel system.

⟨ Extension by user-held locks. B.1.6 ⟩ ≡

[[⟨ State and invariant (locks). B.1.6.1 ⟩

; ⟨ Operations (locks). B.1.6.2 ⟩

; ⟨ Module assembly (locks). B.1.6.9 ⟩

; ⟨ Validity of the specification (locks). B.1.6.10 ⟩

]]

B.1.6.1. The state is extended by a new component that describes which revisions are locked by whom. The constructor and the destructors are adapted accordingly. The invariant is extended to ensure that there are no locked revisions outside the revision control system.

⟨ State and invariant (locks). B.1.6.1 ⟩ ≡

$$
\begin{array}{llllll}
[\![\ Uid & : & sort & & & \\
;\ L_{st} & := & K_{st} \otimes (Rid \xrightarrow{m} Uid) & & & \\
;\ L & := & \langle\!|\ mk & := & (\mapsto) \therefore [K_{st}; (Rid \xrightarrow{m} Uid) \vdash L_{st}\,] \\
& & ,\ K & := & [\,rg : L_{st} \ \vdash sel_1(rg)\,] \\
& & ,\ cont & := & [\,rg : L_{st} \ \vdash K.\ cont\ (sel_1(rg))\,] \\
& & ,\ dep & := & [\,rg : L_{st} \ \vdash K.\ dep\ (sel_1(rg))\,] \\
& & ,\ locks & := & [\,rg : L_{st} \ \vdash sel_2(rg)\,] \\
& & |\!\rangle & &
\end{array}
$$

$; L_{inv} := [\, rg : L_{st}$

$\quad \vdash K_{mod} \,.\, inv\ (L.\, K\ (rg))$

$\quad\quad \wedge\ \mathbf{dom}\ L \,.\, locks\ (rg) \subseteq \mathbf{dom}\ L \,.\, cont\ (rg)$

$\quad]$

$]\!]$

B.1.6.2. The presence of locks gives rise to operations to set and release locks. The other operations are adapted to the new state. This will be done in a structured way using the calculus of operations and the mapping module interface from the method-specific library. Finally, the structured specifications are evaluated, to allow inspection of their expanded versions.

⟨ Operations (locks). B.1.6.2 ⟩ ≡

$[\![$ **import** $MAPPINGS\_int$

$; L_{op} := \langle\!|\ RESET$:= ⟨ Reset the system (locks). B.1.6.3 ⟩

$, OPEN$:= ⟨ Open the system with a root file (locks). B.1.6.4 ⟩

$, SET$:= ⟨ Lock a file for a user. B.1.6.5 ⟩

$, FREE$:= ⟨ Release a lock. B.1.6.6 ⟩

$, CHECKIN$:= ⟨ Check in a file (locks). B.1.6.7 ⟩

$, CHECKOUT$:= ⟨ Check out a file (locks). B.1.6.8 ⟩

$|\!\rangle$

; ⟨ Evaluation of the operations (locks). B.1.6.11 ⟩

$]\!]$

B.1.6.3. The new reset operation the conjunction of the old reset operation and the creation of an empty mapping, since initially there are neitehr revisions nor locks. Note that the order of arguments matters, since otherwise the operation would not be of the required type.

⟨ Reset the system (locks). B.1.6.3 ⟩ ≡

$K_{op} \,.\, RESET \bigwedge MAP_{op}.\, CREATE$

$\quad \therefore\ op_{st}(L_{st})$

B.1.6.4. This operation is extended in an analogous fashion, except that (to ensure type consistency) the input sort of $MAP_{op}\,.CREATE$ must be initialised.

⟨ Open the system with a root file (locks). B.1.6.4 ⟩ ≡

$K_{op} \,.\, OPEN \bigwedge init.\, in\ (MAP_{op}.CREATE)$

$\quad \therefore\ op_{in}(File \otimes Rid, L_{st})$

B.1.6.5. To set a lock corresponds essentially to the *INSERT* operation for mappings. However, two additional things need to be done: the type of the insertion operation must be adapted to conform to the state of the revision control system ($xtnd \,.\, st$ **.LEFT**) and (more importantly) the precondition must be strengthened to require that a revision to be locked must be present in the revision control system.

$\langle$ Lock a file for a user. B.1.6.5 $\rangle \equiv$

$$
\begin{array}{l}
xtnd\,.\,st\,.sel_1(MAP_{op}.\,INSERT\ (max := RevMax)) \\
\wedge_{PRE}\ [\ in\ :\ \ (Rid \otimes Uid);\ rg\ :\ \ L_{st} \\
\qquad ;\ r\ \ := sel_1(in);\qquad u\ \ := sel_2(in) \\
\qquad \vdash r \notin \mathbf{dom}\, L\,.\,cont\ (rg) \\
\qquad] \\
\quad \therefore op_{in}(Rid \otimes Uid, L_{st})
\end{array}
$$

B.1.6.6. To release a lock corresponds essentially to the *DELETE* operation on mappings, except that the state upon which this operation is working must be extended.

$\langle$ Release a lock. B.1.6.6 $\rangle \equiv$

$$
\begin{array}{l}
xtnd\,.\,st\,.sel_1(MAP_{op}.DELETE) \\
\quad \therefore op_{in}(Rid, L_{st})
\end{array}
$$

B.1.6.7. The old check-in operation is adapted to the new state and equipped with a new input parameter (the identifier of the user calling the operation). Furthermore, and this is the crucial step, its precondition its strengthened to check whether the user calling the operation actually owns a locks on the previous version.

$\langle$ Check in a file (locks). B.1.6.7 $\rangle \equiv$

$$
\begin{array}{l}
xtnd\,.\,in\,.sel_1(xtnd.\,st\,.sel_2(K_{op}.CHECKIN)) \\
\wedge_{PRE}\ [\ in\ \ :\ \ Uid \otimes (File \otimes (Rid \otimes Rid));\ rg\ :\ \ L_{st} \\
\qquad ;\ prev\ := sel_1(sel_2(sel_2(in)));\qquad u\ \ := sel_1(in) \\
\qquad \vdash prev \in \mathbf{dom}\, L\,.\,locks\ (rg) \\
\qquad\quad \wedge\ L.\,locks\ (rg) \nabla prev = u \\
\qquad] \\
\quad \therefore op_{in}(Uid \otimes (File \otimes (Rid \otimes Rid)), L_{st})
\end{array}
$$

B.1.6.8. The old check-out operation is adapted to the new state.

$\langle$ Check out a file (locks). B.1.6.8 $\rangle \equiv$

$$
\begin{array}{l}
xtnd\,.\,st\,.sel_2(K_{op}.CHECKOUT) \\
\quad \therefore op(Rid, File, L_{st})
\end{array}
$$

B.1.6.9. The module is assembled as susual.

$\langle$ Module assembly (locks). B.1.6.9 $\rangle \equiv$

$$
\begin{array}{l}
L_{mod}\ := \langle\!|\ inv\ := \ L_{inv} \\
\qquad\qquad ,\ ops\ := \langle L_{op}.RESET \rangle \odot \langle L_{op}.OPEN \rangle \\
\qquad\qquad\qquad\qquad \odot \langle L_{op}.CHECKIN \rangle \odot \langle L_{op}.CHECKOUT \rangle \\
\qquad\qquad\qquad\qquad \odot \langle L_{op}.SET \rangle \odot \langle L_{op}.FREE \rangle \\
\qquad\qquad |\!\rangle
\end{array}
$$

B.1.6.10.

$\langle$ Validity of the specification (locks). B.1.6.10 $\rangle \equiv$

$[\![\langle$ Validity lemmas (locks). B.5.1.1 $\rangle$

; $\langle$ Evaluation proofs B.5.8 $\rangle$

; L_RESET_{val} := $\langle$ Proof of $val\_op\ (L_{op}.RESET, L_{mod}.inv)$ B.5.2 $\rangle$

$\therefore val\_op(L_{op}.RESET, L_{mod}.inv)$

; L_OPEN_{val} := $\langle$ Proof of $val\_op\ (L_{op}.OPEN, L_{mod}.inv)$ B.5.3 $\rangle$

$\therefore val\_op(L_{op}.OPEN, L_{mod}.inv)$

; L_SET_{val} := $\langle$ Proof of $val\_op\ (L_{op}.SET, L_{mod}.inv)$ B.5.4 $\rangle$

$\therefore val\_op(L_{op}.SET, L_{mod}.inv)$

; L_FREE_{val} := $\langle$ Proof of $val\_op\ (L_{op}.FREE, L_{mod}.inv)$ B.5.5 $\rangle$

$\therefore val\_op(L_{op}.FREE, L_{mod}.inv)$

; $L\_CHECKOUT_{val}$:= $\langle$ Proof of $val\_op\ (L_{op}.CHECKOUT, L_{mod}.inv)$ B.5.7 $\rangle$

$\therefore val\_op(L_{op}.CHECKOUT, L_{mod}.inv)$

; $L\_CHECKIN_{val}$:= $\langle$ Proof of $val\_op\ (L_{op}.CHECKIN, L_{mod}.inv)$ B.5.6 $\rangle$

$\therefore val\_op(L_{op}.CHECKIN, L_{mod}.inv)$

; L_valid := $\langle\!|\langle\!|\langle\!|\ L\_RESET_{val}, L\_OPEN_{val}, L\_CHECKIN_{val}, L\_CHECKOUT_{val}\ |\!\rangle$

$\backslash\ val\_assembly_4$

$,\ def\_val\_oplist\ .\ sing\ .\ up\ (L\_SET_{val})$

$|\!\rangle$

$\backslash\ and.\ in\ \backslash\ def\_val\_oplist\ .\ cons\ .\ up$

$,\ def\_val\_oplist\ .\ sing\ .\ up\ (L\_FREE_{val})$

$|\!\rangle$

$\backslash\ and.\ in\ \backslash\ def\_val\_oplist\ .\ cons\ .\ up$

$\therefore mod\_valid(L_{mod})$

$]\!]$

B.1.6.11. Finally, all specifications are presented in completely evaluated form. The main purpose is to check whether the use of the specification operators corresponed adequately to the intended meaning. The evaluated forms are listed without further comment.

$\langle$ Evaluation of the operations (locks). B.1.6.11 $\rangle \equiv$

$[\![\langle$ Evaluation of *RESET*. B.1.6.12 $\rangle$

; $\langle$ Evaluation of *OPEN*. B.1.6.13 $\rangle$

; $\langle$ Evaluation of *SET*. B.1.6.14 $\rangle$

; $\langle$ Evaluation of *FREE*. B.1.6.15 $\rangle$

; $\langle$ Evaluation of *CHECKIN*. B.1.6.16 $\rangle$

; $\langle$ Evaluation of *CHECKOUT*. B.1.6.17 $\rangle$

$]\!]$

B.1.6.12.

$\langle$ Evaluation of *RESET*. B.1.6.12 $\rangle \equiv$

L_RESET_eval :

L_{op} . $RESET$

$$
\begin{array}{l}
=_{op} ([\_ : \textbf{void}\,;\overleftarrow{rg} :\ L_{st} \\
\quad \vdash \langle\!| \ pre \ \ := \ true \wedge true \\
\quad\quad ,\ post \ := [\_ : \textbf{void}\,; rg :\ L_{st} \\
\quad\quad\quad\quad \vdash L\,.\,K\ (rg) = K\,.\,mk\ (\langle\rangle, \tau) \\
\quad\quad\quad\quad\quad \wedge\ L.\ locks\ (rg) = \langle\rangle \\
\quad\quad\quad\quad] \\
\quad\quad |\!\rangle \\
\quad] \\
\quad \therefore op_{st}(L_{st}))
\end{array}
$$

B.1.6.13.

$\langle$ Evaluation of $OPEN$. B.1.6.13$\rangle \equiv$

L_OPEN_eval :

L_{op} . $OPEN$

$$
\begin{array}{l}
=_{op} ([\ in\ : (File \otimes Rid); \overleftarrow{rg} :\ L_{st} \\
\quad \vdash [\ f\ := sel_1(in);\ r\ := sel_2(in) \\
\quad\quad \vdash \langle\!|\ pre \ \ := \textbf{dom}\ L\,.\,cont\ (\overleftarrow{rg}) = \{\,\} \wedge true \\
\quad\quad\quad ,\ post\ := [\_ : \textbf{void}\,; rg :\ L_{st} \\
\quad\quad\quad\quad\quad \vdash L\,.\,K\ (rg) = K\,.\,mk\ (\langle r \mapsto f\rangle, node(r, \langle\rangle)) \\
\quad\quad\quad\quad\quad\quad \wedge\ L.\ locks\ (rg) = \langle\rangle \\
\quad\quad\quad\quad\quad] \\
\quad\quad\quad |\!\rangle \\
\quad\quad] \\
\quad] \\
\quad \therefore op_{in}(File \otimes Rid, L_{st}))
\end{array}
$$

B.1.6.14.

$\langle$ Evaluation of SET. B.1.6.14$\rangle \equiv$

L_SET_eval :

L_{op} . SET

$$
\begin{array}{l}
=_{op} ([\ in\ : (Rid \otimes Uid); \overleftarrow{rg} :\ L_{st} \\
\quad \vdash [\ r\ := sel_1(in);\ u\ := sel_2(in) \\
\quad\quad \vdash \langle\!|\ pre \ \ := r \notin \textbf{dom}\ L\,.\,locks\ (\overleftarrow{rg}) \\
\quad\quad\quad\quad\quad \wedge\ \textbf{card}(\textbf{dom}\ L\,.\,locks\ (\overleftarrow{rg}))\ ¡\ RevMax \\
\quad\quad\quad\quad\quad \wedge\ r \in \textbf{dom}\ L\,.\,cont\ (\overleftarrow{rg}) \\
\quad\quad\quad ,\ post\ := [\_ : \textbf{void}\,; rg :\ L_{st}\ \vdash L.\ K\ (rg) = L.\ K\ (\overleftarrow{rg})\,] \\
\quad\quad\quad\quad\quad\quad \wedge\ L.\ locks\ (rg) = (r \mapsto u) \odot L.\ locks\ (\overleftarrow{rg}) \\
\quad\quad\quad |\!\rangle \\
\quad\quad] \\
\quad] \\
\quad \therefore op_{in}(Rid \otimes Uid, L_{st}))
\end{array}
$$

B.1.6.15.

$\langle$ Evaluation of *FREE*. B.1.6.15 $\rangle \equiv$

$$
\begin{array}{l}
L\_FREE\_eval \hspace{20em} : \\
L_{op} \,.\, FREE \\
=_{op} ([\, r \,:\, Rid \,;\, \overleftarrow{rg} :\, L_{st} \\
\quad \vdash \langle\!| \; pre \;\; := r \in \mathbf{dom}\, L \,.\, locks\, (\overleftarrow{rg}) \\
\qquad , post \; := [\_ : \mathbf{void} \,; rg :\, L_{st} \\
\qquad\qquad \vdash L \,.\, K\, (rg) = L.\, K\, (\overleftarrow{rg}) \\
\qquad\qquad\quad \wedge\, L.\, locks\, (rg) = [\, r_1 : Rid \,;\, u :\, Uid \,\vdash \neg r_1 = r \,] \rhd L.\, locks\, (\overleftarrow{rg}) \\
\qquad\qquad] \\
\qquad |\!\rangle \\
\quad] \\
\;\; \therefore\, op_{in}(Rid, L_{st}))
\end{array}
$$

B.1.6.16.

$\langle$ Evaluation of *CHECKIN*. B.1.6.16 $\rangle \equiv$

$$
\begin{array}{l}
L\_CHECKIN\_eval \hspace{15em} : \\
L_{op} \,.\, CHECKIN \\
=_{op} ([\, in \,:\, Uid \otimes (File \otimes (Rid \otimes Rid)); \overleftarrow{rg} :\, L_{st} \\
\quad \vdash [\; uid \;\; := sel_1(in); \qquad\qquad f \;\; := sel_1(sel_2(in)) \\
\qquad ; \; prev \; := sel_1(sel_2(sel_2(in))); \; new \; := sel_2(sel_2(sel_2(in))) \\
\qquad \vdash \langle\!| \; pre \;\; := prev \in \mathbf{dom}\, L \,.\, cont\, (\overleftarrow{rg}) \\
\qquad\qquad\quad \wedge\, \neg(new \in \mathbf{dom}\, L \,.\, cont\, (\overleftarrow{rg})) \\
\qquad\qquad\quad \wedge\, RevMax \,¿\, \mathbf{card}(\mathbf{dom}\, L \,.\, cont\, (\overleftarrow{rg})) \\
\qquad\qquad\quad \wedge\, prev \in \mathbf{dom}\, L \,.\, locks\, (\overleftarrow{rg}) \\
\qquad\qquad\quad \wedge\, L.\, locks\, (\overleftarrow{rg}) \nabla prev = uid \\
\qquad\quad , post \; := [\_ : \mathbf{void} \,; rg :\, L_{st} \\
\qquad\qquad\quad \vdash L \,.\, K\, (rg) = K \,.\, mk\, ((new \mapsto f) \odot L.\, cont\, (\overleftarrow{rg}), \\
\qquad\qquad\qquad insert\, (prev, new, L.\, dep\, (\overleftarrow{rg}))) \\
\qquad\qquad\qquad \wedge\, L.\, locks\, (rg) = L.\, locks\, (\overleftarrow{rg}) \\
\qquad\qquad\quad] \\
\qquad\quad |\!\rangle \\
\qquad] \\
\quad] \\
\;\; \therefore\, op_{in}(Uid \otimes (File \otimes (Rid \otimes Rid)), L_{st}))
\end{array}
$$

B.1.6.17.

$\langle$ Evaluation of *CHECKOUT*. B.1.6.17 $\rangle \equiv$

$$
L\_CHECKOUT\_eval \hspace{12em} :
$$

$$
\begin{array}{l}
L_{op} \mathbin{.} CHECKOUT \\
=_{op} ([\, r \,:\, Rid \,;\, \overleftarrow{rg} \,:\, L_{st} \\
\quad \vdash \langle\!| \; pre \;:= r \in \mathbf{dom}\, L \mathbin{.} cont\,(\overleftarrow{rg}) \\
\quad\quad , post \;:= [\, f \,:\, File \,;\, rg : \, L_{st} \\
\quad\quad\quad\quad \vdash (f = L.\, cont\,(\overleftarrow{rg}) \nabla\, r \wedge L \mathbin{.} K\,(rg) = L.\, K\,(\overleftarrow{rg})) \\
\quad\quad\quad\quad\quad \wedge\, L.\, locks\,(rg) = L.\, locks\,(\overleftarrow{rg}) \\
\quad\quad\quad\quad] \\
\quad\quad |\!\rangle \\
\quad] \\
\quad \therefore op(Rid, File, L_{st}))
\end{array}
$$

B.1.7 Extension to robust operations

The final extension of the specification is a straightforward step in which each operation is augmented so as to keep the revision group unchanged in case its precondition is not satisfied. This is achieved using precondition complementation and operation disjunction.

$$
\begin{array}{l}
\langle\, \text{Extension to robust operations. B.1.7} \,\rangle \equiv \\
[\![\; T_{st} \;\; := L_{st} \\
;\; T_{inv} \; := L_{inv} \\
;\; T_{op} \;\; := \langle\!| \; RESET \quad\quad := L_{op} \mathbin{.} RESET \\
\quad\quad\quad\quad , OPEN \quad\quad := L_{op} \mathbin{.} OPEN \bigvee L_{op}.\, OPEN^{\scriptscriptstyle CPRE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \therefore op_{in}(File \otimes Rid,\, T_{st}) \\
\quad\quad\quad\quad , SET \quad\quad\quad := L_{op} \mathbin{.} SET \bigvee L_{op}.\, SET^{\scriptscriptstyle CPRE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \therefore op_{in}(Rid \otimes Uid,\, T_{st}) \\
\quad\quad\quad\quad , FREE \quad\quad := L_{op} \mathbin{.} FREE \bigvee L_{op}.\, FREE^{\scriptscriptstyle CPRE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \therefore op_{in}(Rid,\, T_{st}) \\
\quad\quad\quad\quad , CHECKIN \quad := L_{op} \mathbin{.} CHECKIN \bigvee L_{op}.\, CHECKIN^{\scriptscriptstyle CPRE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \therefore op_{in}(Uid \otimes (File \otimes (Rid \otimes Rid)),\, T_{st}) \\
\quad\quad\quad\quad , CHECKOUT := L_{op} \mathbin{.} CHECKOUT \bigvee L_{op}.\, CHECKOUT^{\scriptscriptstyle CPRE} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \therefore op(Rid, File,\, T_{st}) \\
\quad\quad\quad\quad |\!\rangle \\
;\; T_{mod} \; := \langle\, \text{Module assembly (total operations). B.1.7.1} \,\rangle \\
;\langle\, \text{Validity of the specification (total operations). B.1.7.2} \,\rangle \\
]\!]
\end{array}
$$

B.1.7.1.

$$
\begin{array}{l}
\langle\, \text{Module assembly (total operations). B.1.7.1} \,\rangle \equiv \\
\langle\!| \; inv \;:= L_{mod} \mathbin{.} inv \\
,\; ops \;:= \langle T_{op}.RESET \rangle \odot \langle T_{op}.OPEN \rangle \\
\quad\quad\quad \odot \langle T_{op}.CHECKIN \rangle \odot \langle T_{op}.CHECKOUT \rangle \odot \langle T_{op}.SET \rangle \\
\quad\quad\quad \odot \langle T_{op}.FREE \rangle \\
|\!\rangle
\end{array}
$$

B.1.7.2.

$\langle$ Validity of the specification (total operations). B.1.7.2 $\rangle \equiv$

$$
\begin{array}{lll}
[\![& T\_RESET_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.RESET, L_{mod}.inv).\ \text{B.6} \rangle \\
 & & \quad \therefore val\_op(T_{op}.RESET, L_{mod}.inv) \\
; & T\_OPEN_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.OPEN, L_{mod}.inv).\ \text{B.6.1} \rangle \\
 & & \quad \therefore val\_op(T_{op}.OPEN, L_{mod}.inv) \\
; & T\_SET_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.SET, L_{mod}.inv).\ \text{B.6.2} \rangle \\
 & & \quad \therefore val\_op(T_{op}.SET, L_{mod}.inv) \\
; & T\_FREE_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.FREE, L_{mod}.inv).\ \text{B.6.3} \rangle \\
 & & \quad \therefore val\_op(T_{op}.FREE, L_{mod}.inv) \\
; & T\_CHECKOUT_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.CHECKOUT, L_{mod}.inv).\ \text{B.6.4} \rangle \\
 & & \quad \therefore val\_op(T_{op}.CHECKOUT, L_{mod}.inv) \\
; & T\_CHECKIN_{val} & := \langle \text{Proof of } val\_op\,(T_{op}.CHECKIN, L_{mod}.inv).\ \text{B.6.5} \rangle \\
 & & \quad \therefore val\_op(T_{op}.CHECKIN, L_{mod}.inv) \\
; & T\_valid & := \langle\!|\langle\!|\langle\!| \; T\_RESET_{val}\,,\, T\_OPEN_{val}\,,\, T\_CHECKIN_{val}\,,\, T\_CHECKOUT_{val} \;|\!\rangle \\
 & & \qquad \backslash\; val\_assembly_4 \\
 & & \quad ,\; def\_val\_oplist\,.\,sing\,.\,up\;(T\_SET_{val}) \\
 & & \quad |\!\rangle \\
 & & \quad \backslash\; and.\,in \;\backslash\; def\_val\_oplist\,.\,cons\,.\,up \\
 & & ,\; def\_val\_oplist\,.\,sing\,.\,up\;(T\_FREE_{val}) \\
 & & |\!\rangle \\
 & & \backslash\; and.\,in \;\backslash\; def\_val\_oplist\,.\,cons\,.\,up \\
 & & \quad \therefore val\_oplist(T_{mod}.ops, L_{mod}.inv) \\
]\!] & &
\end{array}
$$

B.1.8 Further extensions

Because of limitation of time and resources, this development case study has concentrated on some essential aspects of the development of a revision control system.

A full practical revision control system has many features not captured by the above specification. The following list presents some typical examples:

- Systematically generated revision names,
- a designated user, the project manager, with special access and modification rights,
- attribution of the revisions with information about the history of their origin,
- the integration with a file management system and its functionality.

B.2 Specification of the kernel system: proofs

B.2.1 Reset operation

According to the definition of validity of operations (*val_op*), one has to prove satisfiability and invariant preservation. The global structure of the proof is akin to that of the definition of *val_op*. Note that the precondition of *RESET* is *true*. Note also, that the proof does not depend on the assumptions about the invariant and the precondition, i.e. these assumptions remain unnamed.

$\langle$ Proof of $val\_op\ (K_{op}.RESET, K_{mod}.inv)$. B.2.1 $\rangle \equiv$

$[\ in\ :\ \mathbf{void}\,;\overleftarrow{rg}\ :\ K_{st}$

$K_{mod}\ .\ inv\ (\overleftarrow{rg});$

$true$

$\vdash$ $new\ :=\ K\ .\ mk\ (\langle\rangle, \tau)$

$\lVert\ satisfiability\ :=\ \langle$ Proof of satisfiability (RESET). B.2.1.1 $\rangle$

$,\ preservation\ :=\ \langle$ Proof of invariant preservation (RESET). B.2.1.2 $\rangle$

$\rVert$

$]$

$\therefore val\_op(K_{op}.RESET, K_{mod}.inv)$

B.2.1.1. The proof of satisfiability is trivial, the existential proposition is proven by simply taking new as rg (and $\_$ as the dummy-output value).

$\langle$ Proof of satisfiability (RESET). B.2.1.1 $\rangle \equiv$

$refl - \therefore new = new$

$\therefore K_{op}.\ RESET\ (in, \overleftarrow{rg}).\ post\ (\_, new)$

$\setminus\ ex2\_eq\_intro$

$\therefore \exists_2[\ out : \mathbf{void}\,; rg :\ K_{st}\ \vdash K_{op}.\ RESET\ (in, \overleftarrow{rg}).\ post\ (out, rg)\,]$

B.2.1.2. The proof of invariant preservation is slightly more involved. When assuming the post-condition of $RESET$, one has to prove the invariant for the post-state rg. This proof is split into 3 subproofs, which will be detailed below.

$\langle$ Proof of invariant preservation (RESET). B.2.1.2 $\rangle \equiv$

$[\ out\ :\ \mathbf{void}\,; rg :\ K_{st}$

$\vdash [\ hyp\_post\ : rg = new$

$\vdash \lVert \langle$ Proof of the first part of the invariant (RESET). B.2.1.3 $\rangle$

$\therefore \mathbf{dom}\ K\ .\ cont\ (new) = info(K.\ dep\ (new))$

$, \langle$ Proof of the second part of the invariant (RESET). B.2.1.4 $\rangle$

$\therefore nodup(K.\ dep\ (new))$

$, \langle$ Proof of the third part of the invariant (RESET). B.2.1.5 $\rangle$

$\therefore RevMax \geq \mathbf{card}(\mathbf{dom}\ K\ .\ cont\ (new))$

$, \langle$ Proof of the fourth part of the invariant (RESET). B.2.1.6 $\rangle$

$\therefore \forall[\ r : Rid\ \vdash r \in \mathbf{dom}\ K\ .\ cont\ (new) \Rightarrow wff_F(K.\ cont\ (new) \nabla r)\,]$

$\rVert$

$\setminus\ and4$

$\therefore K_{mod}.\ inv\ (new)$

$\setminus\ rsubst(hyp\_post)$

$\therefore K_{mod}.\ inv\ (rg)$

$]$

$]$

B.2.1.3. The proof of the first conjunct amounts to a simple equaltional reasoning to prove $\mathbf{dom}\, K\,.\, cont\,(new) = info(K.\, dep\,(new))$. The below proof proceeds by simplifying both expressions to $\{\,\}$.

$\langle$ Proof of the first part of the invariant (RESET). B.2.1.3 $\rangle \equiv$

$[\, lhs \; := \mathbf{dom}\, K\,.\, cont\,(new);\; rhs \; := \; info\,(K.\, dep\,(new))$

$\vdash \langle\!| \; refl$

$\quad\quad \therefore lhs = \mathbf{dom}\, K\,.\, cont\,(new)$

$\quad \backslash \; unfold(def\_sel_1)$

$\quad\quad \therefore lhs = \mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} File)$

$\quad \backslash \; unfold(domain.empty)$

$\quad\quad \therefore lhs = \{\,\}$

$\;,\; refl$

$\quad\quad \therefore rhs = info(K.\, dep\,(new))$

$\quad \backslash \; unfold(def_s el_2)$

$\quad\quad \therefore rhs = info(\tau)$

$\quad \backslash \; unfold(def\_info.empty)$

$\quad\quad \therefore rhs = \{\,\}$

$\; |\!\rangle$

$\; \backslash \; indirect\_product$

$\quad \therefore lhs = rhs$

$]$

B.2.1.4. The second part of the invariant $nodup\,(K.\, dep\,(new))$ is proven by transforming it into *true*.

$\langle$ Proof of the second part of the invariant (RESET). B.2.1.4 $\rangle \equiv$

$[\, goal \; := \; nodup\,(K.\, dep\,(new))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow nodup(K.\, dep\,(new))$

$\; \backslash \; unfold(def_s el_2)$

$\quad \therefore goal \Leftrightarrow nodup(\tau \therefore tree(Rid))$

$\; \backslash \; unfold(def\_nodup.empty)$

$\quad \therefore goal \Leftrightarrow true$

$\; \backslash \; valid.\, down$

$\quad \therefore goal$

$]$

B.2.1.5. The third part of the invariant is proven by the same goal reduction technique.

$\langle$ Proof of the third part of the invariant (RESET). B.2.1.5 $\rangle \equiv$

$[\,goal\ := RevMax \geq \mathbf{card}(\mathbf{dom}\ K\,.\,cont\ (new))$

$\vdash refl$

$\quad\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\ K\,.\,cont\ (new))$

$\ \backslash\ unfold(def\_sel_1)$

$\quad\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} File))$

$\ \backslash\ unfold(domain.empty)$

$\quad\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\{\,\} \therefore set(Rid))$

$\ \backslash\ unfold(def\_card.empty)$

$\quad\therefore goal \Leftrightarrow RevMax \geq 0$

$\ \backslash\ unfold(valid.\ up\ (geq\_prop(RevMax_{pos})))$

$\quad\therefore goal \Leftrightarrow true$

$\ \backslash\ valid.\ down$

$\quad\therefore goal$

$]$

B.2.1.6.

$\langle$ Proof of the fourth part of the invariant (RESET). B.2.1.6 $\rangle \equiv$

$[\,goal\ := [\,r\ :\ Rid$

$\qquad\vdash r \in \mathbf{dom}\ K\,.\,cont\ (new)$

$\qquad\quad\Rightarrow wff_F(K.\ cont\ (new)\nabla r)$

$\qquad]$

$\vdash[\,r\ :\ Rid$

$\ \vdash[\,hyp\ : r \in \mathbf{dom}\ K\,.\,cont\ (new)$

$\quad\vdash hyp$

$\qquad\therefore r \in \mathbf{dom}\ K\,.\,cont\ (new)$

$\quad\backslash\ subst(def\_sel_1)$

$\qquad\therefore r \in \mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} File)$

$\quad\backslash\ subst(domain.empty)$

$\qquad\therefore r \in \{\,\}$

$\quad\backslash\ not.\ out\ (member.empty)$

$\qquad\therefore false$

$\quad\backslash\ false\_out$

$\qquad\therefore wff_F(K.\ cont\ (new)\nabla r)$

$\quad]$

$\quad\backslash\ imp.\ in$

$\ ]$

$\ \backslash\ univ.\ in$

$\quad\therefore \forall[\,r : Rid\ \vdash goal(r)\,]$

$]$

B.2.2 Open operation

$\langle$ Proof of $val\_op\ (K_{op}.OPEN, K_{mod}.inv)$. B.2.2 $\rangle \equiv$

$[\ in\ :\ File \otimes Rid\ ; \overleftarrow{rg}\ :\ K_{st}$

$\vdash [\ f\ := sel_1(in);\ r\ := sel_2(in);\ new\ :=\ K\ .\ mk\ (\langle r \mapsto f \rangle, node(r, \langle\rangle))$

$K_{mod}\ .\ inv\ (\overleftarrow{rg});$

$hyp\_pre \quad : K_{op}\ .\ OPEN\ (in, \overleftarrow{rg}).\ pre$

$\vdash \langle\!|\ satisfiability\ := \langle$ Proof of satisfiability (OPEN). B.2.2.1 $\rangle$

$,\ preservation\ := \langle$ Proof of invariant preservation (OPEN). B.2.2.2 $\rangle$

$|\!\rangle$

$]$

$]$

$\therefore val\_op(K_{op}.OPEN, K_{mod}.inv)$

B.2.2.1. The satisfiability clause is proven as above.

$\langle$ Proof of satisfiability (OPEN). B.2.2.1 $\rangle \equiv$

$refl - \therefore new = new$

$\therefore K_{op}.\ OPEN\ (in, \overleftarrow{rg}).\ post\ (\_\_, new)$

$\setminus\ ex2\_eq\_intro$

$\therefore \exists_2[\ out : \mathbf{void}; rg :\ K_{st}\ \vdash K_{op}.\ OPEN\ (in, \overleftarrow{rg}).\ post\ (out, rg)\ ]$

B.2.2.2. The proof is not difficult. It's structure is analogous to the corresponding proof for the *RESET* operation.

$\langle$ Proof of invariant preservation (OPEN). B.2.2.2 $\rangle \equiv$

$[\ out\ :\ \mathbf{void}; rg :\ K_{st}$

$\vdash [\ hyp\_post\ : rg = new$

$\vdash \langle\!|\langle$ Proof of the first part of the invariant (OPEN). B.2.2.3 $\rangle$

$\therefore \mathbf{dom}\ K\ .\ cont\ (new) = info(K.\ dep\ (new))$

$,\langle$ Proof of the second part of the invariant (OPEN). B.2.2.4 $\rangle$

$\therefore nodup(K.\ dep\ (new))$

$,\langle$ Proof of the third part of the invariant (OPEN). B.2.2.5 $\rangle$

$\therefore RevMax \geq \mathbf{card}(\mathbf{dom}\ K\ .\ cont\ (new))$

$,\langle$ Proof of the fourth part of the invariant (OPEN). B.2.2.6 $\rangle$

$\therefore \forall[\ r : Rid\ \vdash r \in \mathbf{dom}\ K\ .\ cont\ (new) \Rightarrow wff_F(K.\ cont\ (new) \nabla r)\ ]$

$|\!\rangle$

$\setminus\ and4$

$\therefore K_{mod}.\ inv\ (new)$

$\setminus\ rsubst(hyp\_post)$

$\therefore K_{mod}.\ inv\ (rg)$

$]$

$]$

B.2.2.3.

$\langle$ Proof of the first part of the invariant (OPEN). B.2.2.3 $\rangle \equiv$

$[\, lhs \ := \mathbf{dom}\, K \,.\, cont\ (new);\ rhs\ := \ info\ (K.\ dep\ (new))$

$\vdash \langle\!| \ refl$

$\quad \therefore lhs = \mathbf{dom}\, K \,.\, cont\ (new)$

$\ \setminus\ unfold(def\_sel_1)$

$\quad \therefore lhs = \mathbf{dom}\langle r \mapsto f\rangle$

$\ \setminus\ unfold(dom\_prop.single)$

$\quad \therefore lhs = \{r\}$

$,\ refl$

$\quad \therefore rhs = info(K.\ dep\ (new))$

$\ \setminus\ unfold(def_sel_2)$

$\quad \therefore rhs = info(node(r, \langle\rangle))$

$\ \setminus\ unfold(info\_prop.single)$

$\quad \therefore rhs = \{r\}$

$|\!\rangle$

$\setminus\ indirect\_product$

$\quad \therefore lhs = rhs$

$]$

B.2.2.4.

$\langle$ Proof of the second part of the invariant (OPEN). B.2.2.4 $\rangle \equiv$

$[\, goal\ := \ nodup\ (K.\ dep\ (new))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow nodup(K.\ dep\ (new))$

$\setminus\ unfold(def_sel_2)$

$\quad \therefore goal \Leftrightarrow nodup(node(r, \langle\rangle))$

$\setminus\ unfold(nodup\_prop.single)$

$\quad \therefore goal \Leftrightarrow true$

$\setminus\ valid.\ down$

$\quad \therefore goal$

$]$

B.2.2.5.

$\langle$ Proof of the third part of the invariant (OPEN). B.2.2.5 $\rangle \equiv$

$[\, goal \; := RevMax \geq \mathbf{card}(\mathbf{dom}\; K\, \mathbf{.}\; cont\; (new))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\; K\, \mathbf{.}\; cont\; (new))$

$\; \setminus unfold(def\_sel_1)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\langle r \mapsto f\rangle)$

$\; \setminus unfold(dom\_prop\mathbf{.}single)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\{r\})$

$\; \setminus unfold(sing\mathbf{.card})$

$\quad \therefore goal \Leftrightarrow RevMax \geq 1$

$\; \setminus unfold(valid\mathbf{.}\; up\; (RevMax_{pos}))$

$\quad \therefore goal \Leftrightarrow true$

$\; \setminus valid\mathbf{.}\; down$

$\quad \therefore goal$

$]$

B.2.2.6.

$\langle$ Proof of the fourth part of the invariant (OPEN). B.2.2.6 $\rangle \equiv$

$[\, goal \; := [\, r1 \; : \; Rid$

$\qquad \vdash r1 \in \mathbf{dom}\; K\, \mathbf{.}\; cont\; (new)$

$\qquad\quad \Rightarrow wff_F(K\mathbf{.}\; cont\; (new) \nabla r1)$

$\qquad]$

$\vdash [\, r1 \; : \; Rid$

$\;\vdash [\, hyp \quad : \quad r1 \in \mathbf{dom}\; K\, \mathbf{.}\; cont\; (new)$

$\quad ;\; side \quad := \langle$ Side deduction (OPEN). B.2.2.7 $\rangle \therefore r1 = r$

$\quad ;\; subgoal \; := wff_F(K\mathbf{.}\; cont\; (new) \nabla r1)$

$\quad \vdash refl$

$\qquad \therefore subgoal \Leftrightarrow wff_F(K\mathbf{.}\; cont\; (new) \nabla r1)$

$\quad\; \setminus unfold(def\_sel_1)$

$\qquad \therefore subgoal \Leftrightarrow wff_F(\langle r \mapsto f\rangle \nabla r1)$

$\quad\; \setminus unfold(side)$

$\qquad \therefore subgoal \Leftrightarrow wff_F(\langle r \mapsto f\rangle \nabla r)$

$\quad\; \setminus unfold(app\mathbf{.}first)$

$\qquad \therefore subgoal \Leftrightarrow wff_F(f)$

$\quad\; \setminus unfold(valid\mathbf{.}\; up\; (pRight(hyp\_pre)))$

$\qquad \therefore subgoal \Leftrightarrow true$

$\quad\; \setminus valid\mathbf{.}\; down$

$\qquad \therefore subgoal$

$\quad]$

$\quad\; \setminus imp\mathbf{.}\; in$

$\;]$

$\; \setminus univ\mathbf{.}\; in\; (P := goal)$

$]$

B.2.2.7.

$\langle$ Side deduction (OPEN). B.2.2.7 $\rangle \equiv$

hyp

$\quad \therefore r1 \in \mathbf{dom}\, K \,.\, cont\ (new)$

$\setminus\ subst(def\_sel_1)$

$\quad \therefore r1 \in \mathbf{dom}\langle r \mapsto f \rangle$

$\setminus\ subst(dom\_prop.single)$

$\quad \therefore r1 \in \{r\}$

$\setminus\ subst(member\_prop.single)$

$\quad \therefore r1 = r$

B.2.3 Checkout operation

The validity proof for the check-out operation has the usual overall structure. Note that the proof will makes use of the assumption about the invariant being satisfied on the initial state (hyp_inv).

$\langle$ Proof of $val\_op\ (K_{op}.CHECKOUT, K_{mod}.inv)$. B.2.3 $\rangle \equiv$

$[\ r : Rid\ ; \overleftarrow{rg} : K_{st}\ ;\ res := K\,.\,cont\ (\overleftarrow{rg}) \nabla\ r$

$\quad hyp\_inv \qquad : K_{mod}\,.\,inv\ (\overleftarrow{rg});$

$\quad K_{op}\,.\,CHECKOUT\ (r, \overleftarrow{rg}).\ pre$

$\vdash \langle\!|\ satisfiability\ := \langle$ Proof of satisfiability (CHECKOUT). B.2.3.1 $\rangle$

$\quad ,\ preservation\ := \langle$ Proof of invariant preservation (CHECKOUT). B.2.3.2 $\rangle$

$\quad |\!\rangle$

$]$

$\quad \therefore val\_op(K_{op}.CHECKOUT, K_{mod}.inv)$

B.2.3.1. The satisfiability proof used above is adapted as follows: note that the post-condition of $CHECKOUT$ is a conjunction of equations.

$\langle$ Proof of satisfiability (CHECKOUT). B.2.3.1 $\rangle \equiv$

$\langle\!|\ refl \qquad ,\ refl \qquad |\!\rangle$

$\quad \therefore res = res \quad \therefore \overleftarrow{rg} = \overleftarrow{rg}$

$\setminus\ and.\,in$

$\quad \therefore K_{op}.\ CHECKOUT\ (r, \overleftarrow{rg}).\ post\ (res, \overleftarrow{rg})$

$\setminus\ ex2\_eq2\_intro$

$\quad \therefore \exists_2[f : File\ ; rg :\ K_{st}\ \vdash K_{op}.\ CHECKOUT\ (r, \overleftarrow{rg}).\ post\ (f, rg)]$

B.2.3.2. The proof of invariant preservation is almost trivial, since the state remains unchanged.

$\langle$ Proof of invariant preservation (CHECKOUT). B.2.3.2 $\rangle \equiv$

$[f\ :\ File\,;rg:\ K_{st}$

$\vdash[\ hyp\_post\ : f = res \wedge rg = \overleftarrow{rg}$

$\ \ \vdash hyp\_inv$

$\qquad \therefore K_{mod}.\ inv\ (\overleftarrow{rg})$

$\quad \setminus\ rsubst(pRight(hyp\_post))$

$\qquad \therefore K_{mod}.\ inv\ (rg)$

$\ \ ]$

$]$

B.2.4 Checkin operation

The most complex validity proof is needed for the *CHECKOUT* operation.

⟨ Proof of $val\_op\ (K_{op}.CHECKIN, K_{mod}.inv)$. B.2.4⟩ ≡

$[\ in\ :\ File \otimes (Rid \otimes Rid)$

$;\overleftarrow{rg}\ :\ K_{st}$

$\vdash[\ hyp\_inv\ :\ K_{inv}\ (\overleftarrow{rg})$

$\ \ ;\ hyp\_pre\ :\ K_{op}\ .\ CHECKIN\ (in, \overleftarrow{rg}).\ pre$

; ⟨ Auxiliary definitions. B.2.4.1⟩

$\vdash\langle\!|$ *satisfiability* := ⟨ Proof of satisfiability (CHECKIN). B.2.4.2⟩

$\qquad\qquad \therefore \exists_2[\ out: \mathbf{void}\,; rg:\ K_{st}\ \vdash K_{op}.\ CHECKIN\ (in, \overleftarrow{rg}).\ post\ (out, rg)]$

, *preservation* := ⟨ Proof of invariant preservation (CHECKIN). B.2.4.3⟩

$$\therefore [\ out: \mathbf{void}\,; rg:\ K_{st}\ \vdash \frac{K_{op}\ .\ CHECKIN\ (in, \overleftarrow{rg}).\ post\ (out, rg)}{K_{inv}\ (rg)}]$$

$\quad |\!\rangle$

$\ \ ]$

$]$

$\quad \therefore val\_op(K_{op}.CHECKIN, K_{mod}.inv)$

B.2.4.1. For convenience, some assumptions are decomposed into their individual conjuncts. Furthermore, two simple auxiliary deductions are recorded.

⟨ Auxiliary definitions. B.2.4.1⟩ ≡

$[\![\ f \quad := sel_1(in)$

$;\ or \quad := sel_1(sel_2(in))$

$;\ new \quad := sel_2(sel_2(in))$

$;\ update \quad := K\ .\ mk\ ((new \mapsto f) \odot K.\ cont\ (\overleftarrow{rg}), insert(or, new, K.\ dep\ (\overleftarrow{rg})))$

$;\ hyp\_inv\_dom \quad := pLeft\ (pLeft(pLeft(hyp\_inv)))$

$;\ hyp\_inv\_unique \quad := pRight\ (pLeft(pLeft(hyp\_inv)))$

$;\ hyp\_inv\_wff \quad := pRight\ (hyp\_inv)$

$;\ hyp\_pre\_old \quad := pLeft\ (pLeft(pLeft(hyp\_pre)))$

$;\ hyp\_pre\_new \quad := pRight\ (pLeft(pLeft(hyp\_pre)))$

$;\ hyp\_pre\_safe \quad := pRight\ (pLeft(hyp\_pre))$

$;\ hyp\_pre\_wff \quad := pRight\ (hyp\_pre)$

$; \; aux1 \quad := hyp\_pre\_old$

$\quad \therefore or \in \mathbf{dom}\, K\,.\, cont\ (\overleftarrow{rg})$

$\quad \backslash\ subst(hyp\_inv\_dom)$

$\quad \therefore or \in info(K.\, dep\ (\overleftarrow{rg}))$

$; \; aux2 \quad := hyp\_pre\_new$

$\quad \therefore new \notin \mathbf{dom}\, K\,.\, cont\ (\overleftarrow{rg})$

$\quad \backslash\ subst(hyp\_inv\_dom)$

$\quad \therefore new \notin info(K.\, dep\ (\overleftarrow{rg}))$

$]\!]$

B.2.4.2. The satisfiability clause is proven as usual.

$\langle$ Proof of satisfiability (CHECKIN). B.2.4.2 $\rangle \equiv$

$refl$

$\quad \therefore K_{op}.\, CHECKIN\ (in, \overleftarrow{rg}).\, post\ (\_, update)$

$\backslash\ ex2\_eq\_intro$

$\quad \therefore \exists_2[\, out : \mathbf{void}\, ; rg :\ K_{st}\ \vdash K_{op}.\, CHECKIN\ (in, \overleftarrow{rg}).\, post\ (out, rg)\,]$

B.2.4.3. The invariant preservation proof has the usual overall structure.

$\langle$ Proof of invariant preservation (CHECKIN). B.2.4.3 $\rangle \equiv$

$[\, out\ :\ \mathbf{void}\, ; rg :\ K_{st}$

$\vdash [\, hyp\_post\ : rg = update$

$\;\vdash \langle\!| \langle$ Proof of the first part of the invariant (CHECKIN). B.2.4.4 $\rangle$

$\qquad \therefore \mathbf{dom}\, K\,.\, cont\ (update) = info(K.\, dep\ (update))$

$\quad , \langle$ Proof of the second part of the invariant (CHECKIN). B.2.4.5 $\rangle$

$\qquad \therefore nodup(K.\, dep\ (update))$

$\quad , \langle$ Proof of the third part of the invariant (CHECKIN). B.2.4.6 $\rangle$

$\qquad \therefore RevMax \geq \mathbf{card}(\mathbf{dom}\, K\,.\, cont\ (update))$

$\quad , \langle$ Proof of the fourth part of the invariant (CHECKIN). B.2.4.7 $\rangle$

$\qquad \therefore \forall[\, r : Rid\ \vdash r \in \mathbf{dom}\, K\,.\, cont\ (update) \Rightarrow wff_F(K.\, cont\ (update) \nabla r)\,]$

$\quad |\!\rangle$

$\quad \backslash\ and4$

$\qquad \therefore K_{mod}.\, inv\ (update)$

$\quad \backslash\ rsubst(hyp\_post)$

$\qquad \therefore K_{mod}.\, inv\ (rg)$

$\;]$

$]$

B.2.4.4.

$\langle$ Proof of the first part of the invariant (CHECKIN). B.2.4.4 $\rangle \equiv$

$[\ lhs\ := \mathbf{dom}\ K\ \mathbf{.}\ cont\ (update);\ rhs\ :=\ info\ (K\mathbf{.}\ dep\ (update))$

$\vdash \langle\!|\ refl$

$\quad\quad \therefore\ lhs = lhs$

$\quad \setminus\ unfold(def\_sel_1)$

$\quad\quad \therefore\ lhs = \mathbf{dom}(new \mapsto f) \odot K\mathbf{.}\ cont\ (\overleftarrow{rg})$

$\quad \setminus\ unfold(domain\mathbf{.}recur)$

$\quad\quad \therefore\ lhs = new \odot \mathbf{dom}\ K\ \mathbf{.}\ cont\ (\overleftarrow{rg})$

$\ ,\ refl$

$\quad\quad \therefore\ rhs = rhs$

$\quad \setminus\ unfold(def_s el_2)$

$\quad\quad \therefore\ rhs = info(insert(or, new, K\mathbf{.}\ dep\ (\overleftarrow{rg})))$

$\quad \setminus\ unfold(info\_prop\mathbf{.}\ insert\ (aux1))$

$\quad\quad \therefore\ rhs = new \odot info(K\mathbf{.}\ dep\ (\overleftarrow{rg}))$

$\quad \setminus\ fold(hyp\_inv\_dom)$

$\quad\quad \therefore\ rhs = new \odot \mathbf{dom}\ K\ \mathbf{.}\ cont\ (\overleftarrow{rg})$

$\ |\!\rangle$

$\ \setminus\ indirect\_product$

$\quad \therefore\ lhs = rhs$

$]$

B.2.4.5.

$\langle$ Proof of the second part of the invariant (CHECKIN). B.2.4.5 $\rangle \equiv$

$[\ goal\ :=\ nodup\ (K\mathbf{.}\ dep\ (update))$

$\vdash\ refl$

$\quad \therefore\ goal \Leftrightarrow goal$

$\ \setminus\ unfold(def_s el_2)$

$\quad \therefore\ goal \Leftrightarrow nodup(insert(or, new, K\mathbf{.}\ dep\ (\overleftarrow{rg})))$

$\ \setminus\ unfold(nodup\_prop\mathbf{.}\ insert\ (hyp\_inv\_unique, aux1, aux2)$

$\ \setminus\ valid\mathbf{.}up)$

$\quad \therefore\ goal \Leftrightarrow true$

$\ \setminus\ valid\mathbf{.}\ down$

$\quad \therefore\ goal$

$]$

B.2.4.6.

$\langle$ Proof of the third part of the invariant (CHECKIN). B.2.4.6 $\rangle \equiv$

$[\,goal\ := RevMax \geq \mathbf{card}(\mathbf{dom}\ K\ \mathbf{.}\ cont\ (update))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow goal$

$\ \setminus unfold(def\_sel_1)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}(new \mapsto f) \odot K\mathbf{.}\ cont\ (\overleftarrow{rg}))$

$\ \setminus unfold(domain\mathbf{.}recur)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(new \odot \mathbf{dom}\ K\ \mathbf{.}\ cont\ (\overleftarrow{rg}))$

$\ \setminus unfold(def\_card\mathbf{.}\ recur\ \mathbf{.}\ new\ (hyp\_pre\_new))$

$\quad \therefore goal \Leftrightarrow RevMax \geq succ(\mathbf{card}(\mathbf{dom}\ K\ \mathbf{.}\ cont\ (\overleftarrow{rg})))$

$\ \setminus unfold(valid\mathbf{.}\ up\ (gth\_prop(hyp\_pre\_safe)))$

$\quad \therefore goal \Leftrightarrow true$

$\ \setminus valid\mathbf{.}\ down$

$\quad \therefore goal$

$]$

B.2.4.7.

$\langle$ Proof of the fourth part of the invariant (CHECKIN). B.2.4.7 $\rangle \equiv$

$[\,goal\ := [\ r\ :\ Rid$

$\qquad\qquad \vdash r \in \mathbf{dom}\ K\ \mathbf{.}\ cont\ (update)$

$\qquad\qquad\quad \Rightarrow wff_F(K\mathbf{.}\ cont\ (update) \nabla r)$

$\qquad\qquad]$

$\vdash [\ r\ :\ Rid$

$\ \vdash [\ hyp \quad :\quad r \in \mathbf{dom}\ K\ \mathbf{.}\ cont\ (update)$

$\quad ;\ subgoal\ := wff_F(K\mathbf{.}\ cont\ (update) \nabla r)$

$\quad ;\ compl\ \ := \langle$ Case distinction is complete (CHECKIN). B.2.4.8 $\rangle$

$\quad \vdash \langle\!|\, [\ case\_a : r = new \vdash \langle$ Case $r = new$ (CHECKIN). B.2.4.9 $\rangle\,]$

$\qquad ,[\ case\_b\ : r \neq new \wedge r \in \mathbf{dom}\ K\ \mathbf{.}\ cont\ (\overleftarrow{rg})$

$\qquad \vdash \langle$ Case r unequal to new (CHECKIN). B.2.4.10 $\rangle$

$\qquad]$

$\qquad |\!\rangle$

$\qquad \setminus cased(compl)$

$\qquad\quad \therefore wff_F(K\mathbf{.}\ cont\ (update) \nabla r)$

$\quad]$

$\quad \setminus imp\mathbf{.}\ in$

$\ ]$

$\ \setminus univ\mathbf{.}\ in\ (P := goal)$

$\quad \therefore \forall [\ r : Rid\ \vdash goal(r)\,]$

$]$

B.2.4.8.

$\langle$ Case distinction is complete (CHECKIN). B.2.4.8 $\rangle \equiv$

hyp

$\quad \therefore r \in \mathbf{dom}\ K\ \mathbf{.}\ cont\ (update)$

$\setminus$ $subst(def\_sel_1)$

$\therefore r \in \mathbf{dom}(new \mapsto f) \odot K.\ cont\ (\overleftarrow{rg})$

$\setminus$ $subst(domain.recur)$

$\therefore r \in new \odot \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\setminus$ $subst(member\_prop.cons)$

$\therefore r = new \vee (r \neq new \wedge r \in \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg}))$

B.2.4.9.

$\langle$ Case $r = new$ (CHECKIN). B.2.4.9 $\rangle \equiv$

$refl$

$\therefore subgoal \Leftrightarrow wff_F(K.\ cont\ (update) \nabla r)$

$\setminus$ $unfold(def\_sel_1)$

$\therefore subgoal \Leftrightarrow wff_F(((new \mapsto f) \odot K.\ cont\ (\overleftarrow{rg})) \nabla r)$

$\setminus$ $unfold(case\_a)$

$\therefore subgoal \Leftrightarrow wff_F(((new \mapsto f) \odot K.\ cont\ (\overleftarrow{rg})) \nabla new)$

$\setminus$ $unfold(app.first)$

$\therefore subgoal \Leftrightarrow wff_F(f)$

$\setminus$ $unfold(valid.\ up\ (hyp\_pre\_wff))$

$\therefore subgoal \Leftrightarrow true$

$\setminus$ $valid.\ down$

$\therefore subgoal$

B.2.4.10.

$\langle$ Case r unequal to new (CHECKIN). B.2.4.10 $\rangle \equiv$

[$aux\ := imp\ .\ out\ (univ.\ out\ (hyp\_inv\_wff, r), pRight(case\_b))$

$\therefore wff_F(K.\ cont\ (\overleftarrow{rg}) \nabla r)$

$\vdash refl$

$\therefore subgoal \Leftrightarrow wff_F(K.\ cont\ (update) \nabla r)$

$\setminus$ $unfold(def\_sel_1)$

$\therefore subgoal \Leftrightarrow wff_F(((new \mapsto f) \odot K.\ cont\ (\overleftarrow{rg})) \nabla r)$

$\setminus$ $unfold(app.\ recur\ (case\_b))$

$\therefore subgoal \Leftrightarrow wff_F(K.\ cont\ (\overleftarrow{rg}) \nabla r)$

$\setminus$ $unfold(valid.\ up\ (aux))$

$\therefore subgoal \Leftrightarrow true$

$\setminus$ $valid.\ down$

$\therefore subgoal$

]

B.3 1st reification step: validity proofs

B.3.1 Reset operation

$\langle$ Proof of $val\_op\ (D_{op}.RESET, D_{mod}.inv)$. B.3.1 $\rangle \equiv$

$[\, in\ :\ \mathbf{void}\, ;\overleftarrow{rg} :\ D_{st}$

$\vdash [\, D_{mod}\, .\, inv\ (\overleftarrow{rg})$

$;\ true$

$new\ :=\ D\ .\ mk\ (\langle\rangle, \tau)$

$\vdash \langle\!|\ satisfiability\ :=$ ⟨ Proof of satisfiability (D:RESET). B.3.1.1 ⟩

$,\ preservation\ :=$ ⟨ Proof of invariant preservation (D:RESET). B.3.1.2 ⟩

$|\!\rangle$

$]$

$]$

$\therefore val\_op(D_{op}.RESET, D_{mod}.inv)$

B.3.1.1.

⟨ Proof of satisfiability (D:RESET). B.3.1.1 ⟩ ≡

$refl - \therefore new = new$

$\therefore D_{op}.\ RESET\ (in, \overleftarrow{rg}).\ post\ (\_\_, new)$

$\setminus\ ex2\_eq\_intro$

$\therefore \exists_2[\, out : \mathbf{void}\, ; rg :\ D_{st}\ \vdash D_{op}.\ RESET\ (in, \overleftarrow{rg}).\ post\ (out, rg)\,]$

B.3.1.2.

⟨ Proof of invariant preservation (D:RESET). B.3.1.2 ⟩ ≡

$[\, out\ :\ \mathbf{void}\, ; rg :\ D_{st}$

$\vdash [\, hyp\_post\ : rg = new$

$\vdash \langle\!|$⟨ Proof of the first part of the invariant (D:RESET). B.3.1.3 ⟩

$\therefore \mathbf{dom}\, D\ .\ cont\ (new) = info(D.\ dep\ (new))$

, ⟨ Proof of the second part of the invariant (D:RESET). B.3.1.4 ⟩

$\therefore nodup(D.\ dep\ (new))$

, ⟨ Proof of the third part of the invariant (D:RESET). B.3.1.5 ⟩

$\therefore RevMax \geq \mathbf{card}(\mathbf{dom}\, D\ .\ cont\ (new))$

, ⟨ Proof of the fourth part of the invariant (D:RESET). B.3.1.6 ⟩

$\therefore \forall [\ r\ \ :\ \ Rid$

$;\ del\ :=\ D\ .\ cont\ (new) \nabla\ r$

$\vdash r \in \mathbf{dom}\, D\ .\ cont\ (new) \Rightarrow (w\!f\!f_{\Delta}(del) \wedge w\!f\!f_{F}(changed(del)))$

$]$

$|\!\rangle$

$\setminus\ and4$

$\therefore D_{mod}.\ inv\ (new)$

$\setminus\ rsubst(hyp\_post)$

$\therefore D_{mod}.\ inv\ (rg)$

$]$

$]$

B.3.1.3.

$\langle$ Proof of the first part of the invariant (D:RESET). B.3.1.3 $\rangle \equiv$

$[\ lhs\ := \mathbf{dom}\ D\ .\ cont\ (new);\ rhs\ :=\ info\ (D.\ dep\ (new))$

$\vdash \langle\!|\ refl$

$\quad \therefore\ lhs = \mathbf{dom}\ D\ .\ cont\ (new)$

$\quad \backslash\ unfold(def\_sel_1)$

$\quad \therefore\ lhs = \mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} \Delta(Line))$

$\quad \backslash\ unfold(domain.empty)$

$\quad \therefore\ lhs = \{\ \}$

$,\ refl$

$\quad \therefore\ rhs = info(D.\ dep\ (new))$

$\quad \backslash\ unfold(def_s el_2)$

$\quad \therefore\ rhs = info(\tau)$

$\quad \backslash\ unfold(def\_info.empty)$

$\quad \therefore\ rhs = \{\ \}$

$|\!\rangle$

$\backslash\ indirect\_product$

$\quad \therefore\ lhs = rhs$

$]$

B.3.1.4.

$\langle$ Proof of the second part of the invariant (D:RESET). B.3.1.4 $\rangle \equiv$

$[\ goal\ :=\ nodup\ (D.\ dep\ (new))$

$\vdash\ refl$

$\quad \therefore\ goal \Leftrightarrow nodup(D.\ dep\ (new))$

$\backslash\ unfold(def_s el_2)$

$\quad \therefore\ goal \Leftrightarrow nodup(\tau \therefore tree(Rid))$

$\backslash\ unfold(def\_nodup.empty)$

$\quad \therefore\ goal \Leftrightarrow true$

$\backslash\ valid.\ down$

$\quad \therefore\ goal$

$]$

B.3.1.5.

$\langle$ Proof of the third part of the invariant (D:RESET). B.3.1.5 $\rangle \equiv$

$[\,goal\ := RevMax \geq \mathbf{card}(\mathbf{dom}\ D\,\textbf{.}\,cont\ (new))$
$\vdash refl$
  $\therefore goal \Leftrightarrow goal$
$\setminus\ unfold(def\_sel_1)$
  $\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} \Delta(Line)))$
$\setminus\ unfold(domain\textbf{.}empty)$
  $\therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\{\,\} \therefore set(Rid))$
$\setminus\ unfold(def\_card\textbf{.}empty)$
  $\therefore goal \Leftrightarrow RevMax \geq 0$
$\setminus\ unfold(valid\textbf{.}\ up\ (geq\_prop(RevMax_{pos})))$
  $\therefore goal \Leftrightarrow true$
$\setminus\ valid\textbf{.}\ down$
  $\therefore goal$
$]$

B.3.1.6.

$\langle$ Proof of the fourth part of the invariant (D:RESET). B.3.1.6 $\rangle \equiv$
$[\,goal\ := [\ r \quad : \quad Rid$
$\quad ;\ del\ :=\ D\,\textbf{.}\,cont\ (new)\nabla\ r$
$\quad \vdash r \in \mathbf{dom}\ D\,\textbf{.}\,cont\ (new)$
$\qquad \Rightarrow (wff_\Delta(del) \wedge wff_F(changed(del)))$
$\quad]$
$\vdash [\ r \quad : \quad Rid$
$\ ;\ del\ :=\ D\,\textbf{.}\,cont\ (new)\nabla\ r$
$\ \vdash [\,hyp\ : r \in \mathbf{dom}\ D\,\textbf{.}\,cont\ (new)$
$\quad \vdash hyp$
$\qquad \therefore r \in \mathbf{dom}\ D\,\textbf{.}\,cont\ (new)$
$\quad \setminus\ subst(def\_sel_1)$
$\qquad \therefore r \in \mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} \Delta(Line))$
$\quad \setminus\ subst(domain\textbf{.}empty)$
$\qquad \therefore r \in \{\,\}$
$\quad \setminus\ not\textbf{.}\ out\ (member\textbf{.}empty)$
$\qquad \therefore false$
$\quad \setminus\ false\_out$
$\qquad \therefore wff_\Delta(D\textbf{.}\ cont\ (new)\nabla r) \wedge wff_F(changed(del))$
$\ \ ]$
$\ \ \setminus\ imp\textbf{.}\ in$
$\ ]$
$\setminus\ univ\textbf{.}\ in$
  $\therefore \forall[\ r : Rid\ \vdash goal(r)\,]$
$]$

B.3.1.7.

B.3.2 Open operation

$\langle$ Proof of $val\_op\,(D_{op}.OPEN, D_{mod}.inv)$. B.3.2 $\rangle \equiv$

$[\ in\ :\quad File \otimes Rid\ ;\ \overleftarrow{rg} :\quad D_{st}$

$;\ f\ \ := sel_1(in);\qquad r\ := sel_2(in)$

$\vdash[\ new\ := D\ .\ mk\ (\langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle, node(r, \langle\rangle))$

$\quad hyp\_inv\ :\ D_{mod}\ .\ inv\ (\overleftarrow{rg});$

$\quad hyp\_pre\ :\ D_{op}\ .\ OPEN\ (in, \overleftarrow{rg}).\ pre$

$\quad\vdash \langle\!\langle\ satisfiability\ := \langle$ Proof of satisfiability (D:OPEN). B.3.2.1 $\rangle$

$\quad\ ,\ preservation\ := \langle$ Proof of invariant preservation (D:OPEN). B.3.2.2 $\rangle$

$\quad\ \rangle\!\rangle$

$\ \ ]$

$]$

$\quad \therefore val\_op(D_{op}.OPEN, D_{mod}.inv)$

B.3.2.1.

$\langle$ Proof of satisfiability (D:OPEN). B.3.2.1 $\rangle \equiv$

$refl - \therefore new = new$

$\quad \therefore D_{op}.\ OPEN\ (in, \overleftarrow{rg}).\ post\ (\_, new)$

$\setminus\ ex2\_eq\_intro$

$\quad \therefore \exists_2[\ out : \mathbf{void}\ ;\ rg :\ D_{st}\ \vdash D_{op}.\ OPEN\ (in, \overleftarrow{rg}).\ post\ (out, rg)\,]$

B.3.2.2.

$\langle$ Proof of invariant preservation (D:OPEN). B.3.2.2 $\rangle \equiv$

$[\ out\ :\ \mathbf{void}\ ;\ rg :\ D_{st}$

$\vdash[\ hyp\_post\ : rg = new$

$\ \vdash \langle\!\langle\ \langle$ Proof of the first part of the invariant (D:OPEN). B.3.2.3 $\rangle$

$\qquad \therefore \mathbf{dom}\ D\ .\ cont\ (new) = info(D.\ dep\ (new))$

$\quad ,\langle$ Proof of the second part of the invariant (D:OPEN). B.3.2.4 $\rangle$

$\qquad \therefore nodup(D.\ dep\ (new))$

$\quad ,\langle$ Proof of the third part of the invariant (D:OPEN). B.3.2.5 $\rangle$

$\qquad \therefore RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ .\ cont\ (new))$

$\quad ,\langle$ Proof of the fourth part of the invariant (D:OPEN). B.3.2.6 $\rangle$

$\qquad \therefore \forall[\ r\quad :\quad Rid$

$\qquad\quad ;\ del\ := D\ .\ cont\ (new) \nabla\ r$

$\qquad\quad \vdash r \in \mathbf{dom}\ D\ .\ cont\ (new) \Rightarrow (wff_{\Delta}(del) \wedge wff_F(changed(del)))$

$\qquad\quad]$

$\quad \rangle\!\rangle$

$\quad \setminus\ and4$

$\qquad \therefore D_{mod}.\ inv\ (new)$

$\quad \setminus\ rsubst(hyp\_post)$

$\qquad \therefore D_{mod}.\ inv\ (rg)$

$\ \ ]$

$]$

B.3.2.3.

$\langle$ Proof of the first part of the invariant (D:OPEN). B.3.2.3$\rangle \equiv$

$[\, lhs := \mathbf{dom}\, D\,.\, cont\ (new);\ rhs := info\ (D.\, dep\ (new))$

$\vdash \langle\!| \ refl$

$\quad\quad \therefore lhs = \mathbf{dom}\, D\,.\, cont\ (new)$

$\quad \setminus unfold(def\_sel_1)$

$\quad\quad \therefore lhs = \mathbf{dom}\langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle$

$\quad \setminus unfold(dom\_prop.single)$

$\quad\quad \therefore lhs = \{r\}$

$\ \ ,\ refl$

$\quad\quad \therefore rhs = info(D.\, dep\ (new))$

$\quad \setminus unfold(def_s el_2)$

$\quad\quad \therefore rhs = info(node(r, \langle\rangle))$

$\quad \setminus unfold(info\_prop.single)$

$\quad\quad \therefore rhs = \{r\}$

$\ \ |\!\rangle$

$\ \ \setminus indirect\_product$

$\quad \therefore lhs = rhs$

$]$

B.3.2.4.

$\langle$ Proof of the second part of the invariant (D:OPEN). B.3.2.4$\rangle \equiv$

$[\, goal := nodup\ (D.\, dep\ (new))$

$\vdash refl$

$\quad\quad \therefore goal \Leftrightarrow nodup(D.\, dep\ (new))$

$\quad \setminus unfold(def_s el_2)$

$\quad\quad \therefore goal \Leftrightarrow nodup(node(r, \langle\rangle))$

$\quad \setminus unfold(nodup\_prop.single)$

$\quad\quad \therefore goal \Leftrightarrow true$

$\quad \setminus valid.\, down$

$\quad\quad \therefore goal$

$]$

B.3.2.5.

$\langle$ Proof of the third part of the invariant (D:OPEN). B.3.2.5$\rangle \equiv$

$[\, goal\ := RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ \textbf{.}\ cont\ (new))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ \textbf{.}\ cont\ (new))$

$\setminus\ unfold(def\_sel_1)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle)$

$\setminus\ unfold(dom\_prop\textbf{.}single)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\{r\})$

$\setminus\ unfold(sing\textbf{.card})$

$\quad \therefore goal \Leftrightarrow RevMax \geq 1$

$\setminus\ unfold(valid\textbf{.}\ up\ (RevMax_{pos}))$

$\quad \therefore goal \Leftrightarrow true$

$\setminus\ valid\textbf{.}\ down$

$\quad \therefore goal$

$]$

B.3.2.6.

$\langle$ Proof of the fourth part of the invariant (D:OPEN). B.3.2.6 $\rangle \equiv$

$[\ goal\ := [\ r\ :\ Rid$
$;\ del\ := D\,.\,cont\ (new) \nabla r$
$\vdash r \in \mathbf{dom}\ D\,.\,cont\ (new) \Rightarrow (wff_{\Delta}(del) \wedge wff_F(changed(del)))$
$]$
$\vdash[\ r1\ :\ Rid$
$;\ del\ := D\,.\,cont\ (new) \nabla r1$
$\vdash[\ hyp\ :\ r1 \in \mathbf{dom}\ D\,.\,cont\ (new)$
$;\ side_A\ := \langle$ Side deduction A (D:OPEN). B.3.2.8 $\rangle \therefore r1 = r$
$;\ side_B\ := \langle$ Side deduction B (D:OPEN). B.3.2.7 $\rangle \therefore del = \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\ \rangle$
$;\ subgoal\ := wff_{\Delta}(del) \wedge wff_F(changed(del))$
$\vdash refl$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(del) \wedge wff_F(changed(del))$
$\backslash\ unfold(side_B)$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(del) \wedge wff_F(changed(\langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\ \rangle))$
$\backslash\ unfold(def\_changed\,.\,unit)$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(del) \wedge wff_F(f)$
$\backslash\ unfold(valid.\ up\ (pRight(hyp\_pre)))$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(del) \wedge true$
$\backslash\ unfold(simp\_andR)$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(del)$
$\backslash\ unfold(side_B)$
$\therefore subgoal \Leftrightarrow wff_{\Delta}(\langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\ \rangle)$
$\backslash\ unfold(prop\_ok\_delta)$
$\therefore subgoal \Leftrightarrow wff_{\Delta_u}(\langle\ 1, f, 0\ \rangle_{\Delta_u})$
$\backslash\ unfold(prop\_unit\,.\,wff)$
$\therefore subgoal \Leftrightarrow true$
$\backslash\ valid.\ down$
$\therefore subgoal$
$]$
$\backslash\ imp.\ in$
$]$
$\backslash\ univ.\ in\ (P := goal)$
$]$

B.3.2.7.

$\langle$ Side deduction B (D:OPEN). B.3.2.7 $\rangle \equiv$
$refl$
$\therefore del = D.\ cont\ (new) \nabla r1$
$\backslash\ unfold(def\_sel_1)$
$\therefore del = \langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle \nabla r1$
$\backslash\ unfold(side_A)$
$\therefore del = \langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle \nabla r$
$\backslash\ unfold(app.first)$
$\therefore del = \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\ \rangle$

B.3.2.8.

$\langle$ Side deduction A (D:OPEN). B.3.2.8 $\rangle \equiv$

hyp

$\quad \therefore r1 \in \mathbf{dom}\, D\,.\, cont\ (new)$

$\setminus\ subst(def\_sel_1)$

$\quad \therefore r1 \in \mathbf{dom}\langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle$

$\setminus\ subst(dom\_prop.single)$

$\quad \therefore r1 \in \{r\}$

$\setminus\ subst(member\_prop.single)$

$\quad \therefore r1 = r$

B.3.3 Checkout operation

$\langle$ Proof of $val\_op\ (D_{op}.CHECKOUT, D_{mod}.inv)$. B.3.3 $\rangle \equiv$

$[\ r\ :\ Rid\ ;\ \overleftarrow{rg}\ :\ D_{st}$

$\vdash [\ res\ := \ retr\_rev_D\ (D.\, cont\ (\overleftarrow{rg}), D.\, dep\ (\overleftarrow{rg}), r)$

$\quad hyp\_inv\ :\ D_{mod}\,.\, inv\ (\overleftarrow{rg});$

$\quad hyp\_pre\ :\ D_{op}\,.\, CHECKOUT\ (r, \overleftarrow{rg}).\, pre$

$\quad \vdash \langle\!|\ satisfiability\ := \langle$ Proof of satisfiability (D:CHECKOUT). B.3.3.1 $\rangle$

$\quad\ ,\ preservation\ := \langle$ Proof of invariant preservation (D:CHECKOUT). B.3.3.2 $\rangle$

$\quad |\!\rangle$

$\ ]$

$]$

$\quad \therefore val\_op(D_{op}.CHECKOUT, D_{mod}.inv)$

B.3.3.1.

$\langle$ Proof of satisfiability (D:CHECKOUT). B.3.3.1 $\rangle \equiv$

$\langle\!|\ refl \qquad\qquad ,\ refl \qquad\qquad |\!\rangle$

$\quad \therefore res = res \qquad \therefore \overleftarrow{rg} = \overleftarrow{rg}$

$\setminus\ and.\, in$

$\quad \therefore D_{op}.\, CHECKOUT\ (r, \overleftarrow{rg}).\, post\ (res, \overleftarrow{rg})$

$\setminus\ ex2\_eq2\_intro$

$\quad \therefore \exists_2[\ f : File\ ;\ rg :\ D_{st}\ \vdash D_{op}.\, CHECKOUT\ (r, \overleftarrow{rg}).\, post\ (f, rg)\,]$

B.3.3.2.

$\langle$ Proof of invariant preservation (D:CHECKOUT). B.3.3.2 $\rangle \equiv$

$$
\begin{array}{l}
[\, f \;:\; File \,;\, rg : \; D_{st} \\
\vdash [\, hyp\_post \;: f = res \wedge rg = \overleftarrow{rg} \\
\quad \vdash hyp\_inv \\
\qquad \therefore D_{mod}.\, inv\ (\overleftarrow{rg}) \\
\quad\;\; \setminus\ rsubst(pRight(hyp\_post)) \\
\qquad \therefore D_{mod}.\, inv\ (rg) \\
\quad] \\
]
\end{array}
$$

B.3.4 Checkin operation

$$
\begin{array}{l}
\langle\, \text{Proof of } \; val\_op\ (D_{op}.CHECKIN, D_{mod}.inv).\ \text{B.3.4}\, \rangle \equiv \\
[\, in \;:\; File \otimes (Rid \otimes Rid);\, \overleftarrow{rg} :\; D_{st} \\
\vdash [\, f \;:= sel_1(in);\; or \;:= sel_1(sel_2(in));\; nr \;:= sel_2(sel_2(in)) \\
\quad \vdash [\; del \;\; := \; diff\ (retr\_rev_D(D.\, cont\ (\overleftarrow{rg}), D.\, dep\ (\overleftarrow{rg}), or), f) \\
\quad\;\; ;\ new \;:= \; D\,.\, mk\ ((nr \mapsto del) \odot D.\, cont\ (\overleftarrow{rg}), insert(or, nr, D.\, dep\ (\overleftarrow{rg}))) \\
\qquad \vdash [\; hyp\_inv \;:\; D_{mod}\,.\, inv\ (\overleftarrow{rg}) \\
\qquad\;\; ;\ hyp\_pre \;:\; D_{op}\,.\, CHECKIN\ (in, \overleftarrow{rg}).\, pre \\
\qquad\;\; \langle\, \text{Some auxiliary deductions (D:CHECKIN).}\ \text{B.3.4.1}\, \rangle \\
\qquad \vdash \llangle\, satisfiability \;:= \langle\, \text{Proof of satisfiability (D:CHECKIN).}\ \text{B.3.4.2}\, \rangle \\
\qquad\quad ,\ preservation \;:= \langle\, \text{Proof of invariant preservation (D:CHECKIN).}\ \text{B.3.4.3}\, \rangle \\
\qquad\quad \rrangle \\
\qquad] \\
\quad\;\;] \\
\quad] \\
] \\
\quad \therefore val\_op(D_{op}.CHECKIN, D_{mod}.inv)
\end{array}
$$

B.3.4.1.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\langle\, \text{Some auxiliary deductions (D:CHECKIN).}\ \text{B.3.4.1}\, \rangle \equiv} \\
\llbracket\ hyp\_inv\_dom & := & pLeft\ (pLeft(pLeft(hyp\_inv))) \\
;\ hyp\_inv\_unique & := & pRight\ (pLeft(pLeft(hyp\_inv))) \\
;\ hyp\_inv\_wff & := & pRight\ (hyp\_inv) \\
;\ hyp\_pre\_old & := & pLeft\ (pLeft(pLeft(hyp\_pre))) \\
;\ hyp\_pre\_new & := & pRight\ (pLeft(pLeft(hyp\_pre))) \\
;\ hyp\_pre\_safe & := & pRight\ (pLeft(hyp\_pre)) \\
;\ hyp\_pre\_wff & := & pRight\ (hyp\_pre) \\
;\ aux1 & := & hyp\_pre\_old \\
 & & \quad \therefore or \in \mathbf{dom}\ D\,.\, cont\ (\overleftarrow{rg}) \\
 & & \setminus\ subst(hyp\_inv\_dom) \\
 & & \quad \therefore or \in info(D.\, dep\ (\overleftarrow{rg}))
\end{array}
$$

$; \ aux2 \quad := hyp\_pre\_new$

$\quad \therefore nr \notin \mathbf{dom}\, D\,.\,cont\,(\overleftarrow{rg})$

$\quad \backslash\ subst(hyp\_inv\_dom)$

$\quad \therefore nr \notin info(D.\,dep\,(\overleftarrow{rg}))$

$; \ del\_wff \quad := wff\_lemma\,(hyp\_inv, hyp\_pre\_old)$

$\quad \therefore wff_F(retr\_rev_D(D.\,cont\,(\overleftarrow{rg}), D.\,dep\,(\overleftarrow{rg}), or))$

$]\!]$

B.3.4.2.

$\langle$ Proof of satisfiability (D:CHECKIN). B.3.4.2 $\rangle \equiv$

$refl - \therefore new = new$

$\quad \therefore D_{op}.\,CHECKIN\,(in, \overleftarrow{rg}).\,post\,(\_, new)$

$\backslash\ ex2\_eq\_intro$

$\quad \therefore \exists_2[\,out : \mathbf{void}\,; rg : D_{st} \vdash D_{op}.\,CHECKIN\,(in, \overleftarrow{rg}).\,post\,(out, rg)\,]$

B.3.4.3.

$\langle$ Proof of invariant preservation (D:CHECKIN). B.3.4.3 $\rangle \equiv$

$[\,out : \mathbf{void}\,; rg : D_{st}$

$\vdash [\,hyp\_post : rg = new$

$\quad \vdash \langle\!| \langle$ Proof of the first part of the invariant (D:CHECKIN). B.3.4.4 $\rangle$

$\qquad \therefore \mathbf{dom}\, D\,.\,cont\,(new) = info(D.\,dep\,(new))$

$\quad ,\langle$ Proof of the second part of the invariant (D:CHECKIN). B.3.4.5 $\rangle$

$\qquad \therefore nodup(D.\,dep\,(new))$

$\quad ,\langle$ Proof of the third part of the invariant (D:CHECKIN). B.3.4.6 $\rangle$

$\qquad \therefore RevMax \geq \mathbf{card}(\mathbf{dom}\, D\,.\,cont\,(new))$

$\quad ,\langle$ Proof of the fourth part of the invariant (D:CHECKIN). B.3.4.7 $\rangle$

$\qquad \therefore \forall[\,r \quad : \quad Rid$

$\qquad\quad ; \ del_1 := D\,.\,cont\,(new) \nabla\, r$

$\qquad\quad \vdash r \in \mathbf{dom}\, D\,.\,cont\,(new) \Rightarrow (wff_\Delta(del_1) \wedge wff_F(changed(del_1)))$

$\qquad\quad]$

$\quad |\!\rangle$

$\quad \backslash\ and4$

$\qquad \therefore D_{mod}.\,inv\,(new)$

$\quad \backslash\ rsubst(hyp\_post)$

$\qquad \therefore D_{mod}.\,inv\,(rg)$

$\quad]$

$]$

B.3.4.4.

$\langle$ Proof of the first part of the invariant (D:CHECKIN). B.3.4.4 $\rangle \equiv$

$[\, lhs \ := \mathbf{dom}\, D \,.\, cont\ (new);\ rhs \ := \ info\ (D.\, dep\ (new))$

$\vdash \langle\!|\ refl$

$\therefore lhs = \mathbf{dom}\, D \,.\, cont\ (new)$

$\backslash\ unfold(def\_sel_1)$

$\therefore lhs = \mathbf{dom}(nr \mapsto del) \odot D.\, cont\ (\overleftarrow{rg})$

$\backslash\ unfold(domain.recur)$

$\therefore lhs = nr \odot \mathbf{dom}\, D \,.\, cont\ (\overleftarrow{rg})$

$,\ refl$

$\therefore rhs = info(D.\, dep\ (new))$

$\backslash\ unfold(def_s el_2)$

$\therefore rhs = info(insert(or, nr, D.\, dep\ (\overleftarrow{rg})))$

$\backslash\ unfold(info\_prop.\, insert\ (aux1))$

$\therefore rhs = nr \odot info(D.\, dep\ (\overleftarrow{rg}))$

$\backslash\ fold(hyp\_inv\_dom)$

$\therefore rhs = nr \odot \mathbf{dom}\, D \,.\, cont\ (\overleftarrow{rg})$

$|\!\rangle$

$\backslash\ indirect\_product$

$\therefore lhs = rhs$

$]$

B.3.4.5.

$\langle$ Proof of the second part of the invariant (D:CHECKIN). B.3.4.5 $\rangle \equiv$

$[\, goal \ := \ nodup\ (D.\, dep\ (new))$

$\vdash refl$

$\therefore goal \Leftrightarrow nodup(D.\, dep\ (new))$

$\backslash\ unfold(def_s el_2)$

$\therefore goal \Leftrightarrow nodup(insert(or, nr, D.\, dep\ (\overleftarrow{rg})))$

$\backslash\ unfold(nodup\_prop.\, insert\ (hyp\_inv\_unique, aux1, aux2)$

$\backslash\ valid.up)$

$\therefore goal \Leftrightarrow true$

$\backslash\ valid.\, down$

$\therefore goal$

$]$

B.3.4.6.

$\langle$ Proof of the third part of the invariant (D:CHECKIN). B.3.4.6 $\rangle \equiv$

$[\ goal\ := RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ .\ cont\ (new))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ .\ cont\ (new))$

$\ \setminus unfold(def\_sel_1)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}(nr \mapsto del) \odot D.\ cont\ (\overleftarrow{rg}))$

$\ \setminus unfold(domain.recur)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(nr \odot \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg}))$

$\ \setminus unfold(def\_card.\ recur\ .\ new\ (hyp\_pre\_new))$

$\quad \therefore goal \Leftrightarrow RevMax \geq succ(\mathbf{card}(\mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg})))$

$\ \setminus unfold(valid.\ up\ (gth\_prop(hyp\_pre\_safe)))$

$\quad \therefore goal \Leftrightarrow true$

$\ \setminus valid.\ down$

$\quad \therefore goal$

$]$

B.3.4.7.

$\langle$ Proof of the fourth part of the invariant (D:CHECKIN). B.3.4.7 $\rangle \equiv$

$[\ goal\ := [\ r\ :\ Rid$

$\qquad ;\ del_1\ :=\ D\ .\ cont\ (new) \nabla\ r$

$\qquad \vdash r \in \mathbf{dom}\ D\ .\ cont\ (new) \Rightarrow (wff_{\Delta}(del_1) \wedge wff_F(changed(del_1)))$

$\qquad]$

$\vdash [\ r\ :\ Rid$

$\ ;\ del_1\ :=\ D\ .\ cont\ (new) \nabla\ r$

$\ \vdash [\ hyp\ :\ r \in \mathbf{dom}\ D\ .\ cont\ (new)$

$\quad ;\ subgoal\ := wff_{\Delta}(del_1) \wedge wff_F(changed(del_1))$

$\quad ;\ compl\ := \langle$ Case distinction is complete (D:CHECKIN). B.3.4.8 $\rangle$

$\quad \vdash \langle\!| [\ case\_a : r = nr \vdash \langle$ Case $r = nr$. B.3.4.9 $\rangle]$

$\qquad , [\ case\_b\ : r \neq nr \wedge r \in \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg})$

$\qquad \vdash \langle$ Case r unequal to nr. B.3.4.11 $\rangle$

$\qquad]$

$\quad |\!\rangle$

$\quad \setminus cased(compl)$

$\qquad \therefore subgoal$

$\quad]$

$\quad \setminus imp.\ in$

$\ ]$

$\ \setminus univ.\ in\ (P := goal)$

$\quad \therefore \forall [\ r : Rid\ \vdash goal(r)\ ]$

$]$

B.3.4.8.

$\langle$ Case distinction is complete (D:CHECKIN). B.3.4.8 $\rangle \equiv$

hyp

$\therefore r \in \mathbf{dom}\, D\, \mathbf{.}\, cont\ (new)$

$\setminus$ $subst(def\_sel_1)$

$\therefore r \in \mathbf{dom}(nr \mapsto del) \odot D\mathbf{.}\ cont\ (\overleftarrow{rg})$

$\setminus$ $subst(domain\mathbf{.}recur)$

$\therefore r \in nr \odot \mathbf{dom}\, D\, \mathbf{.}\, cont\ (\overleftarrow{rg})$

$\setminus$ $subst(member\_prop\mathbf{.}cons)$

$\therefore r = nr \vee (r \neq nr \wedge r \in \mathbf{dom}\, D\, \mathbf{.}\, cont\ (\overleftarrow{rg}))$

B.3.4.9.

$\langle$ Case $r = nr$. B.3.4.9 $\rangle \equiv$

$[$ $side$ $:= \langle$ Side deduction A (D:CHECKIN) B.3.4.10 $\rangle$

$\therefore del_1 = del$

$\vdash$ $refl$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge wff_F(changed(del_1))$

$\setminus$ $unfold(side)$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge wff_F(changed(del))$

$\setminus$ $unfold(valid\mathbf{.}\ up\ (prop\_diff\_wff(del\_wff, hyp\_pre\_wff)))$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge true$

$\setminus$ $unfold(simp\_andR)$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1)$

$\setminus$ $unfold(side)$

$\therefore subgoal \Leftrightarrow wff_\Delta(del)$

$\setminus$ $unfold(valid\mathbf{.}\ up\ (pLeft(prop\_diff)))$

$\therefore subgoal \Leftrightarrow true$

$\setminus$ $valid\mathbf{.}\ down$

$\therefore subgoal$

$]$

B.3.4.10.

$\langle$ Side deduction A (D:CHECKIN) B.3.4.10 $\rangle \equiv$

$refl$

$\therefore del_1 = del_1$

$\setminus$ $unfold(def\_sel_1)$

$\therefore del_1 = ((nr \mapsto del) \odot D\mathbf{.}\ cont\ (\overleftarrow{rg}))\nabla\ r$

$\setminus$ $unfold(case\_a)$

$\therefore del_1 = ((nr \mapsto del) \odot D\mathbf{.}\ cont\ (\overleftarrow{rg}))\nabla\ nr$

$\setminus$ $unfold(app\mathbf{.}first)$

$\therefore del_1 = del$

B.3.4.11.

$\langle$ Case r unequal to nr. B.3.4.11 $\rangle \equiv$

[$side$:= ⟨Side deduction B (D:CHECKIN) B.3.4.12⟩

$\therefore del_1 = D.\ cont\ (\overleftarrow{rg})\nabla\ r$

; aux_1 := $pRight\ (imp.\ out\ (univ.\ out\ (hyp\_inv\_wff, r), pRight(case\_b)))$

$\therefore wff_F(changed(D.\ cont\ (\overleftarrow{rg})\nabla r))$

; aux_2 := $pLeft\ (imp.\ out\ (univ.\ out\ (hyp\_inv\_wff, r), pRight(case\_b)))$

$\therefore wff_\Delta(D.\ cont\ (\overleftarrow{rg})\nabla r)$

$\vdash refl$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge wff_F(changed(del_1))$

$\setminus\ unfold(side)$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge wff_F(changed(D.\ cont\ (\overleftarrow{rg})\nabla r))$

$\setminus\ unfold(valid.\ up\ (aux_1))$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1) \wedge true$

$\setminus\ unfold(simp\_andR)$

$\therefore subgoal \Leftrightarrow wff_\Delta(del_1)$

$\setminus\ unfold(side)$

$\therefore subgoal \Leftrightarrow wff_\Delta(D.\ cont\ (\overleftarrow{rg})\nabla r)$

$\setminus\ unfold(valid.\ up\ (aux_2))$

$\therefore subgoal \Leftrightarrow true$

$\setminus\ valid.\ down$

$\therefore subgoal$

]

B.3.4.12.

⟨Side deduction B (D:CHECKIN) B.3.4.12⟩ ≡

$refl$

$\therefore del_1 = del_1$

$\setminus\ unfold(def\_sel_1)$

$\therefore del_1 = ((nr \mapsto del) \odot D.\ cont\ (\overleftarrow{rg}))\nabla\ r$

$\setminus\ unfold(app.\ recur\ (case\_b))$

$\therefore del_1 = D.\ cont\ (\overleftarrow{rg})\nabla\ r$

B.4 2nd. reification: retrieve validity

B.4.1 Auxiliary lemmas

⟨Projection lemmas. B.4.1⟩ ≡

⟦ get_fd_dep := [$rg\ ?\ D_{st}$

$\vdash refl$

$\therefore K.\ dep\ (retr_D(rg)) = K.\ dep\ (retr_D(rg))$

$\setminus\ unfold(def_s el_2)$

$\therefore K.\ dep\ (retr_D(rg)) = D.\ dep\ (rg)$

]

$; \ \mathit{get\_fd\_cont} \ := [\ rg \ ? \ D_{st}$

$\vdash \mathit{refl}$

$\therefore K.\ \mathit{cont}\ (\mathit{retr}_D(rg)) = K.\ \mathit{cont}\ (\mathit{retr}_D(rg))$

$\setminus \ \mathit{unfold}(\mathit{def\_sel}_1)$

$\therefore K.\ \mathit{cont}\ (\mathit{retr}_D(rg))$

$= \mathit{atm}(\mathit{retr\_rev}_D(D.\ \mathit{cont}\ (rg), D.\ \mathit{dep}\ (rg)), \mathbf{dom}\ D\ .\ \mathit{cont}\ (rg))$

$]$

$]\!]$

B.4.1.1.

$\langle$ Retrieve lemma. B.4.1.1 $\rangle \equiv$

$\mathit{retr\_lemma} \ := [\ m \ ? \ \mathit{Rid} \xrightarrow{m} \Delta(\mathit{Line}); \ t \ ? \ \mathit{tree}\ (\mathit{Rid}); r1, r2, r3 \ ? \ \mathit{Rid} \ ; \ d \ ? \ \Delta(\mathit{Line})$

$\vdash [\ \mathit{hypu} \ : \ \mathit{nodup}\ (t)$

$; \ \mathit{hyp1} \ : \ r1 \in \mathit{info}(t)$

$; \ \mathit{hyp2} \ : \ r2 \in \mathit{info}(t)$

$; \ \mathit{hyp3} \ : \ r3 \notin \mathit{info}(t)$

$; \ \mathit{lemma} \ :=$ $\langle$ Side deduction. B.4.1.2 $\rangle$

$\therefore r3 \notin \mathrm{elems}\ \mathit{init\_path}\ (r1, t)$

$\vdash [\ \mathit{lhs} \ := \ \mathit{retr\_rev}_D\ ((r3 \mapsto d) \odot m, \mathit{insert}(r2, r3, t), r1)$

$\vdash \mathit{refl}$

$\therefore \mathit{lhs} = \langle\rangle \oplus_s (((r3 \mapsto d) \odot m) * \mathit{init\_path}(r1, \mathit{insert}(r2, r3, t)))$

$\setminus \ \mathit{unfold}(\mathit{init\_path\_prop}(\mathit{hypu}, \mathit{hyp1}, \mathit{hyp3}).\ \mathit{inv}\ (\mathit{hyp2}))$

$\therefore \mathit{lhs} = \langle\rangle \oplus_s (((r3 \mapsto d) \odot m) * \mathit{init\_path}(r1, t))$

$\setminus \ \mathit{unfold}(\mathit{map\_map\_prop}.\ \mathit{reduce}\ (\mathit{lemma}))$

$\therefore \mathit{lhs} = \langle\rangle \oplus_s (m * \mathit{init\_path}(r1, t))$

$\therefore \mathit{lhs} = \mathit{retr\_rev}_D(m, t, r1)$

$]$

$]$

$]$

B.4.1.2.

$\langle$ Side deduction. B.4.1.2 $\rangle \equiv$

$[\ \mathit{hyp} \ : r3 \in \mathrm{elems}\ \mathit{init\_path}\ (r1, t)$

$\vdash \mathit{hyp}$

$\setminus \ \mathit{subset\_prop}.\ \mathit{weaken}\ (\mathit{path\_incl})$

$\therefore r3 \in \mathit{info}(t)$

$\setminus \ \mathit{not}.\ \mathit{out}\ (\mathit{hyp3})$

$\therefore \mathit{false}$

$]$

$\setminus \ \mathit{not}.\ \mathit{in}$

B.4.2 Validity of the retrieve function

$\langle$ Proof of $\mathit{val\_retr}\ (D_{mod}.\mathit{inv}, K_{mod}.\mathit{inv}, \mathit{retr}_D)$. B.4.2 $\rangle \equiv$

$\langle\!|$ *preservation* $:= [\ rg_D \quad ?\quad D_{st}$

$\quad ;\ hyp\_inv :\quad D_{mod}\,.\,inv\ (rg_D)$

$\quad ;\ rg \quad := retr_D\ (rg_D)$

$\quad ;\ retr \quad := retr\_rev_D\ (D.\,cont\ (rg_D), D.\,dep\ (rg_D))$

$\quad ;\langle$ Auxiliary lemmas (preservation). B.4.2.1 $\rangle$

$\quad \vdash \langle\!|\langle$ Proof of invariant preservation (1). B.4.2.2 $\rangle$

$\quad ,\langle$ Proof of invariant preservation (2). B.4.2.3 $\rangle$

$\quad ,\langle$ Proof of invariant preservation (3). B.4.2.4 $\rangle$

$\quad ,\langle$ Proof of invariant preservation (4). B.4.2.5 $\rangle$

$\quad |\!\rangle$

$\quad \setminus\ and4$

$\quad\quad \therefore K_{mod}.\,inv\ (rg)$

$\quad]$

$,\ completeness := [\ rg \quad ?\ K_{st}$

$\quad ;\ inv\_hyp\ :\ K_{mod}\,.\,inv\ (rg)$

$\quad \vdash \langle\!|\ prop\_convert\,.\,invar\ (inv\_hyp)$

$\quad ,\ prop\_convert\,.\,inverse\ (rg := rg)$

$\quad |\!\rangle$

$\quad \setminus\ and.\,in$

$\quad \setminus\ hide(x := convert(rg))$

$\quad\quad \therefore \exists[\ rg_D : D_{st}\ \vdash D_{mod}.\,inv\ (rg_D) \wedge retr_D(rg_D) = rg\,]$

$\quad]$

$|\!\rangle$

$\quad \therefore val\_retr(D_{mod}.inv, K_{mod}.inv, retr_D)$

B.4.2.1.

$\langle$ Auxiliary lemmas (preservation). B.4.2.1 $\rangle \equiv$

$[\![\ aux \quad := [\ lhs := \mathbf{dom}\ K\,.\,cont\ (rg)$

$\quad \vdash refl$

$\quad\quad \therefore lhs = \mathbf{dom}\ K\,.\,cont\ (rg)$

$\quad \setminus\ unfold(get\_fd\_cont)$

$\quad\quad \therefore lhs = \mathbf{dom}\ atm\ (retr, \mathbf{dom}\ D\,.\,cont\ (rg_D))$

$\quad \setminus\ unfold(abs\_to\_map\_prop.dom)$

$\quad\quad \therefore lhs = \mathbf{dom}\ D\,.\,cont\ (rg_D)$

$\quad]$

$\quad \therefore \mathbf{dom}\ K\,.\,cont\ (rg) = \mathbf{dom}\ D\,.\,cont\ (rg_D)$

$;\ hyp\_inv\_dom \quad := pLeft\ (pLeft(pLeft(hyp\_inv)))$

$;\ hyp\_inv\_unique := pRight\ (pLeft(pLeft(hyp\_inv)))$

$;\ hyp\_inv\_max \quad := pRight\ (pLeft(hyp\_inv))$

$;\ hyp\_inv\_wff \quad := pRight\ (hyp\_inv)$

$]\!]$

B.4.2.2.

$\langle$ Proof of invariant preservation (1). B.4.2.2 $\rangle \equiv$

$[\, goal \ := \mathbf{dom}\ K\ .\ cont\ (rg) = info(K.\ dep\ (rg))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow \mathbf{dom}\ K\ .\ cont\ (rg) = info(K.\ dep\ (rg))$

$\setminus\ unfold(get\_fd\_dep)$

$\quad \therefore goal \Leftrightarrow \mathbf{dom}\ K\ .\ cont\ (rg) = info(D.\ dep\ (rg_D))$

$\setminus\ unfold(aux)$

$\quad \therefore goal \Leftrightarrow \mathbf{dom}\ D\ .\ cont\ (rg_D) = info(D.\ dep\ (rg_D))$

$\setminus\ unfold(valid.\ up\ (hyp\_inv\_dom))$

$\quad \therefore goal \Leftrightarrow true$

$\setminus\ valid.\ down$

$]$

B.4.2.3.

$\langle$ Proof of invariant preservation (2). B.4.2.3 $\rangle \equiv$

$[\, goal \ := \ nodup\ (K.\ dep\ (rg))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow nodup(K.\ dep\ (rg))$

$\setminus\ unfold(get\_fd\_dep)$

$\quad \therefore goal \Leftrightarrow nodup(D.\ dep\ (rg_D))$

$\setminus\ unfold(valid.\ up\ (hyp\_inv\_unique))$

$\quad \therefore goal \Leftrightarrow true$

$\setminus\ valid.\ down$

$]$

B.4.2.4.

$\langle$ Proof of invariant preservation (3). B.4.2.4 $\rangle \equiv$

$[\, goal \ := RevMax \geq \mathbf{card}(\mathbf{dom}\ K\ .\ cont\ (rg))$

$\vdash refl$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\ K\ .\ cont\ (rg))$

$\setminus\ unfold(aux)$

$\quad \therefore goal \Leftrightarrow RevMax \geq \mathbf{card}(\mathbf{dom}\ D\ .\ cont\ (rg_D))$

$\setminus\ unfold(valid.\ up\ (hyp\_inv\_max))$

$\quad \therefore goal \Leftrightarrow true$

$\setminus\ valid.\ down$

$]$

B.4.2.5.

$\langle$ Proof of invariant preservation (4). B.4.2.5 $\rangle \equiv$

$[\, r \,:\, Rid$

$\vdash [\; hyp \;:\; r \in \mathbf{dom}\, K\,.\, cont\,(rg)$

$\;;\; hyp2 \;:=\; (hyp$

$\qquad\qquad \setminus\, subst(aux)) \;\therefore\; r \in \mathbf{dom}\, D\,.\, cont\,(rg_D)$

$\;;\; goal \;:=\; wff_F(K.\, cont\,(rg)\nabla r)$

$\;\vdash refl$

$\qquad \therefore goal \Leftrightarrow wff_F(K.\, cont\,(retr_D(rg_D))\nabla r)$

$\quad \setminus\, unfold(def\_sel_1)$

$\qquad \therefore goal \Leftrightarrow wff_F(atm(retr, \mathbf{dom}\, D\,.\, cont\,(rg_D))\nabla r)$

$\quad \setminus\, unfold(abs\_to\_map\_prop.\, apply\,(hyp2))$

$\qquad \therefore goal \Leftrightarrow wff_F(retr(r))$

$\quad \setminus\, unfold(valid.\, up\,(wff\_lemma(hyp\_inv, hyp2)))$

$\qquad \therefore goal \Leftrightarrow true$

$\quad \setminus\, valid.\, down$

$\qquad \therefore goal$

$\;]$

$\;\setminus\, imp.\, in$

$]$

$\setminus\, univ.\, in\,(P := [\, r : Rid \,\vdash\, r \in \mathbf{dom}\, K\,.\, cont\,(rg) \Rightarrow wff_F(K.\, cont\,(rg)\nabla r)\,])$

B.4.3 Operation reification condition: $RESET$

$\langle$ Proof of operation reification $(RESET)$. B.4.3 $\rangle \equiv$

$[\, in \;?\; \mathbf{void}\,;\overleftarrow{rg_D}\;?\; D_{st}$

$\vdash [\; hyp\_inv \;:\; D_{mod}\,.\, inv\,(\overleftarrow{rg_D})$

$\;;\; hyp\_pre \;:\; true$

$\;\vdash \langle\!|\; domain \;:=\; true\_is\_true \qquad \therefore D_{op}.\, RESET\,(in, \overleftarrow{rg_D}).\, pre$

$\quad ,\; result \;:=\; [\, out \;?\; \mathbf{void}\,; rg_D \;?\; D_{st}\,; rg := retr_D\,(rg_D)$

$\qquad\qquad \vdash [\; hyp\_post \;:\; rg_D = D.\, mk\,(\langle\rangle, \tau)$

$\qquad\qquad\; \vdash \langle$ Result case (RESET). B.4.3.1 $\rangle$

$\qquad\qquad\qquad \therefore K_{op}.\, RESET\,(in, retr_D(\overleftarrow{rg_D})).\, post\,(out, rg)$

$\qquad\qquad\;]$

$\qquad\qquad]$

$\quad |\!\rangle$

$\;]$

$]$

$\quad \therefore D_{op}.RESET \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\, RESET$

B.4.3.1.

$\langle$ Result case (RESET). B.4.3.1 $\rangle \equiv$

$[\ aux\ := \ retr\_rev_D\ (D.\ cont\ (rg_D), D.\ dep\ (rg_D))$

$\vdash refl$

$\quad \therefore rg = K.\ mk\ (atm(aux, \mathbf{dom}\ D\ .\ cont\ (rg_D)), D.\ dep\ (rg_D))$

$\setminus\ unfold(hyp\_post)$

$\quad \therefore rg = K.\ mk\ (atm(aux, \mathbf{dom}\ D\ .\ cont\ (rg_D)), D.\ dep\ (D.\ mk\ (\langle\rangle, \tau)))$

$\setminus\ unfold(def_s el_2)$

$\quad \therefore rg = K.\ mk\ (atm(aux, \mathbf{dom}\ D\ .\ cont\ (rg_D)), \tau)$

$\setminus\ unfold(hyp\_post)$

$\quad \therefore rg = K.\ mk\ (atm(aux, \mathbf{dom}\ D\ .\ cont\ (D.\ mk\ (\langle\rangle, \tau))), \tau)$

$\setminus\ unfold(def\_sel_1)$

$\quad \therefore rg = K.\ mk\ (atm(aux, \mathbf{dom}(\langle\rangle \therefore Rid \xrightarrow{m} \Delta(Line))), \tau)$

$\setminus\ unfold(domain.empty)$

$\quad \therefore rg = K.\ mk\ (atm(aux, \{\ \}), \tau)$

$\setminus\ unfold(abs\_to\_map.empty)$

$\quad \therefore rg = K.\ mk\ (\langle\rangle, \tau)$

$]$

B.4.4 Operation reification condition: *OPEN*

$\langle$ Proof of operation reification $(OPEN)$. B.4.4 $\rangle \equiv$

$[\ in\ ?\ File \otimes Rid\ ; \overleftarrow{rg_D}\ ?\ D_{st}$

$\vdash [\overleftarrow{rg} := \ retr_D\ (\overleftarrow{rg_D});\ f\ := sel_1(in);\ r\ := sel_2(in)$

$\quad hyp\_inv\ :\ D_{mod}\ .\ inv\ (\overleftarrow{rg_D});$

$\quad hyp\_pre\ : \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg}) = \{\ \} \wedge wff_F(f)$

$\quad \vdash \langle\!|\ domain\ := \langle$ Domain case $(OPEN)$. B.4.4.1 $\rangle$

$\quad ,\ result\ := \langle$ Result case $(OPEN)$. B.4.4.2 $\rangle$

$\quad |\!\rangle$

$\ ]$

$]$

$\quad \therefore D_{op}.OPEN \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\ OPEN$

B.4.4.1.

$\langle$ Domain case $(OPEN)$. B.4.4.1 $\rangle \equiv$

$[\ aux\ := [\ lhs\ := \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\quad \vdash refl$

$\qquad \therefore lhs = \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\quad \setminus unfold(get\_fd\_cont)$

$\qquad \therefore lhs = \mathbf{dom}\ atm\ (retr\_rev_D(D.\ cont\ (\overleftarrow{rg_D}), D.\ dep\ (\overleftarrow{rg_D})), \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D}))$

$\quad \setminus unfold(abs\_to\_map\_prop.dom)$

$\qquad \therefore lhs = \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})$

$\quad]$

$\vdash and\ .\ in\ (\langle\!|\ trans\ (sym(aux), pLeft(hyp\_pre))$

$\qquad ,\ pRight\ (hyp\_pre)$

$\qquad |\!\rangle$

$)$

$\quad \therefore \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D}) = \{\ \} \wedge wff_F(f)$

$\quad \therefore D_{op}.\ OPEN\ (in, \overleftarrow{rg_D}).\ pre$

$]$

B.4.4.2.

$\langle$ Result case $(OPEN)$. B.4.4.2 $\rangle \equiv$

$[\ out\ ?\ \mathbf{void}; rg_D\ ?\ D_{st}$

$\vdash [\ rg \quad := retr_D\ (rg_D)$

$\quad ;\ dep_{Do} \quad := node\ (r, \langle\rangle)$

$\quad ;\ cont_{Do} \quad := \langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u} \rangle\ \rangle$

$\quad ;\ rhs_{Do} \quad := D\ .\ mk\ (cont_{Do}, dep_{Do})$

$\quad ;\ retr_{Do} \quad := retr\_rev_D\ (D.\ cont\ (rg_D), D.\ dep\ (rg_D))$

$\quad ;\ retr_{map} \ := atm\ (retr_{Do}, \{r\})$

$\quad |\ hyp\_post \quad : \quad rg_D = rhs_{Do};$

$\quad |\ map\_lemma\ := \langle$ Retrieve map lemma. B.4.4.3 $\rangle \quad \therefore retr_{map} = \langle r \mapsto f \rangle$

$\quad \vdash \langle$ Abstract post-condition $(OPEN)$. B.4.4.4 $\rangle$

$\qquad \therefore rg = K.\ mk\ (\langle r \mapsto f \rangle, node(r, \langle\rangle))$

$\qquad \therefore K_{op}.\ OPEN\ (in, \overleftarrow{rg}).\ post\ (out, rg)$

$\quad]$

$]$

B.4.4.3.

$\langle$ Retrieve map lemma. B.4.4.3 $\rangle \equiv$

$refl$

$\quad \therefore retr_{map} = atm(retr_{Do}, \{r\})$

$\setminus unfold(abs\_to\_map\_prop.single)$

$\quad \therefore retr_{map} = \langle r \mapsto \langle\rangle \oplus_s (D.\ cont\ (rg_D\rangle * init\_path(r, D.\ dep\ (rg_D))))$

$\setminus unfold(hyp\_post)$

$\quad \therefore retr_{map} = \langle r \mapsto \langle\rangle \oplus_s (D.\ cont\ (rg_D\rangle * init\_path(r, D.\ dep\ (rhs_{Do}))))$

$\setminus unfold(def_s el_2)$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_s (D.\ cont\ (rg_D\rangle * init\_path(r, node(r, \langle\rangle))))$

$\setminus\ unfold(def\_init\_path.\ recur\ .\mathbf{1})$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_s (D.\ cont\ (rg_D\rangle * \langle\ r\ \rangle))$

$\setminus\ unfold(hyp\_post)$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_s (D.\ cont\ (rhs_{Do}\rangle * \langle\ r\ \rangle))$

$\setminus\ unfold(def\_sel_1)$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_s \langle r \mapsto \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\rangle\rangle * \langle\ r\ \rangle\ \rangle$

$\setminus\ unfold(map\_map\_prop.single)$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_s \langle\ \langle\ \langle\ 1, f, 0\ \rangle_{\Delta_u}\rangle\ \rangle\ \rangle$

$\setminus\ unfold(prop\_apply\_delta\_seq.single)$

$\therefore\ retr_{map} = \langle r \mapsto \langle\rangle \oplus_u \langle\ 1, f, 0\rangle\ \rangle_{\Delta_u}$

$\setminus\ unfold(prop\_unit.apply)$

$\therefore\ retr_{map} = \langle r \mapsto f\rangle$

B.4.4.4.

$\langle$ Abstract post-condition $(OPEN)$. B.4.4.4 $\rangle \equiv$

$refl$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \mathbf{dom}\ D\ .\ cont\ (rg_D)), D.\ dep\ (rg_D))$

$\setminus\ unfold(hyp\_post)$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \mathbf{dom}\ D\ .\ cont\ (rg_D)), D.\ dep\ (rhs_{Do}))$

$\setminus\ unfold(def_s el_2)$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \mathbf{dom}\ D\ .\ cont\ (rg_D)), dep_{Do})$

$\setminus\ unfold(hyp\_post)$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \mathbf{dom}\ D\ .\ cont\ (rhs_{Do})), dep_{Do})$

$\setminus\ unfold(def\_sel_1)$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \mathbf{dom} cont_{Do}), dep_{Do})$

$\setminus\ unfold(dom\_prop.single)$

$\therefore\ rg = K.\ mk\ (atm(retr_{Do}, \{r\}), dep_{Do})$

$\setminus\ unfold(map\_lemma)$

$\therefore\ rg = K.\ mk\ (\langle r \mapsto f\rangle, node(r, \langle\rangle))$

B.4.5 Operation reification condition: $CHECKOUT$

$\langle$ Proof of operation reification $(CHECKOUT)$. B.4.5 $\rangle \equiv$

$[\ r\ ?\ \ Rid\ ;\qquad \overleftarrow{rg_D}\ \ ?\ \ D_{st}$

$;\ \overleftarrow{rg} := retr_D\ (\overleftarrow{rg_D});\ retr_{Di} := retr\_rev_D\ (D.\ cont\ (\overleftarrow{rg_D}), D.\ dep\ (\overleftarrow{rg_D}))$

$hyp\_inv \quad : \quad D_{mod}\ .\ inv\ (\overleftarrow{rg_D});$

$hyp\_pre \quad : \quad r \in \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg});$

$dom\_lemma := \langle$ Domain lemma $(CHECKOUT)$. B.4.5.2 $\rangle\ ;$

$\therefore \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg}) = \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})$

$\vdash\ hyp\_pre\_aux := hyp\_pre$

$\therefore r \in \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\setminus\ subst(dom\_lemma)$

$\therefore r \in \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})$

$\langle\!|\ domain := hyp\_pre\_aux$

$\therefore D_{op}.\ CHECKOUT\ (r, \overleftarrow{rg_D}).\ pre$

$,\ result := \langle$ Result case $(CHECKOUT)$. B.4.5.1 $\rangle$

$|\!\rangle$

$]$

$\therefore D_{op}.CHECKOUT \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\ CHECKOUT$

B.4.5.1.

$\langle$ Result case $(CHECKOUT)$. B.4.5.1 $\rangle \equiv$

$[\ f\ ?\ File\ ;\ rg_D\ ?\ D_{st}\ ;\ rg := retr_D\ (rg_D)$

$\vdash [\ hyp\_post\ : \quad f = retr_{Di}(r) \wedge rg_D = \overleftarrow{rg_D}$

$;\ goal \quad := f = K.\ cont\ (\overleftarrow{rg}) \nabla\ r \wedge rg = \overleftarrow{rg}$

$\vdash \langle\!|\ refl$

$\therefore K.\ cont\ (\overleftarrow{rg}) \nabla r = K.\ cont\ (retr_D(\overleftarrow{rg_D})) \nabla\ r$

$\setminus\ unfold(get\_fd\_cont)$

$\therefore K.\ cont\ (\overleftarrow{rg}) \nabla r = atm(retr_{Di}, \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})) \nabla\ r$

$\setminus\ unfold(abs\_to\_map\_prop.\ apply\ (F := retr_{Di}, hyp\_pre\_aux))$

$\therefore K.\ cont\ (\overleftarrow{rg}) \nabla r = retr_{Di}(r)$

$\setminus\ fold(pLeft(hyp\_post))$

$\therefore K.\ cont\ (\overleftarrow{rg}) \nabla r = f$

$\setminus\ sym$

$\therefore f = K.\ cont\ (\overleftarrow{rg}) \nabla\ r$

$,\ refl$

$\therefore rg = retr_D(rg_D)$

$\setminus\ unfold(pRight(hyp\_post))$

$\therefore rg = retr_D(\overleftarrow{rg_D})$

$|\!\rangle$

$\setminus\ and.\ in$

$\therefore K_{op}.\ CHECKOUT\ (r, \overleftarrow{rg}).\ post\ (f, rg)$

$]$

$]$

B.4.5.2.

$\langle$ Domain lemma $(CHECKOUT)$. B.4.5.2 $\rangle \equiv$

$[\ lhs\ := \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\vdash refl$

$\quad \therefore lhs = \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$\quad \setminus unfold(get\_fd\_cont)$

$\quad \therefore lhs = \mathbf{dom}\ atm\ (retr_{Di}, \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D}))$

$\quad \setminus unfold(abs\_to\_map\_prop.dom)$

$\quad \therefore lhs = \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})$

$]$

B.4.6 Operation verification condition: *CHECKIN*

$\langle$ Proof of operation reification $(CHECKIN)$. B.4.6 $\rangle \equiv$

$[\ in\ ?\ File \otimes (Rid \otimes Rid); \overleftarrow{rg_D}\ ?\ D_{st}$

$\vdash [\ \overleftarrow{rg}\ := retr_D\ (\overleftarrow{rg_D});\quad f\ := sel_1(in)$

$\quad ;\ or\ := sel_1(sel_2(in));\ nr\ := sel_2(sel_2(in))$

$\quad \vdash [\ hyp\_inv\ :\ D_{mod}\ .\ inv\ (\overleftarrow{rg_D})$

$\quad\quad ;\ hyp\_pre\ :\ K_{op}\ .\ CHECKIN\ (in, \overleftarrow{rg}).\ pre$

$\quad\quad ; \langle$ Side deductions $(CHECKIN)$. B.4.6.1 $\rangle$

$\quad\quad \vdash \langle\!|\ domain\ := \langle$ Domain case $(CHECKIN)$. B.4.6.2 $\rangle$

$\quad\quad\quad ,\ result\ := \langle$ Result case $(CHECKIN)$. B.4.6.3 $\rangle$

$\quad\quad\quad |\!\rangle$

$\quad\quad]$

$\quad]$

$]$

$\quad \therefore D_{op}.CHECKIN \sqsubseteq^{op}_{D_{mod}.inv, retr_D} K_{op}.\ CHECKIN$

B.4.6.1.

$\langle$ Side deductions $(CHECKIN)$. B.4.6.1 $\rangle \equiv$

$[\![\ inv\_unique\ := hyp\_inv \setminus pLeft \setminus pLeft \setminus pRight$

$;\ inv\_dom\ := hyp\_inv \setminus pLeft \setminus pLeft \setminus pLeft$

$;\ pre\_rev\ := hyp\_pre \setminus pLeft \setminus pLeft$

$;\ pre\_old\ := pre\_rev \setminus pLeft$

$\quad\quad \therefore or \in \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$;\ pre\_new\ := pre\_rev \setminus pRight$

$\quad\quad \therefore nr \notin \mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg})$

$;\ pre\_card\ := hyp\_pre \setminus pLeft \setminus pRight$

$\quad\quad \therefore RevMax$ ¿ $\mathbf{card}(\mathbf{dom}\ K\ .\ cont\ (\overleftarrow{rg}))$

$;\ pre\_wff\ := hyp\_pre \setminus pRight$

$; \; dom\_equal \quad := [\; lhs \; := \mathbf{dom}\, K \,\mathbf{.}\, cont\,(\overleftarrow{rg})$

$\qquad \vdash refl$

$\qquad\qquad \therefore lhs = \mathbf{dom}\, K \,\mathbf{.}\, cont\,(\overleftarrow{rg})$

$\qquad\quad \setminus unfold(get\_fd\_cont)$

$\qquad\qquad \therefore lhs = \mathbf{dom}\; atm\; (retr\_rev_D(D\mathbf{.}\, cont\,(\overleftarrow{rg_D}), D\mathbf{.}\, dep\,(\overleftarrow{rg_D})), \mathbf{dom}\, D \,\mathbf{.}\, cont\,(\overleftarrow{rg_D}))$

$\qquad\quad \setminus unfold(abs\_to\_map\_prop\mathbf{.}dom)$

$\qquad\qquad \therefore \mathbf{dom}\, K \,\mathbf{.}\, cont\,(\overleftarrow{rg}) = \mathbf{dom}\, D \,\mathbf{.}\, cont\,(\overleftarrow{rg_D})$

$\qquad]$

$; \; pre\_old_D \quad := \; pre\_old$

$\qquad \setminus subst(dom\_equal)$

$\qquad\quad \therefore or \in \mathbf{dom}\, D \,\mathbf{.}\, cont\,(\overleftarrow{rg_D})$

$; \; pre\_new_D \quad := \; pre\_new$

$\qquad \setminus subst(dom\_equal)$

$\qquad\quad \therefore nr \notin \mathbf{dom}\, D \,\mathbf{.}\, cont\,(\overleftarrow{rg_D})$

$; \; pre\_card_D \quad := \; pre\_card$

$\qquad \setminus subst(dom\_equal)$

$\qquad\quad \therefore RevMax \text{ ¿ } \mathbf{card}(\mathbf{dom}\, D \,\mathbf{.}\, cont\,(\overleftarrow{rg_D}))$

$; \; inv\_old_D \quad := \; pre\_old_D$

$\qquad \setminus subst(inv\_dom)$

$\qquad\quad \therefore or \in info(D\mathbf{.}\, dep\,(\overleftarrow{rg_D}))$

$; \; inv\_new_D \quad := \; pre\_new_D$

$\qquad \setminus subst(inv\_dom)$

$\qquad\quad \therefore nr \notin info(D\mathbf{.}\, dep\,(\overleftarrow{rg_D}))$

$]\!]$

B.4.6.2.

$\langle$ Domain case $(CHECKIN)$. B.4.6.2 $\rangle \equiv$

$\{\!|\; pre\_old_D\;,\; pre\_new_D\;,\; pre\_card_D\;,\; pre\_wff\; |\!\}$

$\setminus and4$

$\quad \therefore D_{op}\mathbf{.}\; CHECKIN\;(in, \overleftarrow{rg_D})\mathbf{.}\; pre$

B.4.6.3.

$\langle$ Result case $(CHECKIN)$. B.4.6.3 $\rangle \equiv$

$[\ out\ ?\ \mathbf{void}\,;\, rg_D\ ?\ D_{st}\ ;\, rg := retr_D\,(rg_D)$

$$\vdash \left|\begin{array}{lcl} retr_{Di} & := & retr\_rev_D\,(D.\,cont\,(\overleftarrow{rg_D}), D.\,dep\,(\overleftarrow{rg_D}));\\ retr_{Do} & := & retr\_rev_D\,(D.\,cont\,(rg_D), D.\,dep\,(rg_D));\\ del & := & diff\,(retr_{Di}(or), f);\\ cont_{Do} & := & (nr \mapsto del) \odot D.\,cont\,(\overleftarrow{rg_D});\\ dep_{Do} & := & insert\,(or, nr, D.\,dep\,(\overleftarrow{rg_D}));\\ retr_{DoR} & := & retr\_rev_D\,(cont_{Do}, dep_{Do}, nr);\\ hyp\_post & : & rg_D = D.\,mk\,(cont_{Do}, dep_{Do}) \end{array}\right.$$

[⟨ Result lemmas ($CHECKIN$:result). B.4.6.4⟩

⊢⟨ Body of the result case. B.4.6.5⟩

$\quad \therefore K_{op}.\ CHECKIN\,(in, \overleftarrow{rg}).\,post\,(out, rg)$

]

$]$

B.4.6.4.

⟨ Result lemmas ($CHECKIN$:result). B.4.6.4⟩ ≡

$[\![\ post_cont\ := $ ⟨ Result lemma I. B.4.6.9⟩

$\qquad \therefore D.\,cont\,(rg_D) = cont_{Do}$

$;\ post\_dep\ :=$ ⟨ Result lemma II. B.4.6.10⟩

$\qquad \therefore D.\,dep\,(rg_D) = dep_{Do}$

$;\ lemma_{III}\ :=$ ⟨ Result lemma III. B.4.6.11⟩

$\qquad \therefore nr \notin \text{elems}\ init\_path\,(or, D.\,dep\,(\overleftarrow{rg_D}))$

$;\ lemma_{IV}\ :=$ ⟨ Proof of lemma IV. B.4.6.6⟩

$\qquad \therefore retr_{DoR} = f$

$;\ lemma_{V}\ :=$ ⟨ Proof of lemma V. B.4.6.7⟩

$;\ lemma_{VI}\ :=$ ⟨ Proof of lemma VI. B.4.6.8⟩

$\qquad \therefore atm(retr_{Do}, \mathbf{dom}\,D\,.\,cont\,(rg_D)) = (nr \mapsto f) \odot K.\,cont\,(\overleftarrow{rg})$

$]\!]$

B.4.6.5.

⟨ Body of the result case. B.4.6.5⟩ ≡

$refl$

$\quad \therefore rg = K.\,mk\,(atm(retr_{Do}, \mathbf{dom}\,D\,.\,cont\,(rg_D)), D.\,dep\,(rg_D))$

$\setminus\ unfold(lemma_{VI})$

$\quad \therefore rg = K.\,mk\,((nr \mapsto f) \odot K.\,cont\,(\overleftarrow{rg}), D.\,dep\,(rg_D))$

$\setminus\ unfold(post\_dep)$

$\quad \therefore rg = K.\,mk\,((nr \mapsto f) \odot K.\,cont\,(\overleftarrow{rg}), insert(or, nr, D.\,dep\,(\overleftarrow{rg_D})))$

$\setminus\ fold(get\_fd\_dep)$

$\quad \therefore rg = K.\,mk\,((nr \mapsto f) \odot K.\,cont\,(\overleftarrow{rg}), insert(or, nr, K.\,dep\,(\overleftarrow{rg})))$

$\quad \therefore K_{op}.\ CHECKIN\,(in, \overleftarrow{rg}).\,post\,(out, rg)$

B.4.6.6.

$\langle$ Proof of lemma IV. B.4.6.6 $\rangle \equiv$

refl

$\therefore\ retr_{DoR} = \langle\rangle \oplus_s (cont_{Do} * init\_path(nr, insert(or, nr, D.\ dep\ (\overleftarrow{rg_D}))))$

$\setminus\ unfold(init\_path\_prop(inv\_unique, inv\_old_D, inv\_new_D).last)$

$\therefore\ retr_{DoR} = \langle\rangle \oplus_s (cont_{Do} * (init\_path(or, D.\ dep\ (\overleftarrow{rg_D})) +\!\!+ \langle\ nr\ \rangle))$

$\setminus\ unfold(map\_map\_prop.join)$

$\therefore\ retr_{DoR} = \langle\rangle \oplus_s ((cont_{Do} * init\_path(or, D.\ dep\ (\overleftarrow{rg_D}))) +\!\!+ (cont_{Do} * \langle\ nr\ \rangle))$

$\setminus\ unfold(map\_map\_prop.single)$

$\therefore\ retr_{DoR} = \langle\rangle \oplus_s ((cont_{Do} * init\_path(or, D.\ dep\ (\overleftarrow{rg_D}))) +\!\!+ \langle\ del\ \rangle)$

$\setminus\ unfold(F := [\, h : seq\ (\Delta(Line)) \vdash \langle\rangle \oplus_s (h +\!\!+ \langle\ del\ \rangle)\,]$

$,\ map\_map\_prop.\ reduce\ (lemma_{III}))$

$\therefore\ retr_{DoR} = \langle\rangle \oplus_s ((D.\ cont\ (\overleftarrow{rg_D}) * init\_path(or, D.\ dep\ (\overleftarrow{rg_D}))) +\!\!+ \langle\ del\ \rangle)$

$\setminus\ unfold(prop\_apply\_delta\_seq2)$

$\therefore\ retr_{DoR} = retr_{Di}(or) \oplus del$

$\setminus\ unfold(prop\_diff$

$\setminus\ pRight)$

$\therefore\ retr_{DoR} = f$

B.4.6.7.

$\langle$ Proof of lemma V. B.4.6.7 $\rangle \equiv$

$[\ r\ ?\ Rid$

$\vdash [\ hyp \quad : \quad r \in \mathbf{dom}\ D\ .\ cont\ (\overleftarrow{rg_D})$

$;\ hyp\_aux\ :=\ hyp$

$\qquad \setminus\ subst(inv\_dom)$

$\qquad \therefore\ r \in info(D.\ dep\ (\overleftarrow{rg_D}))$

$\vdash retr\_lemma\ (inv\_unique, hyp\_aux, inv\_old_D, inv\_new_D)$

$\therefore\ retr\_rev_D(cont_{Do}, dep_{Do}, r) = retr\_rev_D(D.\ cont\ (\overleftarrow{rg_D}), D.\ dep\ (\overleftarrow{rg_D}), r)$

$]$

$]$

B.4.6.8.

$\langle$ Proof of lemma VI. B.4.6.8 $\rangle \equiv$

$[\,lhs\ :=\ atm\ (retr_{Do}, \mathbf{dom}\ D\ \mathbf{.}\ cont\ (rg_D))$

$\vdash refl$

$\quad \therefore lhs = atm(retr\_rev_D(D\mathbf{.}\ cont\ (rg_D), D\mathbf{.}\ dep\ (rg_D)), \mathbf{dom}\ D\ \mathbf{.}\ cont\ (rg_D))$

$\setminus\ unfold(post\_dep)$

$\quad \therefore lhs = atm(retr\_rev_D(D\mathbf{.}\ cont\ (rg_D), dep_{Do}), \mathbf{dom}\ D\ \mathbf{.}\ cont\ (rg_D))$

$\setminus\ unfold(post\_cont)$

$\quad \therefore lhs = atm(retr\_rev_D(D\mathbf{.}\ cont\ (rg_D), dep_{Do}), \mathbf{dom} cont_{Do})$

$\setminus\ unfold(F := [\,h\ : Rid \xrightarrow{m} \Delta(Line)$ $\qquad , post\_cont)$

$\qquad \vdash atm\ (retr\_rev_D(h, insert(or, nr, D\mathbf{.}\ dep\ (\overleftarrow{rg_D}))), \mathbf{dom} cont_{Do})$

$\qquad]$

$\quad \therefore lhs = atm(retr\_rev_D(cont_{Do}, dep_{Do}), \mathbf{dom} cont_{Do})$

$\setminus\ unfold(domain\mathbf{.}recur)$

$\quad \therefore lhs = atm(retr\_rev_D(cont_{Do}, dep_{Do}), nr \odot \mathbf{dom}\ D\ \mathbf{.}\ cont\ (\overleftarrow{rg_D}))$

$\setminus\ unfold(abs\_to\_map\mathbf{.}recur)$

$\quad \therefore lhs = (nr \mapsto retr_{DoR}) \odot atm(retr\_rev_D(cont_{Do}, insert(or, nr, D\mathbf{.}\ dep\ (\overleftarrow{rg_D}))), \mathbf{dom}\ D\ \mathbf{.}\ cont\ (\overleftarrow{rg_D}))$

$\setminus\ unfold(abs\_to\_map\_prop\mathbf{.}\ exten\ (lemma_V))$

$\quad \therefore lhs = (nr \mapsto retr_{DoR}) \odot atm(retr_{Di}, \mathbf{dom}\ D\ \mathbf{.}\ cont\ (\overleftarrow{rg_D}))$

$\setminus\ fold(get\_fd\_cont)$

$\quad \therefore lhs = (nr \mapsto retr_{DoR}) \odot K\mathbf{.}\ cont\ (\overleftarrow{rg})$

$\setminus\ unfold(lemma_{IV})$

$\quad \therefore lhs = (nr \mapsto f) \odot K\mathbf{.}\ cont\ (\overleftarrow{rg})$

$]$

B.4.6.9.

$\langle$ Result lemma I. B.4.6.9 $\rangle \equiv$

$refl$

$\quad \therefore D\mathbf{.}\ cont\ (rg_D) = D\mathbf{.}\ cont\ (rg_D)$

$\setminus\ unfold(hyp\_post)$

$\quad \therefore D\mathbf{.}\ cont\ (rg_D) = D\mathbf{.}\ cont\ (D\mathbf{.}\ mk\ (cont_{Do}, dep_{Do}))$

$\setminus\ unfold(def\_sel_1)$

$\quad \therefore D\mathbf{.}\ cont\ (rg_D) = cont_{Do}$

B.4.6.10.

$\langle$ Result lemma II. B.4.6.10 $\rangle \equiv$

$refl$

$\quad \therefore D\mathbf{.}\ dep\ (rg_D) = D\mathbf{.}\ dep\ (rg_D)$

$\setminus\ unfold(hyp\_post)$

$\quad \therefore D\mathbf{.}\ dep\ (rg_D) = D\mathbf{.}\ dep\ (D\mathbf{.}\ mk\ (cont_{Do}, dep_{Do}))$

$\setminus\ unfold(def_s el_2)$

$\quad \therefore D\mathbf{.}\ dep\ (rg_D) = dep_{Do}$

B.4.6.11.

$\langle$ Result lemma III. B.4.6.11 $\rangle \equiv$

$[\, hyp \;: nr \in \text{elems}\; init\_path\; (or, D\textbf{.}\; dep\; (\overleftarrow{rg_D}))$

$\vdash hyp$

$\quad \setminus\; subset\_prop\textbf{.}\; weaken\; (path\_incl)$

$\qquad \therefore\; nr \in info(D\textbf{.}\; dep\; (\overleftarrow{rg_D}))$

$\quad \setminus\; not\textbf{.}\; out\; (inv\_new_D)$

$\qquad \therefore\; false$

$]$

$\setminus\; not\textbf{.}\; in$

$\quad \therefore\; nr \notin \text{elems}\; init\_path\; (or, D\textbf{.}\; dep\; (\overleftarrow{rg_D}))$

B.5 Extension by user-held locks: proofs

B.5.1 Validity lemmas

B.5.1.1.

$\langle$ Validity lemmas (locks). B.5.1.1 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\; inv_{KMAP} & := [\, rg \;:\; L_{st} & \\
 & \quad \vdash K_{mod}\textbf{.}\; inv\; (sel_1(rg)) & \\
 & \qquad \wedge\; MAP_{mod}(Rid, Uid, RevMax)\textbf{.}\; inv\; (sel_2(rg)) & \\
 & \quad] & \\
;\; inv_E & := [\, rg : L_{st} \;\vdash \textbf{dom}\; L\textbf{.}\; locks\; (rg) \subseteq \textbf{dom}\; L\textbf{.}\; cont\; (rg)\,] & \\
;\; inv_{KMAPE} & := [\, rg \;:\; L_{st} & \\
 & \quad \vdash inv_{KMAP}\; (rg) \wedge inv_E(rg) & \\
 & \quad] & \\
;\; inv_{LMAP} & := [\, rg \;:\; L_{st} & \\
 & \quad \vdash L_{mod}\textbf{.}\; inv\; (rg) & \\
 & \qquad \wedge\; MAP_{mod}(Rid, Uid, RevMax)\textbf{.}\; inv\; (sel_2(rg)) & \\
 & \quad] &
\end{array}
$$

$; \ inv\_eqv1 \quad := [\ rg \ : \ L_{st}$

$\vdash \langle\!|\ [\ hyp \ : \ inv_{KMAPE}\ (rg)$

$\vdash and\ .\ in\ (\langle\!|\ pLeft\ (pLeft(hyp)),\ pRight\ (hyp) |\!\rangle)$

$\therefore L_{mod}.\ inv\ (rg)$

$]$

$,[\ hyp \ : \quad L_{mod}\ .\ inv\ (rg)$

$;\ a \quad := pRight\ (hyp)$

$;\ b \quad := pLeft\ (hyp)$

$;\ new\ := card\_prop\ (pRight(pLeft(b)), a)$

$\vdash \langle\!|\ b\ ,\ new\ ,\ a\ |\!\rangle \qquad \backslash\ and3$

$\therefore inv_{KMAPE}(rg)$

$]$

$|\!\rangle$

$\backslash\ equiv.\ in$

$\therefore inv_{KMAPE}(rg) \Leftrightarrow L_{mod}.\ inv\ (rg)$

$]$

$; \ inv\_eqv2 \quad := [\ rg \ : \ L_{st}$

$\vdash \langle\!|\ [\ hyp : inv_{LMAP}\ (rg) \vdash pLeft(hyp) \therefore L_{mod}.\ inv\ (rg)\ ]$

$,[\ hyp \ : \quad L_{mod}\ .\ inv\ (rg)$

$;\ new\ := card\_prop\ (pRight(pLeft(pLeft(hyp))), pRight(hyp))$

$\vdash \langle\!|\ hyp\ ,\ new\ |\!\rangle \qquad \backslash\ and.\ in$

$\therefore inv_{LMAP}(rg)$

$]$

$|\!\rangle$

$\backslash\ equiv.\ in$

$\therefore inv_{LMAP}(rg) \Leftrightarrow L_{mod}.\ inv\ (rg)$

$]$

$]\!]$

B.5.2 Reset operation

$\langle$ Proof of $val\_op\ (L_{op}.RESET, L_{mod}.inv)$ B.5.2 $\rangle \equiv$

$val\_oconj\ (RESET_{val}, MAP\_CREATE\_val)$

$\therefore val\_op(L_{op}.RESET, inv_{KMAP})$

$\backslash\ val\_xtnd\_inv(inv_1 := inv_E)(\langle$ Extension lemma for $RESET$ B.5.2.1 $\rangle).sel_2$

$\therefore val\_op(L_{op}.RESET, inv_{KMAPE})$

$\backslash\ val\_subst\_inv(inv_2 := L_{mod}.inv, inv\_eqv1)$

$\therefore val\_op(L_{op}.RESET, L_{mod}.inv)$

B.5.2.1.

$\langle$ Extension lemma for $RESET$ B.5.2.1 $\rangle \equiv$

$[i, o : \mathbf{void}\, ; \overleftarrow{rg}, rg \,?\ L_{st}$

$\vdash [\ hyp\_inv \quad : \quad inv_E\ (\overleftarrow{rg})$

$\quad ;\ hyp\_pre \quad : \quad L_{op} \,.\, RESET\ (i, \overleftarrow{rg}).\ pre$

$\quad ;\ hyp\_post \quad : \quad L_{op} \,.\, RESET\ (i, \overleftarrow{rg}).\ post\ (o, rg)$

$\quad ;\ eval\_post\ := \langle$ Derivation of the evaluated postcondition of $RESET$ B.5.2.2 $\rangle$

$\qquad \therefore L.\ K\ (rg) = K.\ mk\ (\langle\rangle, \tau) \wedge L.\ locks\ (rg) = \langle\rangle$

$\quad \vdash subset\_empty$

$\qquad \therefore \{\ \} \subseteq \mathbf{dom}\, L \,.\, cont\ (rg)$

$\qquad \setminus\ rsubst(domain.empty)$

$\qquad \therefore \mathbf{dom}\langle\rangle \subseteq \mathbf{dom}\, L \,.\, cont\ (rg)$

$\qquad \setminus\ rsubst(pRight(eval\_post))$

$\qquad \therefore \mathbf{dom}\, L \,.\, locks\ (rg) \subseteq \mathbf{dom}\, L \,.\, cont\ (rg)$

$\quad]$

$]$

B.5.2.2.

$\langle$ Derivation of the evaluated postcondition of $RESET$ B.5.2.2 $\rangle \equiv$
$equiv \,.\, out\ (proj\_opeq(L\_RESET\_eval\_pr).post).\ down\ (hyp\_post)$

B.5.3 Open operation

$\langle$ Proof of $val\_op\ (L_{op}.OPEN, L_{mod}.inv)$ B.5.3 $\rangle \equiv$
$val\_oconj\ (OPEN_{val}, MAP\_CREATE\_val$

$\setminus\ val\_init\_in)$

$\quad \therefore val\_op(L_{op}.OPEN, inv_{KMAP})$

$\setminus\ val\_xtnd\_inv(inv_1 := inv_E)(\langle$ Extension lemma for $OPEN$ B.5.3.1 $\rangle).sel_2$

$\quad \therefore val\_op(L_{op}.OPEN, inv_{KMAPE})$

$\setminus\ val\_subst\_inv(inv\_eqv1)$

$\quad \therefore val\_op(L_{op}.OPEN, L_{mod}.inv)$

B.5.3.1.

$\langle$ Extension lemma for $OPEN$ B.5.3.1 $\rangle \equiv$

$[\ i\ : (File \otimes Rid);\ o\ : \mathbf{void}\,; \overleftarrow{rg}, rg\ ?\ L_{st}\ ;\ f\ := sel_1(i);\ newr\ := sel_2(i)$

$\vdash[\ hyp\_inv\ \ :\ \ inv_E\ (\overleftarrow{rg})$

$\ ;\ hyp\_pre\ \ :\ \ L_{op}\ .\ OPEN\ (i, \overleftarrow{rg}).\ pre$

$\ ;\ hyp\_post\ \ :\ \ L_{op}\ .\ OPEN\ (i, \overleftarrow{rg}).\ post\ (o, rg)$

$\ ;\ eval\_post\ := \langle$ Derivation of the evaluated postcondition of $OPEN$ B.5.3.2$\rangle$

$\therefore L.\ K\ (rg) = K.\ mk\ (\langle newr \mapsto f\rangle, node(newr, \langle\rangle))$

$\wedge\ L.\ locks\ (rg) = \langle\rangle$

$\ ;\ co\_post\ := \langle$ Derivation of the post-content $(OPEN)$. B.5.3.3$\rangle$

$\therefore L.\ cont\ (rg) = \langle newr \mapsto f\rangle$

$\vdash subset\_empty$

$\therefore \{\,\} \subseteq \mathbf{dom}\langle newr \mapsto f\rangle$

$\setminus\ rsubst(co\_post)$

$\therefore \{\,\} \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

$\setminus\ rsubst(domain.empty)$

$\therefore \mathbf{dom}\langle\rangle \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

$\setminus\ rsubst(pRight(eval\_post))$

$\therefore \mathbf{dom}\ L\ .\ locks\ (rg) \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

]

]

B.5.3.2.

$\langle$ Derivation of the evaluated postcondition of $OPEN$ B.5.3.2$\rangle \equiv$

$equiv\ .\ out\ (proj\_opeq(L\_OPEN\_eval\_pr).post).\ down\ (hyp\_post)$

B.5.3.3.

$\langle$ Derivation of the post-content $(OPEN)$. B.5.3.3$\rangle \equiv$

$refl$

$\therefore L.\ cont\ (rg) = K.\ cont\ (L.\ K\ (rg))$

$\setminus\ unfold(pLeft(eval\_post))$

$\therefore L.\ cont\ (rg) = K.\ cont\ (K.\ mk\ (\langle newr \mapsto f\rangle, node(newr, \langle\rangle)))$

$\setminus\ unfold(def\_sel_1)$

$\therefore L.\ cont\ (rg) = \langle newr \mapsto f\rangle$

B.5.4 Set lock operation

$\langle$ Proof of $val\_op\ (L_{op}.SET, L_{mod}.inv)$ B.5.4$\rangle \equiv$

MAP_INSERT_val

$\setminus\ val\_xtnd\_st.sel_1$

$\setminus\ val\_strpre(P := [\ in : (Rid \otimes Uid);\ rg :\ L_{st}\ \vdash sel_1(in) \notin \mathbf{dom}\ L\ .\ cont\ (rg)\,])$

$\setminus\ val\_xtnd\_inv(inv_1 := L_{mod}.inv)(\langle$ Extension: SET B.5.4.1$\rangle).sel_1$

$\therefore val\_op(L_{op}.SET, inv_{LMAP})$

$\setminus\ val\_subst\_inv(inv\_eqv2)$

$\therefore val\_op(L_{op}.SET, L_{mod}.inv)$

B.5.4.1.

$\langle$ Extension: SET B.5.4.1 $\rangle \equiv$

$[\, i\, : (Rid \otimes Uid);\ o\ :\ \mathbf{void}\, ; \overleftarrow{rg}, rg\ ?\ L_{st}\ ;\ r\ := sel_1(i);\ u\ := sel_2(i)$

$\vdash [\ hyp\_inv\ \ :\ \ L_{mod}\,.\, inv\ (\overleftarrow{rg})$

$\ ;\ hyp\_pre\ \ :\ \ L_{op}\,.\, SET\ (i, \overleftarrow{rg}).\, pre$

$\ ;\ hyp\_post\ :\ \ L_{op}\,.\, SET\ (i, \overleftarrow{rg}).\, post\ (o, rg)$

$\ ;\ eval\_pre\ :=$ $\langle$ Derivation of the evaluated precondition of SET B.5.4.4 $\rangle$

$\quad \therefore r \notin \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg})$

$\quad \wedge\, \mathbf{card}(\mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}))$ ¡ $RevMax$

$\quad \wedge\, r \in \mathbf{dom}\, L\,.\, cont\ (\overleftarrow{rg})$

$\ ;\ eval\_post\ :=$ $\langle$ Derivation of the evaluated postcondition of SET B.5.4.5 $\rangle$

$\quad \therefore L.\, K\ (rg) = L.\, K\ (\overleftarrow{rg}) \wedge L.\, locks\ (rg) = (r \mapsto u) \odot L.\, locks\ (\overleftarrow{rg})$

$\ ;\ co\_post\ \ := refl$

$\quad \therefore L.\, cont\ (rg) = K.\, cont\ (L.\, K\ (rg))$

$\quad \setminus\ unfold(pLeft(eval\_post))$

$\quad \therefore L.\, cont\ (rg) = L.\, cont\ (\overleftarrow{rg})$

$\ \vdash \langle\!|$ $\langle$ Old part of the invariant (SET). B.5.4.2 $\rangle$

$\qquad \therefore K_{mod}.\, inv\ (L.\, K\ (rg))$

$\quad ,$ $\langle$ New part of the invariant (SET). B.5.4.3 $\rangle$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\ (rg) \subseteq \mathbf{dom}\, L\,.\, cont\ (rg)$

$\quad |\!\rangle$

$\quad \setminus\ and.\, in$

$\qquad \therefore L_{mod}.\, inv\ (rg)$

$\ ]$

$]$

B.5.4.2.

$\langle$ Old part of the invariant (SET). B.5.4.2 $\rangle \equiv$

$pLeft\ (hyp\_inv)$

$\quad \therefore K_{mod}.\, inv\ (L.\, K\ (\overleftarrow{rg}))$

$\setminus\ rsubst(pLeft(eval\_post))$

$\quad \therefore K_{mod}.\, inv\ (L.\, K\ (rg))$

B.5.4.3.

$\langle$ New part of the invariant (SET). B.5.4.3 $\rangle \equiv$

$pRight\ (hyp\_inv)$

$\quad \therefore \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\ (\overleftarrow{rg})$

$\setminus\ subset\_prop.\, extend\ (new := pRight(eval\_pre))$

$\quad \therefore (r \odot \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg})) \subseteq \mathbf{dom}\, L\,.\, cont\ (\overleftarrow{rg})$

$\setminus\ rsubst(co\_post)$

$\quad \therefore (r \odot \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg})) \subseteq \mathbf{dom}\, L\,.\, cont\ (rg)$

$\setminus$ $rsubst(domain.recur)$

$\quad \therefore \mathbf{dom}((r \mapsto u) \odot L.\, locks\, (\overleftarrow{rg})) \subseteq \mathbf{dom}\, L\, .\, cont\, (rg)$

$\setminus$ $rsubst(pRight(eval\_post))$

$\quad \therefore \mathbf{dom}\, L\, .\, locks\, (rg) \subseteq \mathbf{dom}\, L\, .\, cont\, (rg)$

B.5.4.4.

$\langle$ Derivation of the evaluated precondition of SET B.5.4.4 $\rangle \equiv$

$equiv\, .\, out\, (proj\_opeq(L\_SET\_eval).pre).\, down\, (hyp\_pre)$

B.5.4.5.

$\langle$ Derivation of the evaluated postcondition of SET B.5.4.5 $\rangle \equiv$

$equiv\, .\, out\, (proj\_opeq(L\_SET\_eval).post).\, down\, (hyp\_post)$

B.5.5 Release Lck operation

$\langle$ Proof of $val\_op\ (L_{op}.FREE, L_{mod}.inv)$ B.5.5 $\rangle \equiv$

$val\_xtnd\_st\, .sel_1(MAP\_DELETE\_val)$

$\setminus$ $val\_xtnd\_inv(inv_1 := L_{mod}.inv)(\langle$ Extension: $FREE$ B.5.5.1 $\rangle).sel_1$

$\quad \therefore val\_op(L_{op}.FREE, inv_{LMAP})$

$\setminus$ $val\_subst\_inv(inv\_eqv2)$

$\quad \therefore val\_op(L_{op}.FREE, L_{mod}.inv)$

B.5.5.1.

$\langle$ Extension: $FREE$ B.5.5.1 $\rangle \equiv$

$[\ r\ :\ Rid\ ;\ o\ :\ \mathbf{void}\ ; \overleftarrow{rg}, rg\ ?\ L_{st}$

$\vdash [\ hyp\_inv \quad : \quad L_{mod}\ .\ inv\ (\overleftarrow{rg})$

$\quad ;\ hyp\_pre \quad : \quad L_{op}\ .\ FREE\ (r, \overleftarrow{rg}).\ pre$

$\quad ;\ hyp\_post \quad : \quad L_{op}\ .\ FREE\ (r, \overleftarrow{rg}).\ post\ (o, rg)$

$\quad ;\ eval\_post\ := \langle$ Derivation of the evaluated postcondition of $FREE$ B.5.5.5 $\rangle$

$\qquad \therefore L.\ K\ (rg) = L.\ K\ (\overleftarrow{rg})$

$\qquad \wedge\ L.\ locks\ (rg) = [\ r1 : Rid\ ; u :\ Uid\ \vdash \neg r1 = r\ ] \rhd L.\ locks\ (\overleftarrow{rg})$

$\quad ;\ co\_post \quad := refl$

$\qquad \therefore L.\ cont\ (rg) = K.\ cont\ (L.\ K\ (rg))$

$\qquad \setminus\ unfold(pLeft(eval\_post))$

$\qquad \therefore L.\ cont\ (rg) = L.\ cont\ (\overleftarrow{rg})$

$\quad ;\ aux \quad := \langle$ Auxiliary lemma. B.5.5.4 $\rangle$

$\qquad \therefore \mathbf{dom}\ L\ .\ locks\ (rg) \subseteq \mathbf{dom}\ L\ .\ locks\ (\overleftarrow{rg})$

$\quad \vdash \langle\!|\ \langle$ Old part of the invariant $(FREE)$. B.5.5.2 $\rangle$

$\qquad \therefore K_{mod}.\ inv\ (L.\ K\ (rg))$

$\quad ,\langle$ New part of the invariant $(FREE)$. B.5.5.3 $\rangle$

$\qquad \therefore \mathbf{dom}\ L\ .\ locks\ (rg) \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

$\quad |\!\rangle$

$\quad \setminus\ and.\ in$

$\qquad \therefore L_{mod}.\ inv\ (rg)$

$\quad]$

$]$

B.5.5.2.

$\langle$ Old part of the invariant $(FREE)$. B.5.5.2 $\rangle \equiv$

$pLeft\ (hyp\_inv)$

$\quad \therefore K_{mod}.\ inv\ (L.\ K\ (\overleftarrow{rg}))$

$\setminus\ rsubst(pLeft(eval\_post))$

$\quad \therefore K_{mod}.\ inv\ (L.\ K\ (rg))$

B.5.5.3.

$\langle$ New part of the invariant $(FREE)$. B.5.5.3 $\rangle \equiv$

$pRight\ (hyp\_inv)$

$\quad \therefore \mathbf{dom}\ L\ .\ locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\ L\ .\ cont\ (\overleftarrow{rg})$

$\setminus\ rsubst(co\_post)$

$\quad \therefore \mathbf{dom}\ L\ .\ locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

$\setminus\ subset\_prop.\ trans\ (aux)$

$\quad \therefore \mathbf{dom}\ L\ .\ locks\ (rg) \subseteq \mathbf{dom}\ L\ .\ cont\ (rg)$

B.5.5.4.

$\langle$ Auxiliary lemma. B.5.5.4 $\rangle \equiv$

$mfilter\_subset$

$\quad \therefore \mathbf{dom}([\, r1 : Rid \,;\, u : \; Uid \;\vdash \neg r1 = r \,] \rhd L.\, locks\; (\overleftarrow{rg})) \subseteq \mathbf{dom}\, L \,.\, locks\; (\overleftarrow{rg})$

$\setminus\; rsubst(pRight(eval\_post))$

$\quad \therefore \mathbf{dom}\; L \,.\, locks\; (rg) \subseteq \mathbf{dom}\; L \,.\, locks\; (\overleftarrow{rg})$

B.5.5.5.

$\langle$ Derivation of the evaluated postcondition of $FREE$ B.5.5.5 $\rangle \equiv$

$equiv \,.\, out\; (proj\_opeq(L\_FREE\_eval).post).\; down\; (hyp\_post)$

B.5.6 Checkin operation

$\langle$ Proof of $\;val\_op\; (L_{op}.CHECKIN, L_{mod}.inv)$ B.5.6 $\rangle \equiv$

$val\_xtnd\_in\; .sel_1(val\_xtnd\_st.sel_2(CHECKIN_{val}))$

$$
\begin{array}{lllll}
\setminus\; val\_strpre(P := [& in & : & Uid \otimes (File \otimes (Rid \otimes Rid))) \\
 & ;\; rg & : & L_{st} \\
 & ;\; oldr & := & sel_1(sel_2(sel_2(in))) \\
 & ;\; uid & := & sel_1(in) \\
 & \multicolumn{4}{l}{\vdash oldr \in \mathbf{dom}\; L \,.\, locks\; (rg)} \\
 & \multicolumn{4}{l}{\quad \wedge L.\, locks\; (rg) \nabla oldr = uid} \\
 &] & & &
\end{array}
$$

$\quad \therefore val\_op(L_{op}.CHECKIN, [\, rg : L_{st} \;\vdash K_{mod}.\; inv\; (L.\, K\; (rg))\,])$

$\setminus\; val\_xtnd\_inv(inv_1 := inv_E)(\langle$ Extension: $CHECKIN$ B.5.6.1 $\rangle).sel_2$

$\quad \therefore val\_op(L_{op}.CHECKIN, L_{mod}.inv)$

B.5.6.1.

$\langle$ Extension: $\;CHECKIN$ B.5.6.1 $\rangle \equiv$

$[\ i \quad : \quad Uid \otimes (File \otimes (Rid \otimes Rid));\ o \quad : \quad \mathbf{void}$

$;\overleftarrow{rg} \quad ? \quad L_{st}\ ; \qquad\qquad rg \quad ? \quad L_{st}$

$;\ uid\ := sel_1(i); \qquad\qquad f\ := sel_1(sel_2(i))$

$;\ or\ := sel_1(sel_2(sel_2(i))); \qquad\qquad nr\ := sel_2(sel_2(sel_2(i)))$

$\vdash [\ hyp\_inv \quad : \quad \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\ (\overleftarrow{rg})$

$\quad ;\ hyp\_pre \quad : \quad L_{op}\,.\, CHECKIN\ (i, \overleftarrow{rg}).\, pre$

$\quad ;\ hyp\_post \quad : \quad L_{op}\,.\, CHECKIN\ (i, \overleftarrow{rg}).\, post\ (o, rg)$

$\quad ;\ eval\_post\ := \langle$ Derivation of the evaluated postcondition of $CHECKIN$ B.5.6.3$\rangle$

$\qquad\qquad \therefore L.\, K\ (rg) = K.\, mk\ ((nr \mapsto f) \odot L.\, cont\ (\overleftarrow{rg}), insert(or, nr, L.\, dep\ (\overleftarrow{rg})))$

$\qquad\qquad \wedge\ L.\, locks\ (rg) = L.\, locks\ (\overleftarrow{rg})$

$\quad ;\ co\_post \quad := \langle$ Derivation of the post-content $(CHECKIN)$. B.5.6.2$\rangle$

$\qquad\qquad \therefore L.\, cont\ (rg) = (nr \mapsto f) \odot L.\, cont\ (\overleftarrow{rg})$

$\quad \vdash hyp\_inv$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\ (\overleftarrow{rg})$

$\quad \setminus\ subset\_prop.\, trans\ (snd := dom\_prop.subset)$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}((nr \mapsto f) \odot L.\, cont\ (\overleftarrow{rg}))$

$\quad \setminus\ rsubst(co\_post)$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\ (\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\ (rg)$

$\quad \setminus\ rsubst(pRight(eval\_post))$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\ (rg) \subseteq \mathbf{dom}\, L\,.\, cont\ (rg)$

$\quad]$

$]$

B.5.6.2.

$\langle$ Derivation of the post-content $(CHECKIN)$. B.5.6.2$\rangle \equiv$

$refl$

$\quad \therefore L.\, cont\ (rg) = K.\, cont\ (L.\, K\ (rg))$

$\setminus\ unfold(pLeft(eval\_post))$

$\quad \therefore L.\, cont\ (rg) = K.\, cont\ (K.\, mk\ ((nr \mapsto f) \odot L.\, cont\ (\overleftarrow{rg}), insert(or, nr, L.\, dep\ (\overleftarrow{rg}))))$

$\setminus\ unfold(def\_sel_1)$

$\quad \therefore L.\, cont\ (rg) = (nr \mapsto f) \odot L.\, cont\ (\overleftarrow{rg})$

B.5.6.3.

$\langle$ Derivation of the evaluated postcondition of $CHECKIN$ B.5.6.3$\rangle \equiv$

$equiv\,.\, out\ (proj\_opeq(L\_CHECKIN\_eval).post).\, down\ (hyp\_post)$

B.5.7 Checkout operation

$\langle$ Proof of $val\_op\ (L_{op}.CHECKOUT, L_{mod}.inv)$ B.5.7$\rangle \equiv$

$(val\_xtnd\_st.sel_2(CHECKOUT_{val})$

$\quad \therefore val\_op(L_{op}.CHECKOUT, [\ rg : L_{st}\ \vdash K_{mod}.\, inv\ (L.\, K\ (rg))\ ])$

$\setminus\ val\_xtnd\_inv(inv_1 := inv_E)(\langle$ Extension: $CHECKOUT$ B.5.7.1$\rangle).sel_2$

$\quad \therefore val\_op(L_{op}.CHECKOUT, L_{mod}.inv)$

B.5.7.1.

$\langle$ Extension: $CHECKOUT$ B.5.7.1 $\rangle \equiv$

$[\; r \;:\; Rid \;;\; f \;:\; File \;;\; \overleftarrow{rg}, rg \;?\; L_{st}$

$\vdash [\; hyp\_inv \;:\; \mathbf{dom}\, L\,.\, locks\,(\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\,(\overleftarrow{rg})$

$\quad ;\; hyp\_pre \;:\; L_{op}\,.\, CHECKOUT\,(r, \overleftarrow{rg}).\, pre$

$\quad ;\; hyp\_post \;:\; L_{op}\,.\, CHECKOUT\,(r, \overleftarrow{rg}).\, post\,(f, rg)$

$\quad ;\; eval\_post \;:=$ $\langle$ Derivation of the evaluated postcondition of $CHECKOUT$ B.5.7.2 $\rangle$

$\qquad\qquad \therefore (f = L.\, cont\,(\overleftarrow{rg}) \nabla\, r \wedge L\,.\, K\,(rg) = L.\, K\,(\overleftarrow{rg}))$

$\qquad\qquad \wedge\, L.\, locks\,(rg) = L.\, locks\,(\overleftarrow{rg})$

$\quad ;\; co\_post \;:=\; refl$

$\qquad\qquad \therefore L.\, cont\,(rg) = K\,.\, cont\,(L.\, K\,(rg))$

$\qquad\qquad \backslash\; unfold(pRight(pLeft(eval\_post)))$

$\qquad\qquad \therefore L.\, cont\,(rg) = K\,.\, cont\,(L.\, K\,(\overleftarrow{rg}))$

$\qquad\qquad \therefore L.\, cont\,(rg) = L.\, cont\,(\overleftarrow{rg})$

$\quad \vdash hyp\_inv$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\,(\overleftarrow{rg}) \subseteq \mathbf{dom}\, L\,.\, cont\,(\overleftarrow{rg})$

$\qquad \backslash\; rsubst(pRight(eval\_post))$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\,(rg) \subseteq \mathbf{dom}\, L\,.\, cont\,(\overleftarrow{rg})$

$\qquad \backslash\; rsubst(co\_post)$

$\qquad \therefore \mathbf{dom}\, L\,.\, locks\,(rg) \subseteq \mathbf{dom}\, L\,.\, cont\,(rg)$

$\quad]$

$]$

B.5.7.2.

$\langle$ Derivation of the evaluated postcondition of $CHECKOUT$ B.5.7.2 $\rangle \equiv$

$equiv\,.\, out\,(proj\_opeq(L\_CHECKOUT\_eval).post).\, down\,(hyp\_post)$

B.5.8 Evaluation

$\langle$ Evaluation proofs B.5.8 $\rangle \equiv$

$[\![\; L\_RESET\_eval\_pr \;:=\; def\_join$

$\qquad\qquad \therefore L_{op}.\, RESET$

$\qquad\qquad =_{op} [\; v \;:\; \mathbf{void}$

$\qquad\qquad\quad ;\; L_{st}$

$\qquad\qquad\quad \vdash \langle\!|\; pre \;:=\; true \wedge true$

$\qquad\qquad\qquad ,\; post \;:=\; [\; v \;:\; \mathbf{void} \,;\, rg : L_{st}$

$\qquad\qquad\qquad\qquad \vdash L\,.\, K\,(rg) = K.\, mk\,(\langle\rangle, \tau) \wedge L.\, locks\,(rg) = \langle\rangle$

$\qquad\qquad\qquad\qquad]$

$\qquad\qquad\qquad |\!\rangle$

$\qquad\qquad\quad]$

; $L\_OPEN\_eval\_pr \quad := \; \mathit{refl\_opeq}$

$\therefore L_{op}.OPEN =_{op} K_{op}.\ OPEN \bigwedge (init.\ in\ (MAP_{op}.CREATE))$

$\backslash\ subst\_opeq(P := [\ op\ : op_{in}(File \otimes Rid, Rid \xrightarrow{m} Uid)$

$\vdash L_{op}\ .OPEN =_{op} K_{op}.\ OPEN \bigwedge op$

$]$

$, def\_init\_in)$

$\backslash\ subst\_opeq(P := [\ op\ : op_{in}(File \otimes Rid, L_{st})$

$\vdash L_{op}\ .OPEN =_{op} op$

$]$

$, def\_join)$

$]\!]$

B.6 Extension to robust operations: proofs

$\langle$ Proof of $val\_op\ (T_{op}.RESET, L_{mod}.inv)$. B.6 $\rangle \equiv$

L_RESET_{val}

$\therefore val\_op(T_{op}.RESET, L_{mod}.inv)$

B.6.1.

$\langle$ Proof of $val\_op\ (T_{op}.OPEN, L_{mod}.inv)$. B.6.1 $\rangle \equiv$

$val\_odisj\ (L\_OPEN_{val}, val\_complpre(L\_OPEN_{val}))$

$\therefore val\_op(T_{op}.OPEN, L_{mod}.inv)$

B.6.2.

$\langle$ Proof of $val\_op\ (T_{op}.SET, L_{mod}.inv)$. B.6.2 $\rangle \equiv$

$val\_odisj\ (L\_SET_{val}, val\_complpre(L\_SET_{val}))$

$\therefore val\_op(T_{op}.SET, L_{mod}.inv)$

B.6.3.

$\langle$ Proof of $val\_op\ (T_{op}.FREE, L_{mod}.inv)$. B.6.3 $\rangle \equiv$

$val\_odisj\ (L\_FREE_{val}, val\_complpre(L\_FREE_{val}))$

$\therefore val\_op(T_{op}.FREE, L_{mod}.inv)$

B.6.4.

$\langle$ Proof of $val\_op\ (T_{op}.CHECKOUT, L_{mod}.inv)$. B.6.4 $\rangle \equiv$

$val\_odisj\ (L\_CHECKOUT_{val}, val\_complpre(L\_CHECKOUT_{val}))$

$\therefore val\_op(T_{op}.CHECKOUT, L_{mod}.inv)$

B.6.5.

$\langle$ Proof of $val\_op\ (T_{op}.CHECKIN, L_{mod}.inv)$. B.6.5 $\rangle \equiv$

$val\_odisj\ (L\_CHECKIN_{val}, val\_complpre(L\_CHECKIN_{val}))$

$\therefore val\_op(T_{op}.CHECKIN, L_{mod}.inv)$

B.7 Application theories

This appendix contains two auxiliary theories used in the development. Both theories are rather specific to the application area (efficient storage techniques for files), but still too general to be included as part of the actual development.

B.7.1 Files

Files are seen as sequences of lines. Lines are sequences of characters bounded by the maximal length. The difference between two files is specified by means of file deltas (see next section).

$\langle$ Files. B.7.1 $\rangle \equiv$

context *Files* :=

$[\![\ Char \quad : \quad sort$

$;\ Line \quad := seq\,(Char)$

$;\ File \quad := seq\,(Line)$

$;\ LineLength \quad : \quad nat$

$;\ wff_L \quad := [\, l : Line\ \vdash \mathrm{len} l \le LineLength\,]$

$;\ wff_F \quad := [\, f\ :\ File$

$\quad \vdash \forall [\, l : Line\ \vdash l \in \mathrm{elems} f \Rightarrow wff_L(l)\,]$

$\quad]$

$;\ diff \quad : \quad [File; File \vdash \Delta(Line)\,]$

$;\ prop\_diff \quad : \quad [f1, f2\ ?\ File$

$\quad \vdash wff_\Delta(diff(f1, f2)) \wedge f1 \oplus diff(f1, f2) = f2$

$\quad]$

$;\ prop\_diff\_wff \quad : \quad [f1, f2\ ?\ File$

$\quad \vdash [wff_F(f1); wff_F(f2) \vdash wff_F(changed(diff(f1, f2)))\,]$

$\quad]$

$]\!]$

B.7.2 Delta technique

Deltas record modifications that specify the transformation of a file into another file. This datastructure is central for the efficient data management in revision control systems, see [].

$\langle$ Deltas. B.7.2 $\rangle \equiv$

$[\![\ \langle$ Delta Units B.7.2.1 $\rangle$

$;\ \langle$ Full Deltas B.7.2.9 $\rangle$

$]\!]$

Delta units

$\langle$ Delta Units B.7.2.1 $\rangle \equiv$

$[\![\ \langle$ Construction of Delta Units B.7.2.2 $\rangle$

$;\ \langle$ Operations upon Delta Units B.7.2.3 $\rangle$

$;\ \langle$ Derived Laws of Delta Units B.7.2.8 $\rangle$

$]\!]$

B.7.2.2.

$\langle$ Construction of Delta Units B.7.2.2 $\rangle \equiv$

$[\![\, \Delta_u \quad := [\, s : sort \vdash nat \otimes (seq(s) \otimes nat)\,]$

$;\, \langle\, (\cdot), (\cdot), (\cdot)\, \rangle_{\Delta_u} := [\, s \;?\; sort\, ;\, pos \,:\, nat\, ;\, ins \,:\, seq\,(s);\, del \,:\, nat$

$\vdash pos \mapsto (ins \mapsto del)$

$]$

$]\!]$

B.7.2.3.

$\langle$ Operations upon Delta Units B.7.2.3 $\rangle \equiv$

$[\![\, \langle$ Projection Functions B.7.2.4 $\rangle$

$; \langle$ Test of Well-Formedness of Delta Units B.7.2.5 $\rangle$

$; \langle$ Application of a Delta Unit to a Sequence B.7.2.6 $\rangle$

$; \langle$ Overwrite a Delta Unit Inserts by Ascending Integers B.7.2.7 $\rangle$

$]\!]$

B.7.2.4.

$\langle$ Projection Functions B.7.2.4 $\rangle \equiv$

$[\![\, pos \;:= [\, s \;?\; sort\, ;\, du : \Delta_u(s) \vdash sel_1(du)\,]$

$;\; ins \;:= [\, s \;?\; sort\, ;\, du : \Delta_u(s) \vdash sel_1(sel_2(du))\,]$

$;\; del \;:= [\, s \;?\; sort\, ;\, du : \Delta_u(s) \vdash sel_2(sel_2(du))\,]$

$]\!]$

B.7.2.5.

$\langle$ Test of Well-Formedness of Delta Units B.7.2.5 $\rangle \equiv$

$[\![\, wff_{\Delta_u} \quad : [\, s \;?\; sort\, ; \Delta_u(s) \vdash prop\,]$

$;\, def\_wff_{\Delta_u} : [\, s \;?\; sort\, ;\, du \;?\, \Delta_u(s)$

$\vdash wff_{\Delta_u}(du) \Leftrightarrow pos(du)$ ¿ $0 \wedge (\neg(del(du) = 0) \Rightarrow ins(du) = \langle\rangle)$

$]$

$]\!]$

B.7.2.6.

$\langle$ Application of a Delta Unit to a Sequence B.7.2.6 $\rangle \equiv$

$[\![\, (\cdot) \oplus_u (\cdot) \quad : \quad [\, s \;?\; sort \vdash [\, seq\,(s); \Delta_u(s) \vdash seq(s)\,]\,]$

$;$ **opspec** $\oplus_u > =$

$;\; def\_apply\_unit := [\, s \;?\; sort\, ;\, l \;?\; seq\,(s);\, du \;?\, \Delta_u(s)$

$\vdash \langle\!|\, [\, del\,(du) = 0 \vdash l \oplus_u du = paste(l, ins(du), pos(du))\,]$

$, [\neg\, del\,(du) = 0 \vdash l \oplus_u du = cut(l, pos(du), del(du))\,]$

$|\!\rangle$

$]$

$]\!]$

B.7.2.7.

$\langle$ Overwrite a Delta Unit Inserts by Ascending Integers B.7.2.7 $\rangle \equiv$

$number_{\Delta_u} := [\, s \;?\; sort \,;\, n \;:\; nat \,;\, du \;: \Delta_u(s)$
$\quad \vdash \langle\; pos(du), count\_up(n, \text{len}\, ins\; (du\; \rangle_{\Delta_u}), del(du))$
$\quad]$

B.7.2.8.

$\langle$ Derived Laws of Delta Units B.7.2.8 $\rangle \equiv$

$[\![\; prop\_unit \;: [\, s \;?\; sort \,;\, l \;?\; seq\,(s)$
$\quad \vdash \langle\!|\; wff \quad := [\, n \,?\, nat \,\vdash\, wff_{\Delta_u}(\langle\; succ(n), l, 0\; \rangle_{\Delta_u}) = true\,]$
$\quad\;\; ,\, apply \;:= \langle\rangle \oplus_u \langle\; 1, l, 0\; \rangle_{\Delta_u} = l$
$\quad\;\; |\!\rangle$
$\quad]$
$]\!]$

Deltas

$\langle$ Full Deltas B.7.2.9 $\rangle \equiv$
$[\![$ $\langle$ Construction of Deltas B.7.2.10 $\rangle$
; $\langle$ Operations upon Deltas B.7.2.11 $\rangle$
; $\langle$ Derived Laws of Deltas B.7.2.17 $\rangle$
$]\!]$

B.7.2.10.

$\langle$ Construction of Deltas B.7.2.10 $\rangle \equiv$

$\Delta := [\, s : sort \,\vdash\, seq(\Delta_u(s))\,]$

B.7.2.11.

$\langle$ Operations upon Deltas B.7.2.11 $\rangle \equiv$
$[\![$ $\langle$ Test of Well-Formedness of Deltas B.7.2.12 $\rangle$
; $\langle$ Application of a Delta to a List B.7.2.13 $\rangle$
; $\langle$ Application of a Sequence of Deltas to a List B.7.2.14 $\rangle$
; $\langle$ Overwrite a Delta Inserts with Ascending Integers B.7.2.15 $\rangle$
; $\langle$ Apply a Map to a Deltas Inserts B.7.2.16 $\rangle$
; $\langle$ Lines changed by a Delta. B.7.2.18 $\rangle$
$]\!]$

B.7.2.12.

$\langle$ Test of Well-Formedness of Deltas B.7.2.12 $\rangle \equiv$

$[\![\; wff_{\Delta} \quad : [\, s \,?\, sort \,;\, \Delta(s) \vdash prop\,]$

; def_wff_{Δ} : [s ? $sort$; d ? $\Delta(s)$
$\vdash$[wff := [i : nat
$\vdash(1 \leq i \wedge i$ ¡ len$d) \Rightarrow pos(d\nabla i) + del(d\nabla i)$ ¡ $pos(d\nabla(i+1))$
]
$\vdash wff_{\Delta}(d) = wff_{\Delta_u} \rhd d = d \wedge \forall[\, i : nat \vdash wff(i)]$
]
]
⟧

B.7.2.13.

⟨ Application of a Delta to a List B.7.2.13 ⟩ ≡
⟦ $(\cdot) \oplus (\cdot)$: [s ? $sort$ $\vdash$ [$seq\,(s); \Delta(s) \vdash seq(s)$]]
; **opspec** $\oplus >$ =
; def_apply_delta : [s ? $sort$; l ? $seq\,(s)$
$\vdash$⟨| $empty$:= $l \oplus \langle\rangle = l$
, $recur$:= [du ? $\Delta_u(s)$; d ? $\Delta(s)$
$\vdash$[$unit$:= [$du1$: $\Delta_u(s)$
$\vdash(pos(du1) + \mathrm{len}(ins(du1))$ - $del(du1))$
$\mapsto (ins(du1) \mapsto del(du1))$
]
$\vdash l \oplus (du \odot d) = (l \oplus_u du) \oplus (unit * d)$
]
]
|⟩
]
⟧

B.7.2.14.

⟨ Application of a Sequence of Deltas to a List B.7.2.14 ⟩ ≡
⟦ $(\cdot) \oplus_s (\cdot)$: [s ? $sort$ $\vdash$ [$seq\,(s); seq(\Delta(s)) \vdash seq(s)$]]
; **opspec** $\oplus_s >$ =
; $def\_apply\_delta\_seq$: [s ? $sort$; l ? $seq\,(s)$
$\vdash$⟨| $empty$:= $l \oplus_s \langle\rangle = l$
, $recur$:= [d ? $\Delta(s)$; ds ? $seq\,(\Delta(s))$
$\vdash l \oplus_s (d \odot ds) = (l \oplus d) \oplus_s ds$
]
|⟩
]
⟧

B.7.2.15.

⟨ Overwrite a Delta Inserts with Ascending Integers B.7.2.15 ⟩ ≡

$$
\begin{array}{ll}
\llbracket\, number_{\Delta} & : [\, s\ ?\ sort\ ; nat; \Delta(s) \vdash \Delta(nat)\,] \\
;\ def\_number\_delta & : [\ s\ \ ?\ sort \\
 & \quad ;\ n\ \ ?\ nat \\
 & \quad \vdash \langle\!|\ empty\ := number_{\Delta}(n, \langle\rangle \therefore (\Delta(s))) = \langle\rangle \\
 & \qquad ,\ recur\ \ := [\ du\ \ ?\ \Delta_u(s) \\
 & \qquad\qquad\qquad\ ;\ d\ \ \ ?\ \Delta(s) \\
 & \qquad\qquad\qquad\ \vdash number_{\Delta}(n, du \odot d) \\
 & \qquad\qquad\qquad\quad = number_{\Delta_u}(n, du) \odot number_{\Delta}(n + \mathrm{len}\ ins\ (du), d) \\
 & \qquad\qquad\qquad\ ] \\
 & \qquad |\!\rangle \\
 & \quad] \\
\rrbracket &
\end{array}
$$

B.7.2.16.

$\langle$ Apply a Map to a Deltas Inserts B.7.2.16 $\rangle \equiv$

$$
\begin{array}{ll}
apply\_to\_inserts & := [s, t\ ?\ sort \\
 & \quad \vdash [\ m\ : s \xrightarrow{m} t;\ d\ : \Delta(s) \\
 & \qquad \vdash ([\ du : \Delta_u(s) \vdash \langle\ pos(du), m * ins(du), del(du\ \rangle_{\Delta_u})\,] * d) \therefore (\Delta(t)) \\
 & \qquad] \\
 & \quad]
\end{array}
$$

B.7.2.17.

$\langle$ Derived Laws of Deltas B.7.2.17 $\rangle \equiv$

$$
\begin{array}{ll}
\llbracket\ prop\_ok\_delta & : [\ s\ \ ?\ \ sort\ ;\ du\ \ ?\ \Delta_u(s) \\
 & \quad \vdash wff_{\Delta}(\langle\ du\ \rangle) = wff_{\Delta_u}(du) \\
 & \quad] \\
;\ prop\_apply\_delta\_seq2 & : [\ s\ \ ?\ \ sort\ ;\ l\ \ ?\ \ seq\ (s);\ d\ \ ?\ \Delta(s);\ ds\ \ ?\ \ seq\ (\Delta(s)) \\
 & \quad \vdash l \oplus_s (ds +\!\!+ \langle\ d\ \rangle) = (l \oplus_s ds) \oplus d \\
 & \quad] \\
;\ prop\_apply\_delta\_seq & : [\ s\ \ ?\ \ sort\ ;\ du\ \ ?\ \Delta_u(s) \\
 & \quad \vdash \langle\!|\ single\ := \langle\rangle \oplus_s \langle\ \langle\ du\ \rangle\ \rangle = \langle\rangle \oplus_u du \\
 & \qquad |\!\rangle \\
 & \quad] \\
\rrbracket &
\end{array}
$$

B.7.2.18.

$\langle$ Lines changed by a Delta. B.7.2.18 $\rangle \equiv$

$$
\begin{array}{ll}
\llbracket\ changed & : [\, s\ ?\ sort\ ; \Delta(s) \vdash seq(s)\,] \\
;\ def\_changed & : [\ s\ \ \ \ \ ?\ sort \\
 & \quad ; n, m\ ?\ nat \\
 & \quad ;\ l\ \ \ \ \ ?\ seq\ (s) \\
 & \quad \vdash \langle\!|\ unit\ :=\ changed\ (\langle\ \langle\ n, l, m\ \rangle\ \rangle_{\Delta_u}) = l \\
 & \qquad |\!\rangle \\
 & \quad]
\end{array}
$$

$]\!]$

B.8 Reification methodology

The reification methodology used in this case study is essentially an extension of the VDM-formalization in []. The main new material is a simple calculus for implicit VDM-operations. The operators of this calculus can be used to break downs large specifications into manageable parts. Furthermore, a small library of module interfaces has been added.

$\langle$ Methodology. B.8 $\rangle \equiv$

context *ReificationMethodology* $:=$

$[\![\langle$ Developments units: operations. B.8.1 $\rangle$

$;\langle$ Horizontal development: modules and the calculus of operations. B.8.1 $\rangle$

$;\langle$ Vertical development: reification. B.8.3 $\rangle$

$;\langle$ Library of module interfaces. B.8.5 $\rangle$

$]\!]$

B.8.1.

$\langle$ Horizontal development: modules and the calculus of operations. B.8.1 $\rangle \equiv$

$[\![\langle$ Modules. B.8.2 $\rangle$

$;\langle$ Calculus of operations. B.8.4 $\rangle$

$]\!]$

B.8.1 Operations

$\langle$ Developments units: operations. B.8.1 $\rangle \equiv$

$[\![\langle$ Signature of operations. B.8.1.1 $\rangle$

$;\langle$ Proof obligations for operations. B.8.1.2 $\rangle$

$]\!]$

Signature of operations

$\langle$ Signature of operations. B.8.1.1 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\, op((\cdot),(\cdot),(\cdot)) & := & [in, out, state :\ sort \\
 & & \vdash \dfrac{in; state}{\langle\!|\, pre\ := \ prop\ , post\ := [out; state \vdash prop\,]\, |\!\rangle} \\
 & &] \\
;\ \mathbf{void} & : & sort \\
;\ \_ & : & \mathbf{void} \\
;\ op_{in}((\cdot),(\cdot)) & := & [in, state :\ sort\ \vdash op(in, \mathbf{void}, state)\,] \\
;\ op_{out}((\cdot),(\cdot)) & := & [out, state :\ sort\ \vdash op(\mathbf{void}, out, state)\,] \\
;\ op_{st}((\cdot)) & := & [\ state : sort\ \vdash op(\mathbf{void}, \mathbf{void}, state)\,] \\
;\ op_{fun}((\cdot),(\cdot)) & := & [in, out :\ sort\ \vdash op(in, out, \mathbf{void})\,]
\end{array}
$$

$]\!]$

Validity of operations

$\langle$ Proof obligations for operations. B.8.1.2 $\rangle \equiv$

$$
\begin{array}{l}
val\_op \; := [in, out, state \; ? \; sort \; ; \; op \; : op(in, out, state); \; inv \; : [state \vdash prop\,]\\
\qquad \vdash [\, i \; : \; in \; ; \; st_i \; : \; state\\
\qquad\quad \vdash \left|\begin{array}{l} inv\;(st_i); op(i, st_i).\,pre \\ \hline \langle\!| \; satisfiability \; := \exists_2[\, o : out \; ; st_o : \; state \; \vdash op(i, st_i).\,post\;(o, st_o)\,] \\ , \; preservation \; := [\; o : out \; ; st_o : \; state \; \vdash \left|\begin{array}{l} op\;(i, st_i).\,post\;(o, st_o) \\ \hline inv\;(st_o)\end{array}\right.] \\ |\!\rangle \end{array}\right.\\
\qquad\quad]\\
\qquad]
\end{array}
$$

B.8.2 Modules

$\langle$ Modules. B.8.2 $\rangle \equiv$

$[\![$ $\langle$ Operation lists. B.8.2.1 $\rangle$

; $\langle$ Signature of modules. B.8.2.2 $\rangle$

; $\langle$ Proof obligations for modules. B.8.2.3 $\rangle$

; $\langle$ Derived properties of modules. B.8.2.4 $\rangle$

$]\!]$

Operation lists

$\langle$ Operation lists. B.8.2.1 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\; oplist & : [sort \vdash \textbf{prim} \;]\\
; \langle\rangle_{(\cdot)} & : [\, state : sort \; \vdash oplist(state)\,]\\
; \langle(\cdot)\rangle & : [state, in, out \; ? \; sort \; \vdash [op(in, out, state) \vdash oplist(state)\,]\,]\\
; \; (\cdot) \odot (\cdot) & : [\, state \; ? \, sort \; \vdash [\; oplist\;(state); oplist(state) \vdash oplist(state)\,]\,]\\
; \; \textbf{OPSPEC left} \; \odot & \\
]\!] &
\end{array}
$$

Signature of modules

$\langle$ Signature of modules. B.8.2.2 $\rangle \equiv$

$$module \; := [\, state : sort \; \vdash \langle\!| \; inv \; := [state \vdash prop\,], \, ops \; := \; oplist\;(state) |\!\rangle\,]$$

Validity of modules

$\langle$ Proof obligations for modules. B.8.2.3 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\; val\_oplist & : & [\, state \; ? \; sort\\
 & & \vdash [\; oplist\;(state); [state \vdash prop\,] \vdash prop\,]\\
 & &]
\end{array}
$$

$$
\begin{array}{lll}
; \ \mathit{def\_val\_oplist} : & [\, \mathit{state} \ ? \ \mathit{sort} \, ; \, \mathit{inv} \ ? \, [\mathit{state} \vdash \mathit{prop}\,] \\
& \vdash \langle\!| \ \mathit{empty} \ := \ \mathit{val\_oplist} \ (\langle\rangle_{\mathit{state}}) \\
& \quad , \mathit{sing} \quad := [\mathit{in}, \mathit{out} \ ? \ \mathit{sort} \, ; \ \mathit{op} \ ? \ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \qquad\qquad \vdash \dfrac{\mathit{val\_oplist} \ (\langle \mathit{op} \rangle, \mathit{inv})}{\mathit{val\_op} \ (\mathit{op}, \mathit{inv})} \\
& \qquad\qquad] \\
& \quad , \mathit{cons} \quad := [\mathit{ol}_1, \mathit{ol}_2 \ ? \ \mathit{oplist} \ (\mathit{state}) \\
& \qquad\qquad \vdash \dfrac{\mathit{val\_oplist} \ (\mathit{ol}_1 \odot \mathit{ol}_2, \mathit{inv})}{\mathit{val\_oplist} \ (\mathit{ol}_1, \mathit{inv}) \wedge \mathit{val\_oplist}(\mathit{ol}_2, \mathit{inv})} \\
& \qquad\qquad] \\
& \quad |\!\rangle \\
&] \\
; \ \mathit{mod\_valid} & := [\, \mathit{state} \ ? \ \mathit{sort} \\
& \vdash [\, \mathit{mod} \ : \ \mathit{module} \ (\mathit{state}) \\
& \quad \vdash \mathit{val\_oplist} \ (\mathit{mod}.\mathit{ops}, \mathit{mod}.\mathit{inv}) \\
& \quad] \\
&] \\
]\!]
\end{array}
$$

Derived properties of modules

$\langle$ Derived properties of modules. B.8.2.4 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ val\_assembly_4\ := [in_1, out_1, in_2, out_2, in_3, out_3, in_4, out_4, state\ ?\ sort \\
\qquad \vdash [\ inv\ \ ?\ [state \vdash prop\,] \\
\qquad\quad ;\ op_1\ \ ?\ op(in_1, out_1, state) \\
\qquad\quad ;\ op_2\ \ ?\ op(in_2, out_2, state) \\
\qquad\quad ;\ op_3\ \ ?\ op(in_3, out_3, state) \\
\qquad\quad ;\ op_4\ \ ?\ op(in_4, out_4, state) \\
\qquad\quad \vdash [\ opval\ \ : \langle\!|\ val\_op\ (op_1, inv) \\
\qquad\qquad\qquad ,\ val\_op\ (op_2, inv) \\
\qquad\qquad\qquad ,\ val\_op\ (op_3, inv) \\
\qquad\qquad\qquad ,\ val\_op\ (op_4, inv) \\
\qquad\qquad\qquad |\!\rangle \\
\qquad\qquad \vdash \langle\!|\langle\!|\langle\!|\ def\_val\_oplist\,.\,sing\,.\,up\ (opval.\mathbf{1}) \\
\qquad\qquad\quad ,\ def\_val\_oplist\,.\,sing\,.\,up\ (opval.\mathbf{2}) \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad\quad \backslash\ and.\ in\ \backslash\ def\_val\_oplist\,.\,cons\,.\,up \\
\qquad\qquad\quad ,\ def\_val\_oplist\,.\,sing\,.\,up\ (opval.\mathbf{3}) \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad\quad \backslash\ and.\ in\ \backslash\ def\_val\_oplist\,.\,cons\,.\,up \\
\qquad\qquad\quad ,\ def\_val\_oplist\,.\,sing\,.\,up\ (opval.\mathbf{4}) \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad\quad \backslash\ and.\ in\ \backslash\ def\_val\_oplist\,.\,cons\,.\,up \\
\qquad\qquad\qquad \therefore\ mod\_valid(\langle\!|\ inv\ \ :=\ inv\ ,\ ops\ \ := \langle op_1\rangle \odot \langle op_2\rangle \odot \langle op_3\rangle \odot \langle op_4\rangle |\!\rangle) \\
\qquad\qquad] \\
\qquad\quad] \\
\qquad] \\
]\!]
\end{array}
$$

B.8.3 Reification

⟨ Vertical development: reification. B.8.3 ⟩ ≡
⟦ ⟨ Operation reification. B.8.3.1 ⟩
; ⟨ Operation list reification. B.8.3.2 ⟩
; ⟨ Retrieve condition. B.8.3.3 ⟩
; ⟨ Module reification. B.8.3.4 ⟩
; ⟨ Derived properties of reification. B.8.3.5 ⟩
⟧

Operation reification layout simplified because of nest-size problem

⟨ Operation reification. B.8.3.1 ⟩ ≡

$$
\begin{array}{l}
(\cdot) \sqsubseteq^{op}_{(\cdot),(\cdot)} (\cdot) \; := [state_c, state_a, in, out \, ? \; sort \\
\qquad \vdash [\; op_c \;\; : op(in, out, state_c) \\
\qquad\quad ; \; op_a \;\; : op(in, out, state_a) \\
\qquad\quad ; \; inv_c \; : [state_c \vdash prop \,] \\
\qquad\quad ; \; retr \;\; : [state_c \vdash state_a \,] \\
\qquad\quad \vdash [\; i \; ? \; in \; ; \; st_{ci} \; ? \; state_c \; ; \; st_{ai} \; := \; retr \; (st_{ci}) \\
\qquad\qquad \vdash \dfrac{inv_c \; (st_{ci}); op_a(i, st_{ai}) . \, pre}{\begin{array}{l} \langle\!| \; domain \; := \; op_c \; (i, st_{ci}) . \, pre \\ , \; result \quad := [\; o \; ? \; out \; ; \; st_{co} \; ? \; state_c \; ; \; st_{ao} \; := \; retr \; (st_{co}) \\ \qquad\qquad\qquad \vdash \dfrac{op_c \; (i, st_{ci}) . \, post \; (o, st_{co})}{op_a \; (i, st_{ai}) . \, post \; (o, st_{ao})} \\ \qquad\qquad\qquad] \\ |\!\rangle \end{array}} \\
\qquad\quad] \\
\qquad] \\
]
\end{array}
$$

Operation list reification

$\langle$ Operation list reification. B.8.3.2 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\; (\cdot) \sqsubseteq^{oplist}_{(\cdot),(\cdot)} (\cdot) \;\; : & [state_c, state_a \, ? \; sort \\
& \vdash [\; oplist \; (state_c); oplist(state_a); [state_c \vdash prop \,]; [state_c \vdash state_a \,] \vdash prop \,] \\
&] \\
; \; def\_oplist\_reif \; : & [state_c, state_a \, ? \; sort \; ; \; retr \;\; ? \, [state_c \vdash state_a]; \; inv_c \;\; ? \, [state_c \vdash prop \,] \\
& \vdash \langle\!| \; empty \; := \langle\rangle_{state_c} \sqsubseteq^{oplist}_{inv_c, retr} \langle\rangle_{state_a} \\
& \quad , \; sing \quad := [\; in, out \, ? \; sort \\
& \qquad\qquad ; \; op_c \quad ? \; op(in, out, state_c); \; op_a \;\; ? \; op(in, out, state_a) \\
& \qquad\qquad \vdash \dfrac{\langle op_c \rangle \sqsubseteq^{oplist}_{inv_c, retr} \langle op_a \rangle}{op_c \sqsubseteq^{op}_{inv_c, retr} op_a} \\
& \qquad\qquad] \\
& \quad , \; join \quad := [ol_{c1}, ol_{c2} \, ? \;\; oplist \; (state_c); ol_{a1}, ol_{a2} \, ? \;\; oplist \; (state_a) \\
& \qquad\qquad \vdash [ol_{c1} \odot ol_{c2} \sqsubseteq^{oplist}_{inv_c, retr} ol_{a1} \odot ol_{a2} \\
& \qquad\qquad\quad \models ol_{c1} \sqsubseteq^{oplist}_{inv_c, retr} ol_{a1} \wedge ol_{c2} \sqsubseteq^{oplist}_{inv_c, retr} ol_{a2} \\
& \qquad\qquad\quad] \\
& \qquad\qquad] \\
& \quad |\!\rangle \\
&] \\
; \; (\cdot) \sqsubseteq^{mod}_{(\cdot)} (\cdot) \quad & := [state_c, state_a \, ? \;\; sort \\
& \vdash [\; mod_c \; : \; module \; (state_c); \; mod_a \; : \; module \; (state_a); \; retr \; : [state_c \vdash state_a \,] \\
& \quad \vdash mod_c \; . ops \sqsubseteq^{oplist}_{mod_c . inv, retr} mod_a . \; ops \\
& \quad] \\
&] \\
]\!]
\end{array}
$$

Retrieve condition

$\langle$ Retrieve condition. B.8.3.3 $\rangle \equiv$

$$
\begin{array}{l}
val\_retr \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad :=\\
[\,state_c, state_a\,?\;\; sort\\
;\; inv_c \qquad : [state_c \vdash prop];\;\; ainv\;\; : [state_a \vdash prop];\;\; retr\;\; : [state_c \vdash state_a\,]\\
\vdash \langle\!| \; preservation \;\; := [\; cst\,?\,state_c \;\vdash \dfrac{inv_c\;(cst)}{ainv\;(retr(cst))}]\\
\quad ,\; completeness \;\; := [\; ast\;\;?\;\; state_a\\
\qquad\qquad\qquad\qquad \vdash \dfrac{ainv\;(ast)}{\exists[\; cst : state_c \;\vdash inv_c(cst) \wedge retr(cst) = ast\,]}\\
\qquad\qquad\qquad\qquad]\\
\quad |\!\rangle\\
]
\end{array}
$$

Definition of reification

$\langle$ Module reification. B.8.3.4 $\rangle \equiv$

$$
\begin{array}{l}
(\cdot) \sqsubseteq_{(\cdot)} (\cdot) \;:= [state_c, state_a\,?\;\; sort\\
\qquad\qquad \vdash [\; mod_c \;:\; module\;(state_c);\;\; mod_a \;:\; module\;(state_a);\;\; retr\;\; : [state_c \vdash state_a\,]\\
\qquad\qquad\quad \vdash \langle\!| \; module \qquad := \langle\!| \; concrete \;:=\; mod\_valid\;(mod_c),\; abstract \;:=\; mod\_valid\;(mod_a)|\!\rangle\\
\qquad\qquad\qquad ,\; retrieval \quad :=\; val\_retr\;(mod_c.inv, mod_a.inv, retr)\\
\qquad\qquad\qquad ,\; reification \;\; := mod_c \sqsubseteq^{mod}_{retr}\; mod_a\\
\qquad\qquad\qquad |\!\rangle\\
\qquad\qquad\quad]\\
\qquad\qquad]
\end{array}
$$

Derived properties of reification

$\langle$ Derived properties of reification. B.8.3.5 $\rangle \equiv$

$[\![\; w4 \;:\; \mathbf{prim}$

$]\!]$

B.8.4 Calculus of operations

$\langle$ Calculus of operations. B.8.4 $\rangle \equiv$

$[\![\langle$ Op-Equality of operations. B.8.4.1 $\rangle$

$;\langle$ Operations upon operations. B.8.4.2 $\rangle$

$;\langle$ Derived properties of the calculus of operations. B.8.4.14 $\rangle$

$]\!]$

Equality of operations

$\langle$ Op-Equality of operations. B.8.4.1 $\rangle \equiv$

$$
\begin{array}{l}
[\![\; (\cdot) =_{op} (\cdot) \;: [in, out, state\,?\;\; sort\\
\qquad\qquad \vdash [op(in, out, state); op(in, out, state) \vdash prop\,]\\
\qquad\qquad]
\end{array}
$$

; ⟨ Axioms of equality. B.8.4.15 ⟩
; ⟨ Derived properties of equality. B.8.4.16 ⟩
$\rrbracket$

Signatures and definitions

⟨ Operations upon operations. B.8.4.2 ⟩ ≡
$\llbracket$ ⟨ Op-Conjunction of two operations. B.8.4.3 ⟩
; ⟨ Op-Disjunction of two operations. B.8.4.4 ⟩
; ⟨ Signature modifications. B.8.4.5 ⟩
; ⟨ Strengthening of precondition. B.8.4.12 ⟩
; ⟨ Complementation of precondition. B.8.4.13 ⟩
$\rrbracket$

B.8.4.3.

⟨ Op-Conjunction of two operations. B.8.4.3 ⟩ ≡

$$
\begin{array}{l}
\llbracket\ (\cdot) \bigwedge (\cdot) \quad : [in, out, state_1, state_2\ ?\ sort \\
\qquad\qquad\qquad \vdash [op(in, out, state_1); op(in, out, state_2) \vdash op(in, out, state_1 \otimes state_2)\,] \\
\qquad\qquad\qquad] \\
;\ \mathbf{opspec}\ \bigwedge \gg =_{op} \\
;\ \ def\_join\ : [in, out, state_1, state_2\ ?\ sort\ ;\ op_1\ ?\ op(in, out, state_1);\ op_2\ ?\ op(in, out, state_2)) \\
\qquad\qquad\qquad \vdash op_1 \bigwedge op_2 \\
\qquad\qquad\qquad\quad =_{op} [\ i\ :\ in\ ;\ st_i\ :\ state_1 \otimes state_2 \\
\qquad\qquad\qquad\qquad \vdash \langle\!| \ pre\ \ := \ op_1\ (i, sel_1(st_i)).\ pre \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ op_2(i, sel_2(st_i)).\ pre \\
\qquad\qquad\qquad\qquad\quad ,\ post\ \ := [\ o\ :\ out\ ;\ st_o\ :\ state_1 \otimes state_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdash op_1\ (i, sel_1(st_i)).\ post\ (o, sel_1(st_o)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ op_2(i, sel_2(st_i)).\ post\ (o, sel_2(st_o)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad\qquad\quad |\!\rangle \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad] \\
\rrbracket
\end{array}
$$

B.8.4.4.

⟨ Op-Disjunction of two operations. B.8.4.4 ⟩ ≡

$$
\begin{array}{l}
\llbracket\ (\cdot) \bigvee (\cdot) \quad : [in, out, state\ ?\ sort \\
\qquad\qquad\qquad \vdash [op(in, out, state); op(in, out, state) \vdash op(in, out, state)\,] \\
\qquad\qquad\qquad] \\
;\ \mathbf{OPSPEC}\ \bigvee \gg =_{op}
\end{array}
$$

$$
\begin{array}{l}
;\ \mathit{def\_opor}\ : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \mathit{sort}\ ; \mathit{op}_1, \mathit{op}_2\ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
\qquad\qquad \vdash \mathit{op}_1 \bigvee \mathit{op}_2 \\
\qquad\qquad\quad =_{op} [\, i\ :\ \mathit{in}\ ;\ \mathit{st}_i\ :\ \mathit{state} \\
\qquad\qquad\qquad \vdash \langle\!| \mathit{pre}\ := \mathit{op}_1\ (i, \mathit{st}_i).\,\mathit{pre} \vee \mathit{op}_2\ (i, \mathit{st}_i).\,\mathit{pre} \\
\qquad\qquad\qquad\quad ,\ \mathit{post}\ := [\, o\ :\ \mathit{out}\ ;\ \mathit{st}_o\ :\ \mathit{state} \\
\qquad\qquad\qquad\qquad\qquad \vdash (\mathit{op}_1(i, \mathit{st}_i).\,\mathit{pre} \wedge \mathit{op}_1\ (i, \mathit{st}_i).\,\mathit{post}\ (o, \mathit{st}_o)) \\
\qquad\qquad\qquad\qquad\qquad\quad \vee (\mathit{op}_2(i, \mathit{st}_i).\,\mathit{pre} \wedge \mathit{op}_2\ (i, \mathit{st}_i).\,\mathit{post}\ (o, \mathit{st}_o)) \\
\qquad\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad\quad |\!\rangle \\
\qquad\qquad\qquad] \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.8.4.5.

$\langle$ Signature modifications. B.8.4.5 $\rangle \equiv$
$[\![$ $\langle$ Signature initialisation. B.8.4.6 $\rangle$
; $\langle$ Signature extension. B.8.4.9 $\rangle$
$]\!]$

B.8.4.6.

$\langle$ Signature initialisation. B.8.4.6 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathit{init}\ : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \mathit{sort} \\
\qquad \vdash \langle\!|\ \mathit{st}\ \ := [\mathit{op}_{fun}(\mathit{in}, \mathit{out}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\,] \\
\qquad\quad ,\ \mathit{in}\ \ := [\mathit{op}_{out}(\mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\,] \\
\qquad\quad ,\ \mathit{out}\ := [\mathit{op}_{in}(\mathit{in}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\,] \\
\qquad\quad |\!\rangle \\
\qquad]
\end{array}
$$

; $\langle$ Definition of signature initialisation. B.8.4.7 $\rangle$
$]\!]$

B.8.4.7.

$\langle$ Definition of signature initialisation. B.8.4.7 $\rangle \equiv$
$[\![$ $\langle$ Definition of input initialisation. B.8.4.8 $\rangle$
$]\!]$

B.8.4.8.

$\langle$ Definition of input initialisation. B.8.4.8 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathit{def\_init\_in}\ : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \mathit{sort}\ ;\ \mathit{op}\ \ ?\ \mathit{op}_{out}(\mathit{out}, \mathit{state}) \\
\qquad\qquad\qquad \vdash \mathit{init}\ .\ \mathit{in}\ (\mathit{op}) \\
\qquad\qquad\qquad\quad =_{op} [\ i\ :\ \mathit{in}\ ;\ \mathit{st}_i\ :\ \mathit{state} \\
\qquad\qquad\qquad\qquad\quad \vdash \langle\!|\ \mathit{pre}\ \ :=\ \mathit{op}\ (\_\_, \mathit{st}_i).\ \mathit{pre} \\
\qquad\qquad\qquad\qquad\qquad ,\ \mathit{post}\ := [\ o\ :\ \mathit{out}\ ;\ \mathit{st}_o\ :\ \mathit{state} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdash \mathit{op}\ (\_\_, \mathit{st}_i).\ \mathit{post}\ (o, \mathit{st}_o) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad\qquad\qquad |\!\rangle \\
\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad] \\
]\!]
\end{array}
$$

B.8.4.9.

⟨ Signature extension. B.8.4.9 ⟩ ≡

$$
\begin{array}{l}
[\![\ \mathit{xtnd}\ : [\mathit{in}, \mathit{out}, \mathit{state}, \mathit{new}\ ?\ \mathit{sort} \\
\qquad \vdash \langle\!|\ \mathit{st}\ \ \ := \langle\!| \mathit{sel}_1 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{new} \otimes \mathit{state})\,] \\
\qquad\qquad\qquad\qquad , \mathit{sel}_2 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state} \otimes \mathit{new})\,] \\
\qquad\qquad\qquad\qquad |\!\rangle \\
\qquad\quad ,\ \mathit{in}\ \ \ := \langle\!| \mathit{sel}_1 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{new} \otimes \mathit{in}, \mathit{out}, \mathit{state})\,] \\
\qquad\qquad\qquad\qquad , \mathit{sel}_2 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in} \otimes \mathit{new}, \mathit{out}, \mathit{state})\,] \\
\qquad\qquad\qquad\qquad |\!\rangle \\
\qquad\quad ,\ \mathit{out}\ := \langle\!| \mathit{sel}_1 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{new} \otimes \mathit{out}, \mathit{state})\,] \\
\qquad\qquad\qquad\qquad , \mathit{sel}_2 := [\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \vdash \mathit{op}(\mathit{in}, \mathit{out} \otimes \mathit{new}, \mathit{state})\,] \\
\qquad\qquad\qquad\qquad |\!\rangle \\
\qquad\quad |\!\rangle \\
\qquad]
\end{array}
$$

; ⟨ Definition of signature extension. B.8.4.10 ⟩

$]\!]$

B.8.4.10.

⟨ Definition of signature extension. B.8.4.10 ⟩ ≡

$[\![$ ⟨ Definition of state extension. B.8.4.11 ⟩

$]\!]$

B.8.4.11.

⟨ Definition of state extension. B.8.4.11 ⟩ ≡

$$
\begin{array}{l}
[\![\ \mathit{def\_xtnd\_stl}\ :[\mathit{in}, \mathit{out}, \mathit{state}, \mathit{new}\ ?\ \ \mathit{sort}\ ;\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\\
\qquad\qquad \vdash \mathit{xtnd}\ .\ \mathit{st}\ .\mathit{sel}_1(\mathit{op})\\
\qquad\qquad\quad =_{op}\ [\ i\ :\ \mathit{in}\ ;\ \mathit{st}_i\ :\ \mathit{new} \otimes \mathit{state}\\
\qquad\qquad\qquad \vdash\langle\!|\ \mathit{pre}\ \ := \ \mathit{op}\ (i, \mathit{sel}_2(\mathit{st}_i)).\ \mathit{pre}\\
\qquad\qquad\qquad\quad ,\ \mathit{post}\ := [\ o\ :\ \mathit{out}\ ;\ \mathit{st}_o\ :\ \mathit{new} \otimes \mathit{state}\\
\qquad\qquad\qquad\qquad\qquad \vdash \mathit{op}\ (i, \mathit{sel}_2(\mathit{st}_i)).\ \mathit{post}\ (o, \mathit{sel}_2(\mathit{st}_o)) \wedge \mathit{sel}_1(\mathit{st}_o) = \mathit{sel}_1(\mathit{st}_i)\\
\qquad\qquad\qquad\qquad\qquad]\\
\qquad\qquad\qquad\quad |\!\rangle\\
\qquad\qquad\qquad]\\
\qquad\qquad]\\
]\!]
\end{array}
$$

B.8.4.12.

$\langle$ Strengthening of precondition. B.8.4.12 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot) \wedge_{PRE} (\cdot)\qquad :[\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \ \mathit{sort}\\
\qquad\qquad\qquad \vdash[\mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}); [\mathit{in}; \mathit{state} \vdash \mathit{prop}\,] \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\,]\\
\qquad\qquad\qquad]\\
;\ \textbf{opspec}\ \ \wedge_{PRE} > =_{op}\\
;\ \ \mathit{def\_strengthen\_pre}\ :[\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \ \mathit{sort}\ ;\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state});\ p\ \ ?\ [\mathit{in}; \mathit{state} \vdash \mathit{prop}\,]\\
\qquad\qquad\qquad \vdash \mathit{op} \wedge_{PRE}\ p\\
\qquad\qquad\qquad\quad =_{op}\ [\ i\ :\ \mathit{in}\ ;\ \mathit{st}_i\ :\ \mathit{state}\\
\qquad\qquad\qquad\qquad \vdash\langle\!|\ \mathit{pre}\ \ := \ \mathit{op}\ (i, \mathit{st}_i).\ \mathit{pre} \wedge p\ (i, \mathit{st}_i)\\
\qquad\qquad\qquad\qquad\quad ,\ \mathit{post}\ := \ \mathit{op}\ (i, \mathit{st}_i).\ \mathit{post}\\
\qquad\qquad\qquad\qquad\quad |\!\rangle\\
\qquad\qquad\qquad\qquad]\\
\qquad\qquad\qquad]\\
]\!]
\end{array}
$$

B.8.4.13.

$\langle$ Complementation of precondition. B.8.4.13 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot)^{cPRE}\qquad :[\mathit{in}, \mathit{out}, \mathit{ext}\ ?\ \ \mathit{sort}\\
\qquad\qquad \vdash[\mathit{op}(\mathit{in}, \mathit{out}, \mathit{ext}) \vdash \mathit{op}(\mathit{in}, \mathit{out}, \mathit{ext})\,]\\
\qquad\qquad]\\
;\ \ \mathit{def\_compl\_pre}\ :[\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \ \mathit{sort}\ ;\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state})\\
\qquad\qquad \vdash \mathit{op}^{cPRE}\\
\qquad\qquad\quad =_{op}\ [\ i\ :\ \mathit{in}\ ;\ \mathit{st}_i\ :\ \mathit{state}\\
\qquad\qquad\qquad \vdash\langle\!|\ \mathit{pre}\ \ := \neg(\mathit{op}(i, \mathit{st}_i).\mathit{pre})\\
\qquad\qquad\qquad\quad ,\ \mathit{post}\ := [\ o : \mathit{out}\ ; \mathit{st}_o :\ \mathit{state}\ \vdash \mathit{st}_o = \mathit{st}_i\,]\\
\qquad\qquad\qquad\quad |\!\rangle\\
\qquad\qquad\qquad]\\
\qquad\qquad]\\
]\!]
\end{array}
$$

Derived properties

$\langle$ Derived properties of the calculus of operations. B.8.4.14 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \mathit{val\_subst\_inv} & : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \mathit{sort}\ ; \mathit{inv}_1, \mathit{inv}_2\ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \vdash [[\ \mathit{st} : \mathit{state}\ \vdash \mathit{inv}_1(\mathit{st}) \Leftrightarrow \mathit{inv}_2(\mathit{st})\ ] \\
& \quad \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv}_1)}{\mathit{val\_op}\ (\mathit{op}, \mathit{inv}_2)} \\
& \quad] \\
&] \\
;\ \mathit{val\_oconj} & : [\mathit{in}, \mathit{out}, \mathit{state}_1, \mathit{state}_2\ ?\ \ \mathit{sort} \\
& \vdash [\ \mathit{inv}_1\ \ ?\ [\mathit{state}_1 \vdash \mathit{prop}];\qquad \mathit{inv}_2\ \ ?\ [\mathit{state}_2 \vdash \mathit{prop}] \\
& \quad ;\ \mathit{op}_1\ \ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}_1);\ \mathit{op}_2\ \ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}_2) \\
& \quad \vdash [\ \mathit{inv}\ \ := [\ \mathit{st} : \mathit{state}_1 \otimes \mathit{state}_2\ \vdash \mathit{inv}_1(\mathit{sel}_1(\mathit{st})) \wedge \mathit{inv}_2(\mathit{sel}_2(\mathit{st}))\ ] \\
& \qquad \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}_1, \mathit{inv}_1);\ \mathit{val\_op}(\mathit{op}_2, \mathit{inv}_2)}{\mathit{val\_op}\ (\mathit{op}_1 \bigwedge \mathit{op}_2, \mathit{inv})} \\
& \qquad] \\
& \quad] \\
&] \\
;\ \mathit{val\_odisj} & : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \ \mathit{sort}\ ;\ \mathit{inv}\ \ ?\ [\mathit{state} \vdash \mathit{prop}]; \mathit{op}_1, \mathit{op}_2\ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}_1, \mathit{inv});\ \mathit{val\_op}(\mathit{op}_2, \mathit{inv})}{\mathit{val\_op}\ (\mathit{op}_1 \bigvee \mathit{op}_2, \mathit{inv})} \\
&] \\
;\ \mathit{val\_xtnd\_st} & : [\mathit{in}, \mathit{out}, \mathit{state}, \mathit{new}\ ?\ \ \mathit{sort}\ ;\ \mathit{inv}\ \ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \vdash \langle\!|\, \mathit{sel}_1 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{xtnd.}\ \mathit{st}\ \mathbf{.}\mathit{sel}_1(\mathit{op}), [\ \mathit{st} : \mathit{new} \otimes \mathit{state}\ \vdash \mathit{inv}(\mathit{sel}_2(\mathit{st}))\ ])} \\
& \quad , \mathit{sel}_2 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{xtnd.}\ \mathit{st}\ \mathbf{.}\mathit{sel}_2(\mathit{op}), [\ \mathit{st} : \mathit{state} \otimes \mathit{new}\ \vdash \mathit{inv}(\mathit{sel}_1(\mathit{st}))\ ])} \\
& \quad |\!\rangle \\
&] \\
;\ \mathit{val\_xtnd\_in} & : [\mathit{in}, \mathit{out}, \mathit{state}\ ?\ \ \mathit{sort}\ ;\ \mathit{inv}\ \ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \vdash \langle\!|\, \mathit{sel}_1 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{xtnd.}\ \mathit{in}\ \mathbf{.}\mathit{sel}_1(\mathit{op}), \mathit{inv})} \\
& \quad , \mathit{sel}_2 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{xtnd.}\ \mathit{in}\ \mathbf{.}\mathit{sel}_2(\mathit{op}), \mathit{inv})} \\
& \quad |\!\rangle \\
&]
\end{array}
$$

$$
\begin{array}{ll}
;\ \mathit{val\_xtnd\_inv} & : [\mathit{in}, \mathit{out}, \mathit{state} \ ?\ \mathit{sort} \\
& \vdash [\ \mathit{inv}_1\ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{inv}_2\ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \quad \vdash \dfrac{\begin{array}{l} [\ i : \mathit{in};\ o : \mathit{out};\ \mathit{st}_i, \mathit{st}_o \ ?\ \mathit{state} \\ \vdash \dfrac{\mathit{inv}_1\ (\mathit{st}_i);\ \mathit{op}(i, \mathit{st}_i).\mathit{pre};\ \mathit{op}(i, \mathit{st}_i).\ \mathit{post}\ (o, \mathit{st}_o)}{\mathit{inv}_1\ (\mathit{st}_o)} \\ ] \end{array}}{\begin{array}{l} \langle\!| \mathit{sel}_2 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv}_2)}{\mathit{val\_op}\ (\mathit{op}, [\ \mathit{st} : \mathit{state} \vdash \mathit{inv}_2(\mathit{st}) \wedge \mathit{inv}_1(\mathit{st})\ ])} \\ , \mathit{sel}_1 := \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv}_2)}{\mathit{val\_op}\ (\mathit{op}, [\ \mathit{st} : \mathit{state} \vdash \mathit{inv}_1(\mathit{st}) \wedge \mathit{inv}_2(\mathit{st})\ ])} \\ |\!\rangle \end{array}} \\
& \quad] \\
&] \\
;\ \mathit{val\_init\_in} & : [\mathit{out}, \mathit{state} \ ?\ \mathit{sort};\ \mathit{inv}\ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ ?\ \mathit{op}_{\mathit{out}}(\mathit{out}, \mathit{state}) \\
& \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{init}.\ \mathit{in}\ (\mathit{op}), \mathit{inv})} \\
&] \\
;\ \mathit{val\_strpre} & : [\ \mathit{in}, \mathit{out}, \mathit{state} \ ?\ \mathit{sort} \\
& \ ;\ \mathit{inv} \qquad ?\ [\mathit{state} \vdash \mathit{prop}] \\
& \ ;\ \mathit{op} \qquad ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \ ;\ P \qquad ?\ [\mathit{in}; \mathit{state} \vdash \mathit{prop}] \\
& \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{op} \wedge_{PRE} P, \mathit{inv})} \\
&] \\
;\ \mathit{val\_complpre} & : [\mathit{in}, \mathit{out}, \mathit{state} \ ?\ \mathit{sort};\ \mathit{inv}\ ?\ [\mathit{state} \vdash \mathit{prop}];\ \mathit{op}\ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{state}) \\
& \vdash \dfrac{\mathit{val\_op}\ (\mathit{op}, \mathit{inv})}{\mathit{val\_op}\ (\mathit{op}^{cPRE}, \mathit{inv})} \\
&] \\
]\!] &
\end{array}
$$

Axioms of equality

$\langle$ Axioms of equality. B.8.4.15 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \mathit{refl\_opeq} & : [\mathit{in}, \mathit{out}, \mathit{st} \ ?\ \mathit{sort};\ \mathit{op}\ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{st}) \\
& \vdash \mathit{op} =_{op} \mathit{op} \\
&] \\
;\ \mathit{sym\_opeq} & : [\mathit{in}, \mathit{out}, \mathit{st} \ ?\ \mathit{sort};\ \mathit{op}_1, \mathit{op}_2 \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{st}) \\
& \vdash \dfrac{\mathit{op}_1 =_{op} \mathit{op}_2}{\mathit{op}_2 =_{op} \mathit{op}_1} \\
&] \\
;\ \mathit{trans\_opeq} & : [\mathit{in}, \mathit{out}, \mathit{st} \ ?\ \mathit{sort};\ \mathit{op}_1, \mathit{op}_2, \mathit{op}_3 \ ?\ \mathit{op}(\mathit{in}, \mathit{out}, \mathit{st}) \\
& \vdash \dfrac{\mathit{op}_1 =_{op} \mathit{op}_2;\ \mathit{op}_2 =_{op} \mathit{op}_3}{\mathit{op}_1 =_{op} \mathit{op}_3} \\
&]
\end{array}
$$

$; \; subst\_opeq \; : [in, out, st \,?\; sort \,; op_1, op_2 \,?\; op(in, out, st); \; P \;?\; [op(in, out, st) \vdash prop]$

$\vdash [op_1 =_{op} op_2$

$\vdash \dfrac{P\ (op_1)}{P\ (op_2)}$

$]$

$]$

$]\!]$

Derived properties

$\langle$ Derived properties of equality. B.8.4.16 $\rangle \equiv$

$[\![\; proj\_opeq \; : [in, out, state \,?\; sort \,; op_1, op_2 \,?\; op(in, out, state); st_i, st_o \,?\; state \,; \; i \;?\; in \,; \; o \;?\; out$

$\vdash \dfrac{op_1 =_{op} op_2}{\begin{array}{l} \langle\!| \; pre \;\; := \; op_1\ (i, st_i).pre \Leftrightarrow op_2(i, st_i).\ pre \\ , \; post \; := \; op_1\ (i, st_i).\ post\ (o, st_o) \Leftrightarrow op_2(i, st_i).\ post\ (o, st_o) \\ |\!\rangle \end{array}}$

$]$

$]\!]$

B.8.5 Module interfaces

$\langle$ Library of module interfaces. B.8.5 $\rangle \equiv$

context *Module_Interface_Library* $:=$

$[\![\langle$ Specification of mappings. B.8.5.1 $\rangle$

; $\langle$ Specification of labelled trees. B.8.5.14 $\rangle$

; $\langle$ Specification of sequences. B.8.5.25 $\rangle$

$]\!]$

Mappings

$\langle$ Specification of mappings. B.8.5.1 $\rangle \equiv$

context *MAPPINGS_int* $:=$

$[\![\langle$ State and invariant (mappings). B.8.5.2 $\rangle$

; $\langle$ Operations (mappings). B.8.5.3 $\rangle$

; $\langle$ Validity of the operations (mappings). B.8.5.11 $\rangle$

; $\langle$ Module assembly (mappings). B.8.5.12 $\rangle$

; $\langle$ Validity of the module (mappings). B.8.5.13 $\rangle$

$]\!]$

B.8.5.2.

$\langle$ State and invariant (mappings). B.8.5.2 $\rangle \equiv$

$[\![\; MAP_{st} \;\; := [D, R : \; sort \; \vdash D \xrightarrow{m} R]$

$; \; MAP_{inv} \; := [D, R : \; sort \,; max : \; nat \; \vdash [\; st : D \xrightarrow{m} R \vdash max \geq \mathbf{card}(\mathbf{dom} st)\,]\,]$

$]\!]$

B.8.5.3.

$\langle$ Operations (mappings). B.8.5.3 $\rangle \equiv$

$MAP_{op} \; := [D, R \; ? \; sort$

$\quad \vdash \{\!| \; CREATE \; := \langle$ Create empty map. B.8.5.4 $\rangle$

$\quad\quad , \; INSERT \; := \langle$ Extend map by new association pair. B.8.5.5 $\rangle$

$\quad\quad , \; EMPTY \; := \langle$ Test if map is empty. B.8.5.6 $\rangle$

$\quad\quad , \; TEST \; := \langle$ Test if object is in domain of map. B.8.5.7 $\rangle$

$\quad\quad , \; APPLY \; := \langle$ Apply map to object in domain. B.8.5.8 $\rangle$

$\quad\quad , \; DELETE \; := \langle$ Delete association pair from map. B.8.5.9 $\rangle$

$\quad\quad , \; SIZE \; := \langle$ Compute the size of map. B.8.5.10 $\rangle$

$\quad\quad |\!\}$

$\quad]$

B.8.5.4.

$\langle$ Create empty map. B.8.5.4 $\rangle \equiv$

$[\_ : \mathbf{void}$

$; D \xrightarrow{m} R$

$\vdash \{\!| \; pre \; := \; true$

$\quad , \; post \; := [\_ : \mathbf{void} \, ; m_o : D \xrightarrow{m} R \vdash m_o = \langle\rangle \,]$

$\quad |\!\}$

$]$

$\quad \therefore \; op_{st}(D \xrightarrow{m} R)$

B.8.5.5.

$\langle$ Extend map by new association pair. B.8.5.5 $\rangle \equiv$

$[\; max \;\; ? \;\; nat$

$\vdash [\; in \; : (D \otimes R); \; m_i \; : D \xrightarrow{m} R$

$\quad \vdash [\; d \; := sel_1(in); \; r \; := sel_2(in)$

$\quad\quad \vdash \{\!| \; pre \; := d \notin \mathbf{dom} \; m_i \wedge \mathbf{card}(\mathbf{dom} m_i) \text{ ¡ } max$

$\quad\quad\quad , \; post \; := [\, \mathbf{void}; m_o : D \xrightarrow{m} R \vdash m_o = (d \mapsto r) \odot m_i \,]$

$\quad\quad\quad |\!\}$

$\quad\quad]$

$\quad]$

$\quad\quad \therefore \; op_{in}(D \otimes R, D \xrightarrow{m} R)$

$]$

B.8.5.6.

$\langle$ Test if map is empty. B.8.5.6 $\rangle \equiv$

$[\_ : \mathbf{void}\,;\ m_i\ : D \xrightarrow{m} R$

$\vdash \langle\!| \ pre\ \ := \ true$

$\quad ,\ post\ := [\ b\ :\ prop\ ;\ m_o\ : D \xrightarrow{m} R$

$\qquad\qquad \vdash (b \Leftrightarrow m_i = \langle\rangle) \wedge m_o = m_i$

$\qquad\qquad]$

$\quad |\!\rangle$

$]$

$\quad \therefore op_{out}(prop, D \xrightarrow{m} R)$

B.8.5.7.

$\langle$ Test if object is in domain of map. B.8.5.7 $\rangle \equiv$

$[\ d\ :\ D\ ;\ m_i\ : D \xrightarrow{m} R$

$\vdash \langle\!| \ pre\ \ := \ true$

$\quad ,\ post\ := [\ b\ :\ prop\ ;\ m_o\ : D \xrightarrow{m} R$

$\qquad\qquad \vdash (d \in \mathbf{dom}\, m_i \Rightarrow (b \Leftrightarrow true))$

$\qquad\qquad\quad \wedge\ (d \notin \mathbf{dom}\, m_i \Rightarrow (b \Leftrightarrow false))$

$\qquad\qquad\quad \wedge\ m_o = m_i$

$\qquad\qquad]$

$\quad |\!\rangle$

$]$

$\quad \therefore op(D, prop, D \xrightarrow{m} R)$

B.8.5.8.

$\langle$ Apply map to object in domain. B.8.5.8 $\rangle \equiv$

$[\ d\ :\ D\ ;\ m_i\ : D \xrightarrow{m} R$

$\vdash \langle\!| \ pre\ \ := d \in \mathbf{dom}\ m_i$

$\quad ,\ post\ := [\ r\ :\ R\ ;\ m_o\ : D \xrightarrow{m} R$

$\qquad\qquad \vdash m_o = m_i \wedge r = m_i \nabla\ d$

$\qquad\qquad]$

$\quad |\!\rangle$

$]$

$\quad \therefore op(D, R, D \xrightarrow{m} R)$

B.8.5.9.

$\langle$ Delete association pair from map. B.8.5.9 $\rangle \equiv$

$[\ d\ :\ D\ ;\ m_i\ : D \xrightarrow{m} R$

$\vdash \langle\!| \ pre\ \ := d \in \mathbf{dom}\ m_i$

$\quad ,\ post\ := [\,\mathbf{void}$

$\qquad\qquad ;\ m_o\ : D \xrightarrow{m} R$

$\qquad\qquad \vdash m_o = [\ x : D\ ;\ y :\ R\ \vdash \neg x = d\ ] \rhd m_i$

$\qquad\qquad]$

$\quad |\!\rangle$

$]$

$\therefore op_{in}(D, D \xrightarrow{m} R)$

B.8.5.10.

$\langle$ Compute the size of map. B.8.5.10 $\rangle \equiv$

$$
\begin{array}{l}
[\_ : \textbf{void} ;\ m_i\ : D \xrightarrow{m} R \\
\vdash \langle\!| \ pre\ \ := true \\
\quad ,\ post\ := [\ n\ \ :\ nat \\
\qquad\qquad ;\ m_o\ : D \xrightarrow{m} R \\
\qquad\qquad \vdash n = \mathbf{card}(\mathbf{dom}\, m_i) \wedge m_o = m_i \\
\qquad\qquad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op_{out}(nat, D \xrightarrow{m} R)
\end{array}
$$

B.8.5.11.

$\langle$ Validity of the operations (mappings). B.8.5.11 $\rangle \equiv$

$$
\begin{array}{lll}
[\![& MAP\_CREATE\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.CREATE, MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_INSERT\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.\ INSERT\ (max := max), MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_EMPTY\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.EMPTY, MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_TEST\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.TEST, MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_APPLY\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.APPLY, MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_DELETE\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.DELETE, MAP_{inv}(D, R, max)) \\
 & &] \\
; & MAP\_SIZE\_val & : [D, R\ ?\ sort\ ;\ max\ ?\ nat \\
 & & \vdash val\_op\ (MAP_{op}.SIZE, MAP_{inv}(D, R, max)) \\
 & &] \\
]\!] & &
\end{array}
$$

B.8.5.12.

$\langle$ Module assembly (mappings). B.8.5.12 $\rangle \equiv$

$MAP_{mod} \; := [D, R : \; sort \; ; \; max \; : \; nat$

$\qquad \vdash\langle\!| \; inv \; := \; MAP_{inv} \; (D, R, max)$

$\qquad \quad , \; ops \; := \langle MAP_{op}.CREATE \rangle \odot \langle MAP_{op}. \; INSERT \; (max := max) \rangle$

$\qquad \qquad \odot \langle MAP_{op}.EMPTY \rangle \odot \langle MAP_{op}.TEST \rangle \odot \langle MAP_{op}.APPLY \rangle$

$\qquad \qquad \odot \langle MAP_{op}.DELETE \rangle \odot \langle MAP_{op}.SIZE \rangle$

$\qquad \qquad \quad \therefore oplist(D \xrightarrow{m} R)$

$\qquad \quad |\!\rangle$

$\qquad]$

B.8.5.13.

$\langle$ Validity of the module (mappings). B.8.5.13 $\rangle \equiv$

$[\![\; valid\_MAP \qquad :=$

$\; [D, R \; ? \; sort$

$\; ; \; max \; ? \; nat$

$\; \vdash\langle\!|\langle\!|\langle\!|\langle\!|\langle\!|\langle\!| \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_CREATE\_val(D := D, R := R, max := max))$

$\qquad , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_INSERT\_val(D := D, R := R, max := max))$

$\qquad |\!\rangle$

$\qquad \backslash \; and. \; in$

$\qquad \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\qquad , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_EMPTY\_val(D := D, R := R, max := max))$

$\qquad |\!\rangle$

$\qquad \backslash \; and. \; in$

$\qquad \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\quad , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_TEST\_val(D := D, R := R, max := max))$

$\quad |\!\rangle$

$\quad \backslash \; and. \; in$

$\quad \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\quad , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_APPLY\_val(D := D, R := R, max := max))$

$\quad |\!\rangle$

$\quad \backslash \; and. \; in$

$\quad \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\; , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_DELETE\_val(D := D, R := R, max := max))$

$\; |\!\rangle$

$\; \backslash \; and. \; in$

$\; \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\; , \; def\_val\_oplist \; . \; sing \; . \; up \; (MAP\_SIZE\_val(D := D, R := R, max := max))$

$\; |\!\rangle$

$\; \backslash \; and. \; in$

$\; \backslash \; def\_val\_oplist. \; cons \; . \; up$

$\;]$

$\quad \therefore [D, R \, ? \; sort \, ; max \, ? \; nat \; \vdash mod\_valid(MAP_{mod}(D, R, max))]$

$]\!]$

Labelled trees

$\langle$ Specification of labelled trees. B.8.5.14 $\rangle \equiv$
context $TREES\_int :=$
$[\![\,\langle$ State and invariant (trees). B.8.5.15 $\rangle$
; $\langle$ Operations (trees). B.8.5.16 $\rangle$
; $\langle$ Validity of the operations (trees). B.8.5.22 $\rangle$
; $\langle$ Module assembly (trees). B.8.5.23 $\rangle$
; $\langle$ Validity of the module (trees). B.8.5.24 $\rangle$
$]\!]$

B.8.5.15.

$\langle$ State and invariant (trees). B.8.5.15 $\rangle \equiv$
$[\![\; TREE_{st} \;:= [\, S : sort \vdash tree(S)\,]$
$;\; TREE_{inv} := [\, S : sort \vdash [\, t : tree\,(S) \vdash nodup(t)\,]\,]$
$]\!]$

B.8.5.16.

$\langle$ Operations (trees). B.8.5.16 $\rangle \equiv$
$TREE_{op} := [\, S \;?\; sort$
$\vdash \langle\!| \; CREATE := \langle$ Create empty tree. B.8.5.17 $\rangle$
$, ROOT := \langle$ Create root. B.8.5.18 $\rangle$
$, INSERT := \langle$ Insert object into tree. B.8.5.19 $\rangle$
$, TEST := \langle$ Test if object occurs in tree. B.8.5.20 $\rangle$
$, PATH := \langle$ Compute path from tree-root to tree-object. B.8.5.21 $\rangle$
$|\!\rangle$
$]$

B.8.5.17.

$\langle$ Create empty tree. B.8.5.17 $\rangle \equiv$
$[\_ : \mathbf{void}$
$;\; tree\,(S)$
$\vdash \langle\!| \; pre \;:= true$
$, post := [\,\mathbf{void}; tr_o : tree\,(S) \vdash tr_o = \tau\,]$
$|\!\rangle$
$]$
$\therefore op_{st}(tree(S))$

B.8.5.18.

$\langle$ Create root. B.8.5.18 $\rangle \equiv$

$$
\begin{array}{l}
[\, r \,:\, S \\
;\; tree\,(S) \\
\vdash \langle\!\langle\, pre \;\; := \; true \\
\quad ,\, post \;:= [\_ : \mathbf{void}\,;\; tr_o \;:\; tree\,(S) \\
\qquad\qquad\quad \vdash tr_o = node(r, \langle\rangle) \\
\qquad\qquad\quad] \\
\quad \rangle\!\rangle \\
] \\
\quad \therefore op_{in}(S, tree(S))
\end{array}
$$

B.8.5.19.

$\langle$ Insert object into tree. B.8.5.19 $\rangle \equiv$

$$
\begin{array}{l}
[\, in \,:\, S \otimes S \,;\; tr_i \,:\, tree\,(S) \\
\vdash [\; a \;:= sel_1(in);\; b \;:= sel_2(in) \\
\quad \vdash \langle\!\langle\, pre \;\; := b \notin info(tr_i) \\
\qquad ,\, post \;:= [\, \mathbf{void}; tr_o :\; tree\,(S) \vdash tr_o = insert(a, b, tr_i)\,] \\
\qquad \rangle\!\rangle \\
\quad] \\
] \\
\quad \therefore op_{in}(S \otimes S, tree(S))
\end{array}
$$

B.8.5.20.

$\langle$ Test if object occurs in tree. B.8.5.20 $\rangle \equiv$

$$
\begin{array}{l}
[\, l \,:\, S \,;\; tr_i \,:\, tree\,(S) \\
\vdash \langle\!\langle\, pre \;\; := \; true \\
\quad ,\, post \;:= [\, b \,:\, prop \,;\; tr_o \;:\; tree\,(S) \\
\qquad\qquad\quad \vdash tr_o = tr_i \\
\qquad\qquad\qquad \wedge (l \in info(tr_i) \Rightarrow (b \Leftrightarrow true)) \\
\qquad\qquad\qquad \wedge (l \notin info(tr_i) \Rightarrow (b \Leftrightarrow false)) \\
\qquad\qquad\quad] \\
\quad \rangle\!\rangle \\
] \\
\quad \therefore op(S, prop, tree(S))
\end{array}
$$

B.8.5.21.

$\langle$ Compute path from tree-root to tree-object. B.8.5.21 $\rangle \equiv$

$$
\begin{array}{l}
[\, a \,:\, S \,;\; tr_i \,:\, tree\,(S) \\
\vdash \langle\!\langle\, pre \;\; := a \in info(tr_i) \\
\quad ,\, post \;:= [\, p \,:\, seq\,(S);\; tr_o \;:\; tree\,(S) \\
\qquad\qquad\quad \vdash tr_o = tr_i \wedge p = init\_path(a, tr_i) \\
\qquad\qquad\quad] \\
\quad \rangle\!\rangle \\
] \\
\quad \therefore op(S, seq(S), tree(S))
\end{array}
$$

B.8.5.22.

$\langle$ Validity of the operations (trees). B.8.5.22 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ TREE\_CREATE_{val} & : [\, S\ ?\ sort\ \vdash val\_op(TREE_{op}.CREATE,\ TREE_{inv}(S))\,] \\
;\ TREE\_ROOT_{val} & : [\, S\ ?\ sort\ \vdash val\_op(TREE_{op}.ROOT,\ TREE_{inv}(S))\,] \\
;\ TREE\_INSERT_{val} & : [\, S\ ?\ sort\ \vdash val\_op(TREE_{op}.INSERT,\ TREE_{inv}(S))\,] \\
;\ TREE\_TEST_{val} & : [\, S\ ?\ sort\ \vdash val\_op(TREE_{op}.TEST,\ TREE_{inv}(S))\,] \\
;\ TREE\_PATH_{val} & : [\, S\ ?\ sort\ \vdash val\_op(TREE_{op}.PATH,\ TREE_{inv}(S))\,] \\
]\!]
\end{array}
$$

B.8.5.23.

$\langle$ Module assembly (trees). B.8.5.23 $\rangle \equiv$

$$
\begin{array}{l}
TREE_{mod} := [\, S : sort \\
\quad \vdash \langle\!|\ inv := TREE_{inv}\,(S) \\
\quad\ ,\ ops := \langle TREE_{op}.CREATE \rangle \\
\qquad\qquad \odot \langle TREE_{op}.ROOT \rangle \\
\qquad\qquad \odot \langle TREE_{op}.INSERT \rangle \\
\qquad\qquad \odot \langle TREE_{op}.TEST \rangle \\
\qquad\qquad \odot \langle TREE_{op}.PATH \rangle \\
\qquad\qquad\quad \therefore oplist(tree(S)) \\
\quad |\!\rangle \\
\quad]
\end{array}
$$

B.8.5.24.

$\langle$ Validity of the module (trees). B.8.5.24 $\rangle \equiv$

$TREE\_val\ : [\, S\ ?\ sort\ \vdash mod\_valid(TREE_{mod}(S))\,]$

Sequences

$\langle$ Specification of sequences. B.8.5.25 $\rangle \equiv$

context $SEQUENCES\_int :=$

$[\![$ $\langle$ State and invariant (sequences). B.8.5.26 $\rangle$

; $\langle$ Operations (sequences). B.8.5.27 $\rangle$

; $\langle$ Validity of the operations (sequences). B.8.5.32 $\rangle$

; $\langle$ Module assembly (sequences). B.8.5.33 $\rangle$

; $\langle$ Validity of the module (sequences). B.8.5.34 $\rangle$

$]\!]$

B.8.5.26.

$\langle$ State and invariant (sequences). B.8.5.26 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ SEQ_{st} & := [\, S : sort\ \vdash seq(S)\,] \\
;\ SEQ_{inv} & := [\, S : sort\ \vdash [\, st : seq\,(S) \vdash true\,]\,] \\
]\!]
\end{array}
$$

B.8.5.27.

$\langle$ Operations (sequences). B.8.5.27 $\rangle \equiv$

$$
\begin{array}{l}
SEQ_{op} := [\, S \;?\; sort \\
\qquad\qquad \vdash \langle\!| \; CREATE \; := \langle \text{ Create empty sequence. B.8.5.28} \rangle \\
\qquad\qquad\quad , INSERT \; := \langle \text{ Insert object at the right end of sequence. B.8.5.29} \rangle \\
\qquad\qquad\quad , READ \; := \langle \text{ Read object from sequence. B.8.5.30} \rangle \\
\qquad\qquad\quad , LENGTH \; := \langle \text{ Compute length of sequence. B.8.5.31} \rangle \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad]
\end{array}
$$

B.8.5.28.

$\langle$ Create empty sequence. B.8.5.28 $\rangle \equiv$

$$
\begin{array}{l}
[\_ : \mathbf{void} \,;\, sq_i \,:\, seq\,(S) \\
\vdash \langle\!| \; pre \; := true \\
\quad , post \; := [\, \mathbf{void}; sq_o : \; seq\,(S) \vdash sq_o = \langle\rangle \,] \\
\quad |\!\rangle \\
] \\
\quad \therefore op_{st}(seq(S))
\end{array}
$$

B.8.5.29.

$\langle$ Insert object at the right end of sequence. B.8.5.29 $\rangle \equiv$

$$
\begin{array}{l}
[\, s \,:\, S \,;\, sq_i \,:\, seq\,(S) \\
\vdash \langle\!| \; pre \; := true \\
\quad , post \; := [\_ : \mathbf{void} \,;\, sq_o \,:\, seq\,(S) \\
\qquad\qquad\quad \vdash sq_o = sq_i +\!\!+ \langle\; s \;\rangle \\
\qquad\qquad\quad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op_{in}(S, seq(S))
\end{array}
$$

B.8.5.30.

$\langle$ Read object from sequence. B.8.5.30 $\rangle \equiv$

$$
\begin{array}{l}
[\, n \,:\, nat \,;\, sq_i \,:\, seq\,(S) \\
\vdash \langle\!| \; pre \; := 0 \text{ ¡ } n \wedge n \leq \text{len } sq_i \\
\quad , post \; := [\, s \,:\, S \,;\, sq_o \,:\, seq\,(S) \\
\qquad\qquad\quad \vdash sq_o = sq_i \wedge s = sq_i \nabla\, n \\
\qquad\qquad\quad] \\
\quad |\!\rangle \\
] \\
\quad \therefore op(nat, S, seq(S))
\end{array}
$$

B.8.5.31.

$\langle$ Compute length of sequence. B.8.5.31 $\rangle \equiv$

$[\ v\ \ : \mathbf{void}$

$;\ sq_i\ :\ seq\ (S)$

$\vdash \langle\!|\ pre\ \ := true$

$\quad ,\ post\ := [\ l : nat\ ; sq_o :\ seq\ (S) \vdash l = \text{len}\ sq_i \wedge sq_o = sq_i\ ]$

$\quad |\!\rangle$

$]$

$\quad \therefore op_{out}(nat, seq(S))$

B.8.5.32.

$\langle$ Validity of the operations (sequences). B.8.5.32 $\rangle \equiv$

$[\![\ SEQ\_CREATE_{val}\ : [\ S\ ?\ sort\ \vdash val\_op(SEQ_{op}.CREATE, SEQ_{inv}(S))\ ]$

$;\ SEQ\_INSERT_{val}\ : [\ S\ ?\ sort\ \vdash val\_op(SEQ_{op}.INSERT, SEQ_{inv}(S))\ ]$

$;\ SEQ\_READ_{val}\ : [\ S\ ?\ sort\ \vdash val\_op(SEQ_{op}.READ, SEQ_{inv}(S))\ ]$

$;\ SEQ\_LENGTH_{val}\ : [\ S\ ?\ sort\ \vdash val\_op(SEQ_{op}.LENGTH, SEQ_{inv}(S))\ ]$

$]\!]$

B.8.5.33.

$\langle$ Module assembly (sequences). B.8.5.33 $\rangle \equiv$

$SEQ_{mod}\ := [\ S\ :\ sort$

$\qquad \vdash \langle\!|\ inv\ := SEQ_{inv}\ (S)$

$\qquad\quad ,\ ops\ := \langle SEQ_{op}.CREATE \rangle \odot \langle SEQ_{op}.INSERT \rangle$

$\qquad\qquad\qquad \odot \langle SEQ_{op}.READ \rangle \odot \langle SEQ_{op}.LENGTH \rangle$

$\qquad\qquad\qquad\quad \therefore oplist(seq(S))$

$\qquad\quad |\!\rangle$

$\qquad]$

B.8.5.34.

$\langle$ Validity of the module (sequences). B.8.5.34 $\rangle \equiv$

$SEQ\_val\ : [\ S\ ?\ sort\ \vdash mod\_valid(SEQ_{mod}(S))\ ]$

B.9 Library of basic theories

This appendix contains the fundamental theories needed for the development case study. It is an excerpt from a currently evolving library of basic theories in Deva.

B.9.1 Equality

$\langle$ Equality B.9.1 $\rangle \equiv$

context $Equality :=$

$[\![\ prop\ \ : sort$

$;\ (\cdot) = (\cdot)\ : [\ s\ ?\ sort\ \vdash [s; s \vdash prop\ ]\ ]$

$;\langle$ Substitution Principle of Equality B.9.1.1 $\rangle$
$;\langle$ Equivalence Relation Laws of Equality B.9.1.3 $\rangle$
$;\langle$ Derived Laws of Equality B.9.1.2 $\rangle$
$]\!]$

B.9.1.1.

$\langle$ Substitution Principle of Equality B.9.1.1 $\rangle \equiv$

$$[\![\ \mathit{subst}\ : [\, s \ ?\ \mathit{sort}\, ; a, b\ ?\ \ s\, ; P\ ?\ [s \vdash \mathit{prop}\,] \vdash [a = b \vdash \left| \frac{P\,(a)}{P\,(b)} \right.]\,]$$

$]\!]$

B.9.1.2.

$\langle$ Derived Laws of Equality B.9.1.2 $\rangle \equiv$
$[\![\langle$ Composition Laws of Equality B.9.1.4 $\rangle$
$;\langle$ Rewriting Laws of Equality B.9.1.5 $\rangle$
$]\!]$

B.9.1.3.

$\langle$ Equivalence Relation Laws of Equality B.9.1.3 $\rangle \equiv$

$$[\![\ \mathit{refl}\quad : [\, s\ ?\ \mathit{sort}\, ; a\ ?\ \ s\ \vdash a = a\,]$$

$$;\ \ \mathit{sym}\quad : [\, s\ ?\ \mathit{sort}\, ; a, b\ ?\ \ s\ \vdash \left| \frac{a = b}{b = a} \right.]$$

$$;\ \ \mathit{trans}\ : [\, s\ ?\ \mathit{sort}\, ; a, b, c\ ?\ \ s\ \vdash [a = b; b = c \vdash\ a = c\,]\,]$$

$]\!]$

B.9.1.4.

$\langle$ Composition Laws of Equality B.9.1.4 $\rangle \equiv$

$$[\![\ \mathit{trans\_product}\quad : [\, s\ ?\ \mathit{sort}\, ; a, b, c\ ?\ \ s\ \vdash \left| \frac{\begin{array}{l} \langle\!| a = b \\ , b = c \\ |\!\rangle \end{array}}{a = c} \right.]$$

$$;\ \ \mathit{indirect}\quad : [\, s\ ?\ \mathit{sort}\, ; a, b, c\ ?\ \ s\ \vdash [a = c; b = c \vdash\ a = b\,]\,]$$

$$;\ \ \mathit{indirect\_product}\ : [\, s\ ?\ \mathit{sort}\, ; a, b, c\ ?\ \ s\ \vdash \left| \frac{\begin{array}{l} \langle\!| a = c \\ , b = c \\ |\!\rangle \end{array}}{a = b} \right.]$$

$]\!]$

B.9.1.5.

$\langle$ Rewriting Laws of Equality B.9.1.5 $\rangle \equiv$

$[\![$ *rsubst* $: [\, s \,?\, sort \,;\, a, b \,?\; s \,;\, P \,?\, [s \vdash prop\,] \vdash [a = b \vdash \dfrac{P\ (b)}{P\ (a)}]\,]$

; *unfold* $: [s\_1, s\_2 \,?\; sort \,;\, a, b \,?\; s\_1 \,;\, c \,?\; s\_2 \,;\, F \,?\, [s\_1 \vdash s\_2\,] \vdash [a = b \vdash \dfrac{c = F(a)}{c = F(b)}]\,]$

; *fold* $: [s\_1, s\_2 \,?\; sort \,;\, a, b \,?\; s\_1 \,;\, c \,?\; s\_2 \,;\, F \,?\, [s\_1 \vdash s\_2\,] \vdash [a = b \vdash \dfrac{c = F(b)}{c = F(a)}]\,]$

; *unfold_left* $: [s\_1, s\_2 \,?\; sort \,;\, a, b \,?\; s\_1 \,;\, c \,?\; s\_2 \,;\, F \,?\, [s\_1 \vdash s\_2\,] \vdash [a = b \vdash \dfrac{F\ (a) = c}{F\ (b) = c}]\,]$

; *fold_left* $: [s\_1, s\_2 \,?\; sort \,;\, a, b \,?\; s\_1 \,;\, c \,?\; s\_2 \,;\, F \,?\, [s\_1 \vdash s\_2\,] \vdash [a = b \vdash \dfrac{F\ (a) = c}{F\ (b) = c}]\,]$

$]\!]$

B.9.2 Propositions

$\langle$ Propositions B.9.2 $\rangle \equiv$
context *Propositions* $:=$
$[\![\langle$ Construction of Propositions B.9.2.1 $\rangle$
; $\langle$ Substitution Principle for Propositions B.9.2.3 $\rangle$
; $\langle$ Basic Laws of Propositions B.9.2.4 $\rangle$
; $\langle$ Derived Laws of Propositions B.9.2.5 $\rangle$
$]\!]$

B.9.2.1.

$\langle$ Construction of Propositions B.9.2.1 $\rangle \equiv$
$[\![$ *true*, *false* : *prop*
; $(\cdot) \Rightarrow (\cdot)$
, $(\cdot) \wedge (\cdot)$
, $(\cdot) \vee (\cdot)$: $[prop; prop \vdash prop\,]$
; $(\cdot) \Leftrightarrow (\cdot)$ $:= (=)(s := prop)$
; $\neg\,(\cdot)$: $[prop \vdash prop\,]$
; $(\cdot) \not\Leftrightarrow (\cdot)$ $:= [p, q :\ prop\ \vdash \neg p \Leftrightarrow q\,]$$\langle$ Priorities B.9.2.2 $\rangle$
$]\!]$

B.9.2.2.

$\langle$ Priorities B.9.2.2 $\rangle \equiv$
; **opspec right** $\Rightarrow$
; **opspec left** $\wedge$
; **opspec left** $\vee$
; **opspec right** $\Leftrightarrow$

; **opspec** $\wedge \lessdot \neg$
; **opspec** $\vee \lessdot \wedge$
; **opspec** $\Rightarrow \lessdot \vee$
; **opspec** $\Leftrightarrow \lessdot \Rightarrow$
; **opspec** $\neg \lessdot =$
; **opspec left** $\not\Leftrightarrow \doteq \Leftrightarrow$

B.9.2.3.

$\langle$ Substitution Principle for Propositions B.9.2.3 $\rangle \equiv$
$[\![$ *prop_subst* $: [p\_1, p\_2\ ?\ prop\ ; Q\ ?\ [prop \vdash prop] \vdash [[p\_1 \models p\_2] \vdash [\ Q\ (p\_1) \vdash Q(p\_2)]]]$
$]\!]$

B.9.2.4.

$\langle$ Basic Laws of Propositions B.9.2.4 $\rangle \equiv$

$$
\begin{array}{lll}
[\![\ true\_in & : true & \\
;\ false\_out & : [\ p\ ?\ prop\ \vdash [false \vdash p\ ]] & \\
;\ imp & : [p, q\ ?\ prop & \\
 & \ \vdash \langle\!|\ in\ \ := [[p \vdash q\ ] \vdash p \Rightarrow q\ ] & \\
 & \quad ,\ out\ := [p \Rightarrow q \vdash [p \vdash q\ ]] & \\
 & \quad |\!\rangle & \\
 & \ ] & \\
;\ and & : [p, q\ ?\ prop & \\
 & \ \vdash \langle\!|\ in\ \ := [\langle\!|\ p\ ,\ q\ |\!\rangle \vdash p \wedge q\ ] & \\
 & \quad ,\ out\ := [\ p \wedge q \vdash \langle\!|\ p\ ,\ q\ |\!\rangle] & \\
 & \quad |\!\rangle & \\
 & \ ] & \\
;\ or & : [p, q, r\ ?\ prop & \\
 & \ \vdash \langle\!|\ in\ \ := [\langle\!|[p \vdash r\ ], [q \vdash r\ ]|\!\rangle \vdash [\ p \vee q \vdash r\ ]] & \\
 & \quad ,\ out\ := [[\ p \vee q \vdash r\ ] \vdash \langle\!|[p \vdash r\ ], [q \vdash r\ ]|\!\rangle] & \\
 & \quad |\!\rangle & \\
 & \ ] & \\
;\ equiv & : [p, q\ ?\ prop & \\
 & \ \vdash \langle\!|\ in\ \ := [[p \models q\ ] \vdash p \Leftrightarrow q\ ] & \\
 & \quad ,\ out\ := [p \Leftrightarrow q \vdash [p \models q\ ]] & \\
 & \quad |\!\rangle & \\
 & \ ] & \\
;\ not & : [\ p\ \ ?\ prop & \\
 & \ \vdash \langle\!|\ in\ \ := [[p \vdash false\ ] \vdash \neg\ p\ ] & \\
 & \quad ,\ out\ := [\neg p \vdash [p \vdash false\ ]] & \\
 & \quad |\!\rangle & \\
 & \ ] &
\end{array}
$$

$; \ nequiv \quad : [p, q \ ? \ prop$

$\vdash \langle\!| \ in \quad := [\neg p \Leftrightarrow q \vdash p \not\Leftrightarrow q]$

$, \ out \ := [\neg p \not\Leftrightarrow q \vdash p \not\Leftrightarrow q]$

$|\!\rangle$

$]$

$; \ tnd \quad : [\ p \ ? \ prop \ \vdash p \vee \neg \ p \ ]$

$]\!]$

B.9.2.5.

$\langle$ Derived Laws of Propositions B.9.2.5 $\rangle \equiv$

$[\![$ $\langle$ Implication B.9.2.6 $\rangle$

; $\langle$ Conjunction B.9.2.7 $\rangle$

; $\langle$ Disjunction B.9.2.8 $\rangle$

; $\langle$ Logical Equivalence B.9.2.9 $\rangle$

; $\langle$ Miscalleneous B.9.2.10 $\rangle$

; $\langle$ Inequality B.9.2.11 $\rangle$

$]\!]$

B.9.2.6.

$\langle$ Implication B.9.2.6 $\rangle \equiv$

$[\![\ any\_imp\_true \ : [\ p \ ? \ prop \ \vdash [p \vdash true\,]]$

$; \ refl\_imp \quad : [\ p \ ? \ prop \ \vdash p \Rightarrow p\,]$

$; \ sym\_imp \quad : [p, q \ ? \ prop \ \vdash [p \Rightarrow q \vdash q \Rightarrow p\,]]$

$; \ trans\_imp \quad : [p, q, r \ ? \ prop \ \vdash [p \Rightarrow q; q \Rightarrow r \vdash p \Rightarrow r\,]]$

$; \ contra \quad : [p, q \ ? \ prop \ \vdash [p \Rightarrow q \vdash \neg q \Rightarrow \neg \ p \ ]]$

$]\!]$

B.9.2.7.

$\langle$ Conjunction B.9.2.7 $\rangle \equiv$

$[\![\ pLeft \quad : [p, q \ ? \ prop \ \vdash [\ p \wedge q \vdash p\,]]$

$; \ pRight \quad : [p, q \ ? \ prop \ \vdash [\ p \wedge q \vdash q\,]]$

$; \ and3 \quad : [p, q, r \ ? \ prop \ \vdash [\langle\!| \ p \ , \ q \ , \ r \ |\!\rangle \vdash p \wedge q \wedge r\,]]$

$; \ and4 \quad : [p, q, r, s \ ? \ prop \ \vdash [\langle\!| \ p \ , \ q \ , \ r \ , \ s \ |\!\rangle \vdash p \wedge q \wedge r \wedge s\,]]$

$; \ simp\_andR \ : [\ p \ ? \ prop \ \vdash p \wedge true \Leftrightarrow p\,]$

$]\!]$

B.9.2.8.

$\langle$ Disjunction B.9.2.8 $\rangle \equiv$

$[\![\ iLeft \quad : [p, q \ ? \ prop \ \vdash [p \vdash p \vee q\,]]$

$; \ iRight \ : [p, q \ ? \ prop \ \vdash [p \vdash q \vee p\,]]$

$$; \ \mathit{cased} \ : [p, q, r ? \ \mathit{prop} \vdash \frac{p \vee q; \quad \langle\!|[p \vdash r],[q \vdash r]|\!\rangle}{r}]$$

$\rrbracket$

B.9.2.9.

$\langle$ Logical Equivalence B.9.2.9 $\rangle \equiv$

$\llbracket \ \mathit{equiv\_prop} : [p, q ? \ \mathit{prop}$
$\quad \vdash \langle\!| \ \mathit{decomp} \ := [p \Leftrightarrow q \vdash p \Rightarrow q \wedge q \Rightarrow p]$
$\quad , \mathit{comp} \ := [p \Rightarrow q \wedge q \Rightarrow p \vdash p \Leftrightarrow q]$
$\quad |\!\rangle$
$\quad]$
$; \ \mathit{prefl} \ : [\, p ? \mathit{prop} \vdash p \Leftrightarrow p \,]$
$; \ \mathit{psym} \ : [p, q ? \ \mathit{prop} \vdash [p \Leftrightarrow q \vdash q \Leftrightarrow p]]$
$; \ \mathit{ptrans} \ : [p, q, r ? \ \mathit{prop} \vdash [p \Leftrightarrow q; q \Leftrightarrow r \vdash p \Leftrightarrow r]]$
$; \ \mathit{valid} \ : [\, p ? \mathit{prop} \vdash [p \Leftrightarrow \mathit{true} \models p]]$
$; \ \mathit{psubst} \ : [q, r ? \ \mathit{prop} \, ; P ? [\mathit{prop} \vdash \mathit{prop}] \vdash [q \Leftrightarrow r; P(q) \vdash P(r)]]$
$; \ \mathit{prsubst} \ : [q, r ? \ \mathit{prop} \, ; P ? [\mathit{prop} \vdash \mathit{prop}] \vdash [q \Leftrightarrow r; P(r) \vdash P(q)]]$
$\rrbracket$

B.9.2.10.

$\langle$ Miscalleneous B.9.2.10 $\rangle \equiv$

$\llbracket \ \mathit{true\_is\_true} \ : \ \mathit{true}$
$; \ \mathit{sym\_not\_equiv} \ : [p, q ? \ \mathit{prop} \vdash [p \not\Leftrightarrow q \vdash q \not\Leftrightarrow p]]$
$\rrbracket$

B.9.2.11.

$\langle$ Inequality B.9.2.11 $\rangle \equiv$

$\llbracket \ (\cdot) \neq (\cdot) \ := [\, s ? \mathit{sort} \, ; a, b : \ s \vdash \neg a = b \,]$
; **opspec left** $\neq \dot{=}=$
$; \ \mathit{sym\_neq} : \ [\, s ? \mathit{sort} \, ; a, b ? \ s \vdash [a \neq b \vdash b \neq a]]$
$\rrbracket$

B.9.3 Ordered pairs

$\langle$ Ordered Pairs B.9.3 $\rangle \equiv$

context *OrderedPairs* :=
$\llbracket \langle$ Construction and Selection Operations B.9.3.1 $\rangle$
; $\langle$ Laws of Ordered Pairs B.9.3.2 $\rangle$
$\rrbracket$

B.9.3.1.

$\langle$ Construction and Selection Operations B.9.3.1 $\rangle \equiv$

$[\![\ (\cdot) \otimes (\cdot) \quad : [sort; sort \vdash sort\,]$

; **opspec left** $\otimes$

; $(\cdot) \mapsto (\cdot)\ : [s\_1, s\_2\ ?\ \ sort\ ; s\_1; s\_2 \vdash s\_1 \otimes s\_2\,]$

; **opspec left** $\mapsto$

; **opspec** $\mapsto\ \gg\ =$

; $sel_1 \qquad : [s\_1, s\_2\ ?\ \ sort\ ; s\_1 \otimes s\_2 \vdash s\_1\,]$

; $sel_2 \qquad : [s\_1, s\_2\ ?\ \ sort\ ; s\_1 \otimes s\_2 \vdash s\_2\,]$

$]\!]$

B.9.3.2.

$\langle$ Laws of Ordered Pairs B.9.3.2 $\rangle \equiv$

$[\![\ pair\_inj\ : [s\_1, s\_2\ ?\ \ sort\ ; a, c\ ?\ \ s\_1\ ; b, d\ ?\ \ s\_2$

$\qquad\qquad \vdash [a \mapsto b = c \mapsto d \models a = c \wedge b = d\,]$

$\qquad\qquad]$

; $def\_sel_1\ \ : [s\_1, s\_2\ ?\ \ sort\ ; a\ ?\ \ s\_1\ ; b\ ?\ \ s\_2\ \vdash sel_1(a \mapsto b) = a\,]$

; $def_s\,el_2\ \ : [s\_1, s\_2\ ?\ \ sort\ ; a\ ?\ \ s\_1\ ; b\ ?\ \ s\_2\ \vdash sel_2(a \mapsto b) = b\,]$

$]\!]$

B.9.4 Quantifiers

$\langle$ Quantifiers B.9.4 $\rangle \equiv$

context *Quantifiers* $:=$

$[\![\langle$ Basic Quantifiers B.9.4.1 $\rangle$

; $\langle$ Basic Laws of Quantifiers B.9.4.2 $\rangle$

; $\langle$ Defined Quantifiers B.9.4.3 $\rangle$

; $\langle$ Derived Laws of Quantifiers B.9.4.4 $\rangle$

$]\!]$

B.9.4.1.

$\langle$ Basic Quantifiers B.9.4.1 $\rangle \equiv$

$[\![\forall\,(\cdot),$

$\ \exists\,(\cdot)\ \ : [\ s\ ?\ sort\ \vdash [[s \vdash prop\,] \vdash prop\,]]$

; **opspec** $\forall \gg =$

; **opspec** $\exists \gg =$

$]\!]$

B.9.4.2.

$\langle$ Basic Laws of Quantifiers B.9.4.2 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathit{univ}\ : [\, s\ ?\ \mathit{sort}\ ;\ P\ \ ?\, [s \vdash \mathit{prop}\,] \\
\qquad\qquad \vdash \langle\!| \ \mathit{in} \quad := [[\, x : s\ \vdash P(x)\,] \vdash \forall\ P\ ] \\
\qquad\qquad\quad ,\ \mathit{out}\ := [\forall P \vdash [\, x : s\ \vdash P(x)\,]\,] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
;\ \mathit{ex} \quad : [\, s\ ?\ \mathit{sort}\ ;\ P\ : [s \vdash \mathit{prop}\,];\ q\ ?\ \mathit{prop} \\
\qquad\qquad \vdash \langle\!| \ \mathit{in} \quad := [[\, x : s\ \vdash [\ P\ (x) \vdash q\,]\,] \vdash [\exists P \vdash q\,]\,] \\
\qquad\qquad\quad ,\ \mathit{out}\ := [[\exists P \vdash q\,] \vdash [\ x : s\ \vdash [\ P\ (x) \vdash q\,]\,]\,] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.4.3.

$\langle$ Defined Quantifiers B.9.4.3 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \forall_2\ (\cdot) & := [s\_1, s\_2\,?\ \mathit{sort} \\
& \quad \vdash [\ P\ \ : [s\_1; s\_2 \vdash \mathit{prop}\,] \\
& \qquad \vdash \forall [\ x : s\_1\ \vdash \forall\ P\ (x)\,] \\
& \qquad] \\
& \quad] \\
;\ \exists_2\ (\cdot) & := [s\_1, s\_2\,?\ \mathit{sort} \\
& \quad \vdash [\ P\ \ : [s\_1; s\_2 \vdash \mathit{prop}\,] \\
& \qquad \vdash \exists [\ x : s\_1\ \vdash \exists\ P\ (x)\,] \\
& \qquad] \\
& \quad] \\
;\ \mathbf{opspec}\ \ \forall_2 \gtrdot = & \\
;\ \mathbf{opspec}\ \ \exists_2 \gtrdot = & \\
;\ \mathit{exists\_pr}\ (\cdot) : & [s\_1, s\_2\,?\ \mathit{sort}\ \vdash [[\langle\!|\ s\_1\ , s\_2\ |\!\rangle \vdash \mathit{prop}\,] \vdash \mathit{prop}\,]\,] \\
]\!] &
\end{array}
$$

B.9.4.4.

$\langle$ Derived Laws of Quantifiers B.9.4.4 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \mathit{univ\_imp} & : [\ s\ ?\ \mathit{sort}\ ;\ P\ \ ?\, [s \vdash \mathit{prop}\,] \\
& \quad \vdash \langle\!|\ \mathit{in} \quad := [[\ x\ ?\ s\ \vdash P(x)\,] \vdash \forall\ P\ ] \\
& \qquad ,\ \mathit{out}\ := [\forall P \vdash [\ x\ ?\ s\ \vdash P(x)\,]\,] \\
& \qquad |\!\rangle \\
& \quad] \\
;\ \mathit{hide} & : [\ s\ \ ?\ \mathit{sort} \\
& \quad ;\ x\ \ ?\ s \\
& \quad ;\ P\ \ ?\, [s \vdash \mathit{prop}\,] \\
& \quad \vdash [\ P\ (x) \vdash \exists\ P\ ] \\
& \quad]
\end{array}
$$

$$
\begin{array}{ll}
; \; ex2\_eq\_intro & : [s\_1, s\_2 \,?\, sort\, ; \; a \;?\; s\_1\, ; b, c \,?\, s\_2 \\
& \vdash [b = c \\
& \quad \vdash \exists_2 [s\_1; x : \; s\_2 \vdash x = c\,] \\
& \quad] \\
&] \\
; \; ex2\_eq2\_intro & : [s\_1, s\_2 \,?\, sort\, ; a, b \,?\, s\_1\, ; c, d \,?\, s\_2 \\
& \vdash [a = b \wedge c = d \\
& \quad \vdash \exists_2 [\, x : s\_1\, ; y : \; s\_2 \vdash x = b \wedge y = d\,] \\
& \quad] \\
&] \\
]\!]
\end{array}
$$

B.9.5 Natural numbers

$\langle$ Natural Numbers B.9.5 $\rangle \equiv$

context $\mathit{NaturalNumbers} :=$

$[\![\; nat \; : \; sort$

$; \langle$ Peano Axioms B.9.5.1 $\rangle$

$; \langle$ Distinguished Numbers B.9.5.2 $\rangle$

$; \langle$ Basic Arithmetical Operations B.9.5.3 $\rangle$

$; \langle$ Basic Relations B.9.5.6 $\rangle$

$; \langle$ Derived Arithmetical Laws B.9.5.7 $\rangle$

$]\!]$

B.9.5.1.

$\langle$ Peano Axioms B.9.5.1 $\rangle \equiv$

$$
\begin{array}{ll}
[\![0 & : \; nat \\
; \; succ & : [nat \vdash nat\,] \\
; \; succ\_new & : [\; n \,?\, nat \;\vdash \neg\, succ\,(n) = 0\,] \\
; \; succ\_one\_to\_one & : [n, m \,?\; nat\, ; succ(n) = succ(m) \vdash n = m\,] \\
; \; nat\_induction & : [\; P \;\; ? \,[nat \vdash prop\,] \\
& \quad \vdash \left| \begin{array}{l} P\,(0); \\ [\; n \,?\, nat\, ; P(n) \vdash P(succ(n))\,] \\ \hline [\; n \,?\, nat \;\vdash P(n)\,] \end{array} \right. \\
& \quad] \\
]\!]
\end{array}
$$

B.9.5.2.

$\langle$ Distinguished Numbers B.9.5.2 $\rangle \equiv$

$[\![1 := \; succ\,(0)$

$; 2 := \; succ\,(1)$

$; 3 := \; succ\,(2)$

$; 4 := \; succ\,(3)$

$]\!]$

B.9.5.3.

$\langle$ Basic Arithmetical Operations B.9.5.3 $\rangle \equiv$
$[\![\langle$ Signatures and Precedences B.9.5.4 $\rangle$
$; \langle$ Defining Axioms B.9.5.5 $\rangle$
$]\!]$

B.9.5.4.

$\langle$ Signatures and Precedences B.9.5.4 $\rangle \equiv$
$[\![\ (\cdot) + (\cdot), (\cdot) \text{-} (\cdot), (\cdot) \times (\cdot) \ : [\, nat ; nat \vdash nat \,]$
; **opspec** **left** $+$
; **opspec** **left** $\times$
; **opspec** $+ \gtrdot =$
; **opspec** **left** $\text{-} \doteq +$
; **opspec** $\times \gtrdot +$
$]\!]$

B.9.5.5.

$\langle$ Defining Axioms B.9.5.5 $\rangle \equiv$
$[\![\ add \ \ : [\ n \ ? \ nat$
$\qquad \vdash \langle\!| \ base \ \ := 0 + n = n$
$\qquad\quad , \ recur \ := [\ m \ ? \ nat \ \vdash succ(n) + m = succ(n + m) \,]$
$\qquad\quad |\!\rangle$
$\qquad]$
$; \ sub \ \ : [\ n \ ? \ nat$
$\qquad \vdash \langle\!| \ base \ \ := \ n \text{-} 0 = n$
$\qquad\quad , \ recur \ := [\, m, k \ ? \ nat \ \vdash [\ n \text{-} m = k \vdash succ(n) \text{-} succ(m) = k \,]\,]$
$\qquad\quad |\!\rangle$
$\qquad]$
$; \ mult \ : [\ n \ ? \ nat$
$\qquad \vdash \langle\!| \ base \ \ := 0 \times n = 0$
$\qquad\quad , \ recur \ := [\ m \ ? \ nat \ \vdash succ(m) \times n = m \times n + n \,]$
$\qquad\quad |\!\rangle$
$\qquad]$
$]\!]$

B.9.5.6.

$\langle$ Basic Relations B.9.5.6 $\rangle \equiv$
$[\![\ (\cdot) \text{¡} (\cdot) \quad : \quad [\, nat ; nat \vdash prop \,]$
; **opspec** **left** ¡ $\doteq =$

; $\mathit{less\_than}$: $[\, n \;?\; nat$

$\vdash \langle\!| \; pos \;:=\; 0$ ¡ $succ(n)$

$, \; neg \;:=\; \neg\, n$ ¡0

$, \; recur \;:=\; [\, m \,?\, nat \;\vdash\; [succ(n)$ ¡ $succ(m) \models n$ ¡ $m\,]\,]$

$|\!\rangle$

$]$

; $(\cdot)$ ¿ $(\cdot)$ $:= [n, m :\; nat \;\vdash\; m$ ¡ $n\,]$

; $(\cdot) \leq (\cdot)$ $:= [n, m :\; nat \;\vdash \neg\, n$ ¿ $m\,]$

; $(\cdot) \geq (\cdot)$ $:= [n, m :\; nat \;\vdash \neg\, n$ ¡ $m\,]$

; **opspec left** ¿ $\doteq\!=$

; **opspec left** $\leq \doteq\!=$

; **opspec left** $\geq \doteq\!=$

$]\!]$

B.9.5.7.

$\langle$ Derived Arithmetical Laws B.9.5.7 $\rangle \equiv$

$[\![$ $\mathit{gth\_prop}$: $[n, m \;?\; nat \;\vdash [\, n$ ¿$m \vdash n \geq succ(m)\,]\,]$

; $\mathit{geq\_prop}$: $[n, m \;?\; nat \;\vdash [n \geq succ(m) \vdash n \geq m\,]\,]$

$]\!]$

B.9.6 Finite sets

$\langle$ Finite Sets B.9.6 $\rangle \equiv$

context $\mathit{FiniteSets} :=$

$[\![\langle$ Set Construction B.9.6.1 $\rangle$

; $\langle$ Basic Laws of Finite Sets B.9.6.2 $\rangle$

; $\langle$ Operations upon Sets B.9.6.3 $\rangle$

; $\langle$ Derived Laws of Finite Sets B.9.6.14 $\rangle$

$]\!]$

B.9.6.1.

$\langle$ Set Construction B.9.6.1 $\rangle \equiv$

$[\![$ set : $[sort \vdash sort\,]$

; $\{\,\}$: $[\, s \,?\, sort \;\vdash\; set(s)\,]$

; $(\cdot) \odot (\cdot)$: $[\, s \,?\, sort\,; s; set(s) \vdash set(s)\,]$

; $\{(\cdot)\}$ $:= [\, s \,?\, sort\,; a :\; s \;\vdash\; a \odot \{\,\}\,]$

; **opspec right** $\odot$

; **opspec** $\odot\!> =$

; **opspec left** $\odot \doteq\!\mapsto$

$]\!]$

B.9.6.2.

$\langle$ Basic Laws of Finite Sets B.9.6.2 $\rangle \equiv$

$$
\begin{array}{lll}
[\![& \mathit{absorp} & : [\, s \;?\; \mathit{sort} \;;\; a \;?\; s \;;\; x \;?\; \mathit{set}\,(s) \\
& & \;\vdash a \odot a \odot x = a \odot x \\
& & \;] \\
; & \mathit{commut} & : [\, s \;?\; \mathit{sort} \;; a, b \;?\; s \;;\; x \;?\; \mathit{set}\,(s) \\
& & \;\vdash a \odot b \odot x = b \odot a \odot x \\
& & \;] \\
; & \mathit{set\_induction} & : [\, s \;?\; \mathit{sort} \;;\; P \;?\, [\, \mathit{set}\,(s) \vdash \mathit{prop}\,] \\
& & \;\vdash \left| \dfrac{\begin{array}{l} P\,(\{\,\}); \\ [\, a : s \,; x :\; \mathit{set}\,(s) \vdash [\, P\,(x) \vdash P(a \odot x)\,]\,] \end{array}}{[\, x : \mathit{set}\,(s) \vdash P(x)\,]} \right. \\
& & \;] \\
]\!] & &
\end{array}
$$

B.9.6.3.

$\langle$ Operations upon Sets B.9.6.3 $\rangle \equiv$

$[\![$ $\langle$ Set Membership B.9.6.4 $\rangle$

; $\langle$ Union of two Sets B.9.6.5 $\rangle$

; $\langle$ Intersection of two Sets B.9.6.6 $\rangle$

; $\langle$ Difference between two Sets B.9.6.7 $\rangle$

; $\langle$ Cardinality of a Set B.9.6.8 $\rangle$

; $\langle$ Subset Relation B.9.6.9 $\rangle$

; $\langle$ Mapping an Abstraction over a Set B.9.6.10 $\rangle$

; $\langle$ Filtering a Set with a Predicate B.9.6.11 $\rangle$

; $\langle$ Union of a Set of Sets B.9.6.12 $\rangle$

; $\langle$ Intersection of a Set of Sets B.9.6.13 $\rangle$

$]\!]$

B.9.6.4.

$\langle$ Set Membership B.9.6.4 $\rangle \equiv$

$$
\begin{array}{l}
[\![\; (\cdot) \in (\cdot) \;:\; [\, s \;?\; \mathit{sort} \;; s ; \mathit{set}(s) \vdash \mathit{prop}\,] \\
;\; \mathbf{opspec\ left} \;\; \in \doteq = \\
;\; \mathit{member} \;:\; [\, s \;?\; \mathit{sort} \;;\; a \;?\; s \\
\qquad\qquad \vdash \langle\!| \;\mathit{empty} \;:= \neg a \in \{\,\} \\
\qquad\qquad\quad ,\; \mathit{recur} \;\; := [\, b \;?\; s \;;\; x \;?\; \mathit{set}\,(s) \\
\qquad\qquad\qquad\qquad\qquad \vdash a \in b \odot x \Leftrightarrow a = b \vee a \in x \\
\qquad\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
;\; (\cdot) \notin (\cdot) \;:= [\, s \;?\; \mathit{sort} \;; a :\; s \;; x :\; \mathit{set}\,(s) \vdash \neg a \in x\,] \\
;\; \mathbf{opspec\ left} \;\; \notin \doteq = \\
]\!]
\end{array}
$$

B.9.6.5.

$\langle$ Union of two Sets B.9.6.5 $\rangle \equiv$

$[\![\ (\cdot) \cup (\cdot) : [\ s \,?\, sort\ ; set(s); set(s) \vdash set(s)\ ]$

; **opspec left** $\cup$

; **opspec** $\cup \gg =$

; **opspec** $\cup \lessdot \odot$

$$
\begin{array}{ll}
; \ \mathit{union} & : [\ s \ ?\ sort\ ;\ x\ ?\ set\ (s) \\
& \vdash \langle\!| \ base \ \ := \{\,\} \cup x = x \\
& \quad ,\ recur := [\ a\ ?\ s\ ;\ y\ ?\ set\ (s) \\
& \qquad\qquad \vdash a \odot x \cup y = a \odot (x \cup y) \\
& \qquad\qquad] \\
& \quad |\!\rangle \\
&]
\end{array}
$$

$]\!]$

B.9.6.6.

$\langle$ Intersection of two Sets B.9.6.6 $\rangle \equiv$

$[\![\ (\cdot) \cap (\cdot) : [\ s \,?\, sort\ ; set(s); set(s) \vdash set(s)\ ]$

; **opspec left** $\cap$

; **opspec** $\cap \gg \cup$

; **opspec** $\cap \lessdot \odot$

$$
\begin{array}{ll}
; \ \mathit{inter} & : [\ s \ ?\ sort\ ;\ x\ ?\ set\ (s) \\
& \vdash \langle\!| \ base \ \ := \{\,\} \cap x = x \\
& \quad ,\ recur := [\ a\ ?\ s\ ;\ y\ ?\ set\ (s) \\
& \qquad\qquad \vdash \langle\!| \ add \ := [a \in y \vdash a \odot x \cap y = a \odot (x \cap y)] \\
& \qquad\qquad\quad ,\ let \ \ := [a \notin y \vdash a \odot x \cup y = x \cap y] \\
& \qquad\qquad\quad |\!\rangle \\
& \qquad\qquad] \\
& \quad |\!\rangle \\
&]
\end{array}
$$

$]\!]$

B.9.6.7.

$\langle$ Difference between two Sets B.9.6.7 $\rangle \equiv$

$[\![\ (\cdot) \setminus (\cdot) : [\ s \,?\, sort\ ; set(s); set(s) \vdash set(s)\ ]$

; **opspec left** $\setminus$

; **opspec** $\setminus \gg \cap$

; **opspec** $\setminus \lessdot \odot$

$; \ \mathit{diff} \quad : [\, s \ ? \ \mathit{sort} \ ; \ x \ ? \ \mathit{set}\,(s)$

$\quad \vdash \langle\!| \ \mathit{base} \ \ := \{\,\} \setminus x = \{\,\}$

$\quad\quad , \ \mathit{recur} \ := [\, a \ ? \ s \ ; \ y \ ? \ \mathit{set}\,(s)$

$\quad\quad\quad \vdash \langle\!| \ \mathit{add} \ := [a \in y \vdash a \odot x \setminus y = x \setminus y\,]$

$\quad\quad\quad\quad , \ \mathit{let} \ \ := [a \notin y \vdash a \odot x \setminus y = a \odot (x \setminus y)\,]$

$\quad\quad\quad\quad |\!\rangle$

$\quad\quad\quad]$

$\quad\quad |\!\rangle$

$\quad]$

$]\!]$

B.9.6.8.

$\langle$ Cardinality of a Set B.9.6.8 $\rangle \equiv$

$[\![\, \mathbf{card} \quad : [\, s \,?\, \mathit{sort} \,;\, \mathit{set}(s) \vdash \mathit{nat}\,]$

$; \ \mathit{def\_card} \ : [\, s \ ? \ \mathit{sort}$

$\quad \vdash \langle\!| \ \mathit{empty} \ := \mathbf{card}(\{\,\}(s := s)) = 0$

$\quad\quad , \ \mathit{recur} \ \ := [\ a \ ? \ s \ ; \ x \ ? \ \mathit{set}\,(s)$

$\quad\quad\quad \vdash \langle\!| \ \mathit{new} \ := [a \notin x \vdash \mathbf{card}(a \odot x) = \mathit{succ}(\mathbf{card}(x))\,]$

$\quad\quad\quad\quad , \ \mathit{inv} \ \ := [a \in x \vdash \mathbf{card}(a \odot x) = \mathbf{card}(x)\,]$

$\quad\quad\quad\quad |\!\rangle$

$\quad\quad\quad]$

$\quad\quad |\!\rangle$

$\quad]$

$]\!]$

B.9.6.9.

$\langle$ Subset Relation B.9.6.9 $\rangle \equiv$

$[\![\ (\cdot) \subseteq (\cdot) \ : [\, s \,?\, \mathit{sort} \,;\, \mathit{set}(s); \mathit{set}(s) \vdash \mathit{prop}\,]$

$; \ \mathbf{opspec} \ \mathbf{left} \ \subseteq \doteq =$

$; \ \mathit{subset} \quad : [\, s \ ? \ \mathit{sort} \,;\, x, y \,?\ \mathit{set}\,(s)$

$\quad \vdash x \subseteq y = \forall[\ a : s \ \vdash \ a \in x \Rightarrow a \in y\,]$

$\quad]$

$]\!]$

B.9.6.10.

$\langle$ Mapping an Abstraction over a Set B.9.6.10 $\rangle \equiv$

$[\![\ (\cdot) * (\cdot) \ \ : [s\_1, s\_2 \,? \ \mathit{sort} \,;\, [s\_1 \vdash s\_2]; \mathit{set}(s\_1) \vdash \mathit{set}(s\_2)\,]$

$; \ \mathbf{opspec} \ \ \mathbf{right} \ *$

$; \ \mathbf{opspec} \ * > \cap$

$; \ \mathbf{opspec} \ * < \odot$

$$
\begin{array}{llll}
; & \mathit{setmap} & : [s\_1, s\_2 \ ?\ \mathit{sort}\ ;\ F\ \ ?\,[s\_1 \vdash s\_2] & \\
 & & \vdash \langle\!|\ \mathit{empty}\ := \ F\, *\{\,\} = \{\,\} & \\
 & & \quad ,\ \mathit{cons}\quad := [\ a\ \ ?\ \ s\_1\ ;\ x\ \ ?\ \ \mathit{set}\ (s\_1) & \\
 & & \qquad\qquad\qquad \vdash F * a \odot x = F(a) \odot (F * x) & \\
 & & \qquad\qquad\qquad] & \\
 & & \quad |\!\rangle & \\
 & &] & \\
\rrbracket & & &
\end{array}
$$

B.9.6.11.

$\langle$ Filtering a Set with a Predicate B.9.6.11 $\rangle \equiv$

$$
\begin{array}{lll}
\llbracket & (\cdot) \rhd (\cdot) & : [\, s \ ?\ \mathit{sort}\ ; [s \vdash \mathit{prop}\,]; \mathit{set}(s) \vdash \mathit{set}(s)\,] \\
; & \textbf{opspec right} & \rhd \doteq * \\
; & \mathit{filter} & : [\ s\ \ ?\ \ \mathit{sort}\ ;\ \ P\ \ ?\,[s \vdash \mathit{prop}\,] \\
 & & \ \vdash \langle\!|\ \mathit{empty}\ := P \rhd \{\,\} = \{\,\} \\
 & & \quad ,\ \mathit{cons}\quad := [\ a\ \ ?\ \ s\ ;\ x\ \ ?\ \ \mathit{set}\ (s) \\
 & & \qquad\qquad\qquad \vdash \langle\!|\ \mathit{yes}\ := [\ P\ (a) \vdash P \rhd a \odot x = a \odot (P \rhd x)\,] \\
 & & \qquad\qquad\qquad\quad ,\ \mathit{no}\ \ := [\neg\, P\ (a) \vdash P \rhd a \odot x = P \rhd x\,] \\
 & & \qquad\qquad\qquad\quad |\!\rangle \\
 & & \qquad\qquad\qquad] \\
 & & \quad |\!\rangle \\
 & &] \\
\rrbracket & &
\end{array}
$$

B.9.6.12.

$\langle$ Union of a Set of Sets B.9.6.12 $\rangle \equiv$

$$
\begin{array}{lll}
\llbracket & \bigcup(\cdot) & : [\, s \ ?\ \mathit{sort}\ ; \mathit{set}(\mathit{set}(s)) \vdash \mathit{set}(s)\,] \\
; & \textbf{opspec}\ \bigcup > = & \\
; & \mathit{def\_bigunion} & : [\ s\ \ ?\ \ \mathit{sort} \\
 & & \ \vdash \langle\!|\ \mathit{empty}\ := \bigcup\{\,\} = (\{\,\} \therefore \mathit{set}(s)) \\
 & & \quad ,\ \mathit{recur}\quad := [\ x\ \ ?\ \ \mathit{set}\ (s);\ \ xx\ \ ?\ \ \mathit{set}\ (\mathit{set}(s)) \\
 & & \qquad\qquad\qquad \vdash \bigcup(x \odot xx) = x \cup \bigcup xx \\
 & & \qquad\qquad\qquad] \\
 & & \quad |\!\rangle \\
 & &] \\
\rrbracket & &
\end{array}
$$

B.9.6.13.

$\langle$ Intersection of a Set of Sets B.9.6.13 $\rangle \equiv$

$$
\begin{array}{lll}
\llbracket & \bigcap(\cdot) & : [\, s \ ?\ \mathit{sort}\ ; \mathit{set}(\mathit{set}(s)) \vdash \mathit{set}(s)\,] \\
; & \textbf{opspec}\ \bigcap > = &
\end{array}
$$

$$
\begin{array}{ll}
;\ \mathit{def\_biginter} & :[\, s\ ?\ \mathit{sort} \\
& \vdash \langle\!|\ \mathit{empty}\ := \bigcap\{\,\} = (\{\,\} \therefore \mathit{set}(s)) \\
& \quad ,\ \mathit{recur}\ \ := [\, x\ ?\ \mathit{set}\,(s);\ xx\ ?\ \mathit{set}\,(\mathit{set}(s)) \\
& \qquad\qquad\qquad \vdash \bigcap(x \odot xx) = x \cap \bigcap xx \\
& \qquad\qquad\qquad] \\
& \quad |\!\rangle \\
&] \\
]\!] &
\end{array}
$$

B.9.6.14.

$\langle$ Derived Laws of Finite Sets B.9.6.14 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \mathit{sing} & :[\, s\ ?\ \mathit{sort}\ ;\ a\ ?\ s \\
& \vdash \langle\!|\, \mathbf{card}\ := \mathbf{card}(\{a\}) = 1 \\
& \quad ,\ \mathit{map}\ := [\, t\ ?\ \mathit{sort}\ ;\ F\ ?\ [s \vdash t\,] \vdash F * \{a\} = \{F(a)\}\,] \\
& \quad |\!\rangle \\
&] \\
;\ \mathit{observ} & :[\, s\ ?\ \mathit{sort}\ ; x, y\ ?\ \mathit{set}\,(s) \\
& \vdash x = y \Leftrightarrow \forall[\, a : s\ \vdash a \in x \Leftrightarrow a \in y\,] \\
&] \\
;\ \mathit{subset\_prop} & :[\, s\ ?\ \mathit{sort}\ ; x, y\ ?\ \mathit{set}\,(s) \\
& \vdash \langle\!|\ \mathit{mutual}\ := \left| \begin{array}{l} x \subseteq y; \\ y \subseteq x \\ \hline x = y \end{array} \right. \\
& \quad ,\ \mathit{trans}\ \ := [\, z\ ?\ \mathit{set}\,(s) \\
& \qquad\qquad \vdash \left| \begin{array}{l} x \subseteq y; \\ \mathit{snd}\quad : y \subseteq z \\ \hline x \subseteq z \end{array} \right. \\
& \qquad\qquad] \\
& \quad ,\ \mathit{weaken}\ := [\, a\ ?\ s\ \vdash \left| \begin{array}{l} x \subseteq y; \\ a \in x \\ \hline a \in y \end{array} \right.] \\
& \quad ,\ \mathit{extend}\ := [\, a\ ?\ s\ \vdash \left| \begin{array}{l} x \subseteq y; \\ \mathit{new}\quad : a \in y \\ \hline a \odot x \subseteq y \end{array} \right.] \\
& \quad |\!\rangle \\
&] \\
;\ \mathit{subset\_empty} & :[\, s\ ?\ \mathit{sort}\ ;\ x\ ?\ \mathit{set}\,(s) \\
& \vdash \{\,\} \subseteq x \\
&]
\end{array}
$$

$;\ card\_prop\ \ :[\,s\ ?\ sort\,; x, y\ ?\ set\,(s);\ m\ ?\ nat$

$$\vdash \begin{array}{|l} m \geq \mathbf{card}(y); \\ x \subseteq y \\ \hline m \geq \mathbf{card}(x) \end{array}$$

$]$

$;\ member\_prop\ :[\,s\ ?\ sort\,;\ a\ ?\ s$

$\vdash\langle\!|\ single\ := [\,b\,?\,s\ \vdash a \in \{b\} \Leftrightarrow a = b\,]$

$,\ cons\ := [\ b\ ?\ s$

$;\ x\ ?\ set\,(s)$

$\vdash a \in b \odot x \Leftrightarrow a = b \vee (a \neq b \wedge a \in x)$

$]$

$,\ filter\ := [\,x\,?\,set\,(s); P\,?\,[s \vdash prop\,] \vdash a \in P \rhd x \Leftrightarrow a \in x \wedge P(a)\,]$

$,\ union\ := [x, y\,?\ set\,(s) \vdash a \in x \cup y \Leftrightarrow a \in x \vee a \in y\,]$

$|\!\rangle$

$]$

$;\ pair\_prop\ \ :[s\_1, s\_2\,?\ sort\,;\ a\ ?\ s\_1\,;\ b\ ?\ s\_2\,;\ x\ ?\ set\,(s\_1 \otimes s\_2)$

$\vdash\langle\!|\,sel_1 := [a \mapsto b \in x \vdash a \in sel_1 * x\,]$

$,sel_2 := [a \mapsto b \in x \vdash b \in sel_2 * x\,]$

$|\!\rangle$

$]$

$]\!]$

B.9.7 Sequences

$\langle$ Sequences B.9.7 $\rangle \equiv$

context *Sequences* :=

$[\![$ $\langle$ Sequence Construction B.9.7.1 $\rangle$

; $\langle$ Basic Laws of Sequences B.9.7.2 $\rangle$

; $\langle$ Operations upon Sequences B.9.7.3 $\rangle$

; $\langle$ Derived Laws of Sequences B.9.7.16 $\rangle$

$]\!]$

B.9.7.1.

$\langle$ Sequence Construction B.9.7.1 $\rangle \equiv$

$[\![\ seq\ \ :\ \ [sort \vdash sort\,]$

$;\ \langle\rangle\ \ :\ \ [\,s\,?\,sort\ \vdash seq(s)\,]$

$;\ (\cdot) \odot (\cdot)\ :\ \ [\,s\,?\,sort\,; s; seq(s) \vdash seq(s)\,]$

$;\ \langle\ (\cdot)\ \rangle\ \ := [\,s\,?\,sort\,; a :\ s\ \vdash a \odot \langle\rangle\,]$

; **opspec** **right** $\odot$

; **opspec** $\odot >=$

; **opspec** **left** $\odot \doteq\!\mapsto$

$]\!]$

B.9.7.2.

$\langle$ Basic Laws of Sequences B.9.7.2 $\rangle \equiv$

$$
\begin{array}{lll}
[\![& cons\_inj & : [\, s\ ?\ sort\, ; a, b\ ?\ s\, ; l\_1, l\_2\ ?\ seq\,(s) \\
& & \vdash [\, a \odot l\_1 = b \odot l\_2 \vdash a = b \wedge l\_1 = l\_2\,] \\
& &] \\
; & seq\_induction & : [\, s\ ?\ sort\, ;\ P\ ?\, [\, seq\,(s) \vdash prop\,] \\
& & \vdash \left| \dfrac{P\,(\langle\rangle); \quad [\, a : s\, ; l :\ seq\,(s) \vdash [\, P\,(l) \vdash P(a \odot l)\,]\,]}{[\, l : seq\,(s) \vdash P(l)\,]} \right. \\
& &] \\
]\!] & &
\end{array}
$$

B.9.7.3.

$\langle$ Operations upon Sequences B.9.7.3 $\rangle \equiv$

$[\![$ $\langle$ Sequential Join of two Sequences B.9.7.4 $\rangle$
; $\langle$ Head and Tail B.9.7.5 $\rangle$
; $\langle$ Length of a Sequence B.9.7.6 $\rangle$
; $\langle$ Mapping an Abstraction over a Sequence B.9.7.7 $\rangle$
; $\langle$ Filtering a Sequence with a Predicate B.9.7.8 $\rangle$
; $\langle$ Flattening a Sequence of Sequences B.9.7.9 $\rangle$
; $\langle$ Set of Elements occuring in a Sequence B.9.7.10 $\rangle$
; $\langle$ Accessing an Element in a Sequence B.9.7.11 $\rangle$
; $\langle$ Extracting a Specified Subsequence B.9.7.12 $\rangle$
; $\langle$ Cut and Paste of a Specified Subsequence B.9.7.13 $\rangle$
; $\langle$ Generating a Sequence of Ascending Numbers B.9.7.14 $\rangle$
; $\langle$ First Position of an Element in a Sequence B.9.7.15 $\rangle$
$]\!]$

B.9.7.4.

$\langle$ Sequential Join of two Sequences B.9.7.4 $\rangle \equiv$

$$
\begin{array}{ll}
[\![& (\cdot) +\!\!+ (\cdot)\ : [\, s\ ?\ sort\, ; seq(s); seq(s) \vdash seq(s)\,] \\
; & \mathbf{opspec}\ +\!\!+ \gg = \\
; & \mathbf{opspec}\ +\!\!+ \ll \odot \\
; & join\ : [\, s\ ?\ sort\, ;\ l\_1\ ?\ seq\,(s) \\
& \quad \vdash \langle\!| \ empty\ := \langle\rangle +\!\!+ l\_1 = l\_1 \\
& \qquad ,\ recur\ := [\, a\ ?\ s\, ;\ l\_2\ ?\ seq\,(s) \\
& \qquad\qquad \vdash a \odot l\_2 +\!\!+ l\_1 = a \odot (l\_2 +\!\!+ l\_1) \\
& \qquad\qquad] \\
& \qquad |\!\rangle \\
& \quad] \\
]\!] &
\end{array}
$$

B.9.7.5.

$\langle$ Head and Tail B.9.7.5 $\rangle \equiv$
$[\![\mathrm{hd}\,(\cdot) \;: [\, s\ ?\ sort\ ; seq(s) \vdash s\,]$
$;\ \mathrm{tl}\ (\cdot) : [\, s\ ?\ sort\ ; seq(s) \vdash seq(s)\,]$
; **opspec** $\mathrm{hd} \gtrdot =$
; **opspec** $\mathrm{tl} \gtrdot =$
$;\ head\ : [\, s\ ?\ sort\ ; a\ ?\ s\ ; l\ ?\ seq\ (s) \vdash \mathrm{hd}\ a \odot l = a\,]$
$;\ tail\ : [\, s\ ?\ sort\ ; a\ ?\ s\ ; l\ ?\ seq\ (s) \vdash \mathrm{tl}\ a \odot l = l\,]$
$]\!]$

B.9.7.6.

$\langle$ Length of a Sequence B.9.7.6 $\rangle \equiv$
$[\![\mathrm{len}\,(\cdot) \;: [\, s\ ?\ sort\ ; seq(s) \vdash nat\,]$
; **opspec** $\mathrm{len} \lessdot +\!\!+$
; **opspec** $\mathrm{len} \gtrdot \times$
$;\ length\ : [\, s\ ?\ sort$
$\quad \vdash \langle\!| \ empty\ := \mathrm{len}(\langle\rangle \therefore seq(s)) = 0$
$\quad\ ,\ cons\ := [\, a\ ?\ s\ ; l\ ?\ seq\ (s) \vdash \mathrm{len}\ a \odot l = succ(\mathrm{len} l)\,]$
$\quad |\!\rangle$
$\quad]$
$]\!]$

B.9.7.7.

$\langle$ Mapping an Abstraction over a Sequence B.9.7.7 $\rangle \equiv$
$[\![\ (\cdot) * (\cdot) \;: [s, t\ ?\ sort\ ; [s \vdash t]; seq(s) \vdash seq(t)\,]$
; **opspec** $* \gtrdot +\!\!+$
; **opspec** $* \lessdot \odot$
$;\ seqmap\ : [s, t\ ?\ sort\ ;\ F\ ?\ [s \vdash t\,]$
$\quad \vdash \langle\!| \ empty\ :=\ F * \langle\rangle = \langle\rangle$
$\quad\ ,\ cons\ := [\, a\ ?\ s\ ;\ l\ ?\ seq\ (s)$
$\qquad \vdash F * a \odot l = F(a) \odot (F * l)$
$\qquad]$
$\quad |\!\rangle$
$\quad]$
$]\!]$

B.9.7.8.

$\langle$ Filtering a Sequence with a Predicate B.9.7.8 $\rangle \equiv$
$[\![\ (\cdot) \rhd (\cdot) \;: [\, s\ ?\ sort\ ; [s \vdash prop\,]; seq(s) \vdash seq(s)\,]$
; **opspec right** $\rhd \doteq *$

$; \ \mathit{seqfilter} \ : [\ s \ ? \ \mathit{sort}$
$\quad ; \ P \ ? \ [s \vdash \mathit{prop}\,]$
$\quad \vdash \langle\!| \ \mathit{empty} \ := P \rhd \langle\rangle = \langle\rangle$
$\quad\quad , \ \mathit{cons} \ \ := [\ a \ ? \ s \ ; \ l \ ? \ \mathit{seq}\,(s)$
$\quad\quad\quad \vdash \langle\!| \ \mathit{true} \ := [\ P\,(a) \vdash P \rhd a \odot l = a \odot (P \rhd l)\,]$
$\quad\quad\quad\quad , \ \mathit{false} \ := [\neg\, P\,(a) \vdash P \rhd a \odot l = P \rhd l\,]$
$\quad\quad\quad\quad |\!\rangle$
$\quad\quad\quad]$
$\quad\quad |\!\rangle$
$\quad]$
$]\!]$

B.9.7.9.

$\langle$ Flattening a Sequence of Sequences B.9.7.9 $\rangle \equiv$
$[\![\mathrm{flatten}\,(\cdot) \ : [\ s \ ? \ \mathit{sort} \ ; \mathit{seq}(\mathit{seq}(s)) \vdash \mathit{seq}(s)\,]$
$; \ \mathbf{opspec} \ \ \mathrm{flatten} > +\!\!+$
$; \ \mathbf{opspec} \ \ \mathrm{flatten} < \odot$
$; \ \mathit{flatten} \ \ : [\ s \ ? \ \mathit{sort}$
$\quad \vdash \langle\!| \ \mathit{empty} \ := \mathrm{flatten}\langle\rangle = (\langle\rangle \therefore \mathit{seq}(s))$
$\quad\quad , \ \mathit{recur} \ \ := [\ l \ ? \ \mathit{seq}\,(s); \ \mathit{ll} \ ? \ \mathit{seq}\,(\mathit{seq}(s))$
$\quad\quad\quad \vdash \mathrm{flatten}\ l \odot \mathit{ll} = l +\!\!+ \mathrm{flatten}\ \mathit{ll}$
$\quad\quad\quad]$
$\quad\quad |\!\rangle$
$\quad]$
$]\!]$

B.9.7.10.

$\langle$ Set of Elements occuring in a Sequence B.9.7.10 $\rangle \equiv$
$[\![\mathrm{elems}\,(\cdot) \ : [\ s \ ? \ \mathit{sort} \ ; \mathit{seq}(s) \vdash \mathit{set}(s)\,]$
$; \ \mathbf{opspec} \ \ \mathrm{elems} < *$
$; \ \mathbf{opspec} \ \ \mathrm{elems} > *$
$; \ \mathit{elems} \ \ : [\ s \ ? \ \mathit{sort}$
$\quad \vdash \langle\!| \ \mathit{empty} \ := \mathrm{elems}(\langle\rangle \therefore \mathit{seq}(s)) = \{\,\}$
$\quad\quad , \ \mathit{recur} \ \ := [\ a \ ? \ s \ ; l \ ? \ \mathit{seq}\,(s) \vdash \mathrm{elems}\ a \odot l = a \odot \mathrm{elems}\ l\ ]$
$\quad\quad |\!\rangle$
$\quad]$
$]\!]$

B.9.7.11.

$\langle$ Accessing an Element in a Sequence B.9.7.11 $\rangle \equiv$
$[\![\ (\cdot)\ \nabla\ (\cdot) \ : [\ s \ ? \ \mathit{sort} \ ; \mathit{seq}(s); \mathit{nat} \vdash s\,]$
$; \ \mathbf{opspec} \ \ \nabla > \odot$

$$
\begin{array}{l}
;\ access\ \ :[\, s\ ?\ sort\,;\ a\ ?\ s\,;\ l\ ?\ seq\,(s) \\
\qquad\qquad \vdash \langle\!\vert\ base\ \ := (a \odot l)\nabla 1 = a \\
\qquad\qquad\quad ,\ recur\ := [\, n\ ?\ nat\,;\, b\ ?\ s\ \vdash [\, l\ \nabla n = a \vdash (b \odot l)\nabla\ succ\,(n) = a\,]\,] \\
\qquad\qquad\quad \vert\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.7.12.

$\langle$ Extracting a Specified Subsequence B.9.7.12 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ subseq\ \ \ : [\, s\ ?\ sort\,;\, nat;\, nat;\, seq(s) \vdash seq(s)\,] \\
;\ def\_subseq\ : [\, s\ ?\ sort\,;\, n, m\ ?\ nat \\
\qquad\qquad \vdash \langle\!\vert\ empty\ := subseq\,(n, m, \langle\rangle \therefore seq(s)) = \langle\rangle \\
\qquad\qquad\quad ,\ single\ := [\, a\ ?\ s \\
\qquad\qquad\qquad\qquad \vdash \langle\!\vert\ subseq\,(1, 1, \langle\ a\ \rangle) = \langle\ a\ \rangle \\
\qquad\qquad\qquad\qquad\quad ,[n \neq 1 \vee m \neq 1 \vdash subseq(n, m, \langle\ a\ \rangle) = \langle\rangle\,] \\
\qquad\qquad\qquad\qquad\quad \vert\!\rangle \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad ,\ recur\ := [l1, l2\ ?\ seq\,(s) \\
\qquad\qquad\qquad\qquad \vdash \langle\!\vert [n \leq \text{len}\, l1 \wedge m \leq \text{len}\, l1 \\
\qquad\qquad\qquad\qquad\quad \vdash subseq\,(n, m, l1 +\!\!+ l2) = subseq(n, m, l1) \\
\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad\qquad\quad ,[n \leq \text{len}\, l1 \wedge m\ \text{¿len}\, l1 \\
\qquad\qquad\qquad\qquad\quad \vdash subseq\,(n, m, l1 +\!\!+ l2) \\
\qquad\qquad\qquad\qquad\qquad = subseq(n, \text{len}l1, l1) +\!\!+ subseq\,(1, m - \text{len}l1, l2) \\
\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad\qquad\quad ,[\, n\ \text{¿len}\, l1 \\
\qquad\qquad\qquad\qquad\quad \vdash subseq\,(n, m, l1 +\!\!+ l2) \\
\qquad\qquad\qquad\qquad\qquad = subseq(n - \text{len}l1, m - \text{len}l1, l2) \\
\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad\qquad\quad \vert\!\rangle \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad \vert\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.7.13.

$\langle$ Cut and Paste of a Specified Subsequence B.9.7.13 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ paste\ := [\, s\ ?\ sort\,;\, l1, l2 :\ seq\,(s);\ pos\ :\ nat \\
\qquad\qquad \vdash subseq\,(1, pos, l1) +\!\!+ l2 +\!\!+ subseq(pos + 1, \text{len}l1, l1) \\
\qquad\qquad] \\
;\ cut\ \ \ := [\, s\ ?\ sort\,;\ l\ :\ seq\,(s);\, begin, end :\ nat \\
\qquad\qquad \vdash subseq\,(1, begin, l) +\!\!+ subseq\,(begin + end + 1, \text{len}l, l) \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.7.14.

$\langle$ Generating a Sequence of Ascending Numbers B.9.7.14 $\rangle \equiv$

$[\![\; count\_up \quad : [nat; nat \vdash seq(nat)]$

$; \; def\_count\_up : [\, n \;?\; nat$

$\quad \vdash \langle\!| \; empty \; := \; count\_up\,(n, 0) = \langle\rangle$

$\quad , \; recur \; := [\, m \;?\; nat$

$\qquad \vdash count\_up\,(n, succ(m)) = count\_up(n, m) +\!\!+ \langle \; n + succ(m \;) \rangle$

$\qquad]$

$\quad |\!\rangle$

$\quad]$

$]\!]$

B.9.7.15.

$\langle$ First Position of an Element in a Sequence B.9.7.15 $\rangle \equiv$

$[\![\; pos\_of \; : [\, s \;?\; sort \;; s; seq(s) \vdash nat \,]$

$]\!]$

B.9.7.16.

$\langle$ Derived Laws of Sequences B.9.7.16 $\rangle \equiv$

$[\![\; dummy \; : \; \mathbf{prim}$

$]\!]$

B.9.8 Finite mappings

$\langle$ Finite Maps B.9.8 $\rangle \equiv$

context $FiniteMaps :=$

$[\![\langle$ Map Construction B.9.8.1 $\rangle$

$; \langle$ Domain of a Map B.9.8.2 $\rangle$

$; \langle$ Basic Laws of Finite Maps B.9.8.3 $\rangle$

$; \langle$ Operations upon Finite Maps B.9.8.5 $\rangle$

$; \langle$ Derived Laws of Finite Maps B.9.8.13 $\rangle$

$]\!]$

B.9.8.1.

$\langle$ Map Construction B.9.8.1 $\rangle \equiv$

$[\![\; (\cdot) \xrightarrow{m} (\cdot) \; : \quad [sort; sort \vdash sort]$

$; \langle\rangle \qquad\quad : \quad [s\_1, s\_2 \;?\; sort \; \vdash s\_1 \xrightarrow{m} s\_2]$

$; \; (\cdot) \odot (\cdot) \quad : \quad [s\_1, s\_2 \;?\; sort \;; s\_1 \otimes s\_2; s\_1 \xrightarrow{m} s\_2 \vdash s\_1 \xrightarrow{m} s\_2]$

$; \; \mathbf{opspec} \;\; \mathbf{right} \; \odot$

$; \; \mathbf{opspec} \; \odot > \, =$

$; \langle(\cdot) \mapsto (\cdot)\rangle \; := [s\_1, s\_2 \;?\; sort \;; a : \; s\_1 \;; b : \; s\_2 \; \vdash (a \mapsto b) \odot \langle\rangle]$

$]\!]$

B.9.8.2.

$\langle$ Domain of a Map B.9.8.2 $\rangle \equiv$

$[\![\mathbf{dom}\,(\cdot) \;:[s\_1, s\_2\,?\;\; sort \;\vdash [s\_1 \xrightarrow{m} s\_2 \vdash set(s\_1)\,]\,]$

$;\; \mathbf{opspec}\;\; \mathbf{dom} \lessdot \odot$

$;\; \mathbf{opspec}\;\; \mathbf{dom} \gtrdot \setminus$

$;\; domain \;:[s\_1, s\_2\,?\;\; sort$

$\quad \vdash \langle\!| \; empty \;:= \mathbf{dom}(\langle\rangle \therefore s\_1 \xrightarrow{m} s\_2) = \{\,\}$

$\quad\quad ,\; recur \;\; := [\, x \;\;?\;\; s\_1 \;;\; y \;\;?\;\; s\_2 \;;\; m \;\;?\, s\_1 \xrightarrow{m} s\_2$

$\quad\quad\quad \vdash \mathbf{dom}(x \mapsto y) \odot m = x \odot \mathbf{dom}\; m$

$\quad\quad\quad]$

$\quad\quad |\!\rangle$

$\quad]$

$]\!]$

B.9.8.3.

$\langle$ Basic Laws of Finite Maps B.9.8.3 $\rangle \equiv$

$[\![\; commute\_map \;\;:[s\_1, s\_2\,?\;\; sort\,;\, x\_1, x\_2\,?\;\; s\_1\,;\, y, z\,?\;\; s\_2\,;\;\; m \;\;?\, s\_1 \xrightarrow{m} s\_2$

$\quad \vdash [x\_1 \neq x\_2$

$\quad\quad \vdash (x\_1 \mapsto y) \odot (x\_2 \mapsto z) \odot m = (x\_2 \mapsto z) \odot (x\_1 \mapsto y) \odot m$

$\quad\quad]$

$\quad]$

$;\;\; overwrite\_map \;\;:[s\_1, s\_2\,?\;\; sort\,;\;\; x \;\;?\;\; s\_1\,;\, y, z\,?\;\; s\_2\,;\;\; m \;\;?\, s\_1 \xrightarrow{m} s\_2$

$\quad \vdash (x \mapsto y) \odot (x \mapsto z) \odot m = (x \mapsto y) \odot m$

$\quad]$

$;\;\; map\_induction \;\;:[s\_1, s\_2\,?\;\; sort\,;\;\; P \;\;?\,[s\_1 \xrightarrow{m} s\_2 \vdash prop\,]$

$$\vdash \dfrac{\begin{array}{l} P\;(\langle\rangle); \\ \quad [\, x \;:\; s\_1 \;;\; y \;:\; s\_2 \;;\; m \;:\, s\_1 \xrightarrow{m} s\_2 \\ \quad \vdash [\; P\;(m); x \notin \mathbf{dom} m \vdash P((x \mapsto y) \odot m)\,] \\ \quad] \end{array}}{[\; m : s\_1 \xrightarrow{m} s\_2 \vdash P(m)\,]}$$

$\quad]$

$]\!]$

B.9.8.4.

B.9.8.5.

$\langle$ Operations upon Finite Maps B.9.8.5 $\rangle \equiv$

$[\![\langle$ Range of a Map B.9.8.6 $\rangle$

$; \langle$ Union of two Maps B.9.8.7 $\rangle$

$; \langle$ Map Application B.9.8.8 $\rangle$

$; \langle$ Filtering a Map by a Predicate B.9.8.9 $\rangle$

$;\langle$ Mapping a Map over a Sequence B.9.8.10 $\rangle$
$;\langle$ Converting an Abstraction into a Map B.9.8.11 $\rangle$
$;\langle$ Relating Sequences and Maps B.9.8.12 $\rangle$
$]\!]$

B.9.8.6.

$\langle$ Range of a Map B.9.8.6 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathrm{rng}\ (\cdot)\ : [s\_1, s\_2\ ?\ sort\ ; s\_1 \xrightarrow{m} s\_2 \vdash set(s\_2)] \\
;\ \mathbf{opspec}\ \ \mathrm{rng}\ \lessdot \odot \\
;\ \mathbf{opspec}\ \ \mathrm{rng}\ \gtrdot \backslash \\
;\ \ range \quad : [s\_1, s\_2\ ?\ sort \\
\qquad\qquad \vdash \langle\!| \ empty\ := \ \mathrm{rng}\ (\langle\rangle \therefore s\_1 \xrightarrow{m} s\_2) = \{\,\} \\
\qquad\qquad\quad ,\ recur\ \ := [\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2\ ;\ m\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad\qquad\qquad\qquad \vdash [a \notin \mathbf{dom} m \vdash\ \mathrm{rng}\ (a \mapsto b) \odot m = b \odot\ \mathrm{rng}\ \ m\ ] \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.8.7.

$\langle$ Union of two Maps B.9.8.7 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot) \cup (\cdot) \quad : [s\_1, s\_2\ ?\ sort\ ; s\_1 \xrightarrow{m} s\_2; s\_1 \xrightarrow{m} s\_2 \vdash s\_1 \xrightarrow{m} s\_2] \\
;\ \mathbf{opspec}\ \ \cup \gtrdot = \\
;\ \mathbf{opspec}\ \ \cup \lessdot \odot \\
;\ \ mapunion\ : [s\_1, s\_2\ ?\ sort\ ;\ m\_1\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad\qquad \vdash \langle\!| \ empty\ := \langle\rangle \cup m\_1 = m\_1 \\
\qquad\qquad\quad ,\ recur\ \ := [\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2\ ;\ m\_2\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad\qquad\qquad\qquad \vdash [a \notin \mathbf{dom}\ m\_1 \\
\qquad\qquad\qquad\qquad\quad \vdash (a \mapsto b) \odot m\_1 \cup m\_2 = (a \mapsto b) \odot (m\_1 \cup m\_2) \\
\qquad\qquad\qquad\qquad\quad] \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.8.8.

$\langle$ Map Application B.9.8.8 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot) \nabla (\cdot)\ : [s\_1, s\_2\ ?\ sort\ ; s\_1 \xrightarrow{m} s\_2; s\_1 \vdash s\_2] \\
;\ \mathbf{opspec}\ \ \mathbf{right}\ \nabla \doteq \cup
\end{array}
$$

$$
\begin{array}{l}
;\ \mathit{app} \quad : [s\_1, s\_2 ?\ \mathit{sort}\ ;\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2\ ;\ m\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad \vdash \lmoustache\ \mathit{first} \ := ((a \mapsto b) \odot m) \nabla a = b \\
\qquad\quad ,\ \mathit{recur}\ := [\ c\ ?\ s\_1 \\
\qquad\qquad\qquad ;\ c \neq a \wedge c \in \mathbf{dom}\ m \\
\qquad\qquad\qquad \vdash (a \mapsto b) \odot m \nabla c = m \nabla\ c \\
\qquad\qquad\qquad] \\
\qquad\quad \rmoustache \\
\qquad] \\
]\!]
\end{array}
$$

B.9.8.9.

$\langle$ Filtering a Map by a Predicate B.9.8.9 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot) \rhd (\cdot) \quad : [s\_1, s\_2 ?\ \mathit{sort}\ ; [s\_1; s\_2 \vdash \mathit{prop}]; s\_1 \xrightarrow{m} s\_2 \vdash s\_1 \xrightarrow{m} s\_2] \\
;\ \mathbf{opspec\ \ right}\ \ \rhd \doteq \cup \\
;\ \mathit{mapfilter}\ : [s\_1, s\_2 ?\ \mathit{sort}\ ;\ P\ ? [s\_1; s\_2 \vdash \mathit{prop}\,] \\
\qquad \vdash \lmoustache\ \mathit{empty}\ := P \rhd \langle\rangle = \langle\rangle \\
\qquad\quad ,\ \mathit{cons} \quad := [\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2\ ;\ m\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad\qquad\qquad \vdash \lmoustache\ \mathit{true}\ \ := [\ P\ (a, b) \\
\qquad\qquad\qquad\qquad\qquad \vdash P \rhd (a \mapsto b) \odot m = (a \mapsto b) \odot (P \rhd m) \\
\qquad\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad\quad ,\ \mathit{false}\ := [\neg\ P\ (a, b) \\
\qquad\qquad\qquad\qquad\qquad ;\ a \notin \mathbf{dom}\ m \\
\qquad\qquad\qquad\qquad\qquad \vdash P \rhd (a \mapsto b) \odot m = P \rhd m \\
\qquad\qquad\qquad\qquad\qquad] \\
\qquad\qquad\qquad\quad \rmoustache \\
\qquad\qquad\qquad] \\
\qquad\quad \rmoustache \\
\qquad] \\
]\!]
\end{array}
$$

B.9.8.10.

$\langle$ Mapping a Map over a Sequence B.9.8.10 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot) * (\cdot) \quad : [s\_1, s\_2 ?\ \mathit{sort}\ ; s\_1 \xrightarrow{m} s\_2; \mathit{seq}(s\_1) \vdash \mathit{seq}(s\_2)\,] \\
;\ \mathbf{opspec\ \ right}\ * \doteq \cup \\
;\ \mathit{map\_map}\ : [s\_1, s\_2 ?\ \mathit{sort}\ ;\ m\ ?\ s\_1 \xrightarrow{m} s\_2 \\
\qquad \vdash \lmoustache\ \mathit{empty}\ :=\ m * \langle\rangle = \langle\rangle \\
\qquad\quad ,\ \mathit{recur}\ \ := [\ a\ ?\ s\_1\ ;\ l\ ?\ \mathit{seq}\ (s\_1) \\
\qquad\qquad\qquad \vdash\ m * a \odot l = (m \nabla a) \odot (m * l) \\
\qquad\qquad\qquad] \\
\qquad\quad \rmoustache \\
\qquad] \\
]\!]
\end{array}
$$

B.9.8.11.

$\langle$ Converting an Abstraction into a Map B.9.8.11 $\rangle \equiv$

$$
\begin{array}{lll}
\llbracket\ atm & : [s\_1, s\_2\ ?\ sort\ ; [s\_1 \vdash s\_2]; set(s\_1) \vdash s\_1 \xrightarrow{m} s\_2\,] & \\
;\ abs\_to\_map & : [s\_1, s\_2\ ?\ sort\ ;\ F\ ?\ [s\_1 \vdash s\_2] & \\
 & \vdash \langle\!|\ empty\ := \ atm\ (F, \{\ \}) = \langle\rangle & \\
 & \quad ,\ recur\ \ := [\ x\ ?\ set\ (s\_1);\ a\ ?\ s\_1 & \\
 & \qquad\qquad \vdash atm\ (F, a \odot x) = (a \mapsto F(a)) \odot atm(F, x) & \\
 & \qquad\qquad] & \\
 & \quad |\!\rangle & \\
 &] & \\
\rrbracket & &
\end{array}
$$

B.9.8.12.

$\langle$ Relating Sequences and Maps B.9.8.12 $\rangle \equiv$

$$
\begin{array}{ll}
\llbracket\ sam & : [\ s\ ?\ sort\ ; seq(s) \vdash nat \xrightarrow{m} s\,] \\
;\ seq\_as\_map & : \langle\!|\ empty\ := \ sam\ (\langle\rangle) = \langle\rangle \\
 & \quad ,\ recur\ \ := [\ s\ ?\ sort\ ;\ a\ ?\ s\ ;\ l\ ?\ seq\ (s) \\
 & \qquad\qquad \vdash sam\ (l +\!\!+ \langle\ a\ \rangle) = ((\mathrm{len}\ l + 1) \mapsto a) \odot sam(l) \\
 & \qquad\qquad] \\
 & \quad |\!\rangle \\
;\ spam & : [s\_1, s\_2\ ?\ sort\ ; seq(s\_1 \otimes s\_2) \vdash s\_1 \xrightarrow{m} s\_2\,] \\
;\ seq\_pair\_as\_map & : [s\_1, s\_2\ ?\ sort \\
 & \vdash \langle\!|\ empty\ := \ spam\ (\langle\rangle \therefore seq(s\_1 \otimes s\_2)) = \langle\rangle \\
 & \quad ,\ recur\ \ := [\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2\ ;\ l\ ?\ seq\ (s\_1 \otimes s\_2) \\
 & \qquad\qquad \vdash spam\ ((a \mapsto b) \odot l) = (a \mapsto b) \odot spam(l) \\
 & \qquad\qquad] \\
 & \quad |\!\rangle \\
 &] \\
\rrbracket &
\end{array}
$$

B.9.8.13.

$\langle$ Derived Laws of Finite Maps B.9.8.13 $\rangle \equiv$

$$
\begin{array}{ll}
\llbracket\ dom\_prop & : [s\_1, s\_2\ ?\ sort\ ;\ a\ ?\ s\_1\ ;\ b\ ?\ s\_2 \\
 & \vdash \langle\!|\ single\ \ := \mathbf{dom}\langle a \mapsto b\rangle = \{a\} \\
 & \quad ,\ subset\ := [\ m\ ?\ s\_1 \xrightarrow{m} s\_2 \\
 & \qquad\qquad \vdash \mathbf{dom} m \subseteq (\mathbf{dom}(a \mapsto b) \odot m) \\
 & \qquad\qquad] \\
 & \quad |\!\rangle \\
 &]
\end{array}
$$

$$
\begin{array}{lll}
; \ \mathit{map\_map\_prop} & : [s\_1, s\_2 ? \ \mathit{sort} \ ; \ m \ ? \ s\_1 \xrightarrow{m} s\_2 & \\
& \vdash \langle\!| \ \mathit{single} \ := [\ a \ ? \ s\_1 \ ; \ b \ ? \ s\_2 & \\
& \qquad\qquad \vdash ((a \mapsto b) \odot m) * \langle \ a \ \rangle = \langle \ b \ \rangle & \\
& \qquad\qquad] & \\
& \quad , \mathit{join} \quad := [l\_1, l\_2 ? \ \mathit{seq}\,(s\_1) & \\
& \qquad\qquad \vdash m * (l\_1 +\!\!+ l\_2) = (m * l\_1) +\!\!+ (m * l\_2) & \\
& \qquad\qquad] & \\
& \quad , \mathit{reduce} \ := [\ a \ ? \ s\_1 \ ; \ b \ ? \ s\_2 \ ; \ l \ ? \ \mathit{seq}\,(s\_1) & \\
& \qquad\qquad \vdash [a \notin \mathrm{elems}\, l & \\
& \qquad\qquad\quad \vdash ((a \mapsto b) \odot m) * l = m * l & \\
& \qquad\qquad\quad] & \\
& \qquad\qquad] & \\
& \quad , \mathit{char} \quad := [\ l \ ? \ \mathit{seq}\,(s\_1) & \\
& \qquad\qquad \vdash m * l & \\
& \qquad\qquad\quad = [\ a : s\_1 \ \vdash \ m \nabla \ a \ ] * ([\ a : s\_1 \ \vdash \ a \in \mathbf{dom}\ m \ ] \rhd l) & \\
& \qquad\qquad] & \\
& \quad |\!\rangle & \\
&] & \\
; \ \mathit{abs\_to\_map\_prop} & : [s\_1, s\_2 ? \ \mathit{sort} \ ; \ F \ ? [s\_1 \vdash s\_2] & \\
& \vdash \langle\!| \ \mathit{single} \ := [\ a \ ? \ s\_1 \ \vdash \ \mathit{atm}(F, \{a\}) = \langle a \mapsto F(a) \rangle] & \\
& \quad , \mathit{dom} \quad := [\ x \ ? \ \mathit{set}\,(s\_1) \vdash \mathbf{dom}\ \mathit{atm}\ (F, x) = x\ ] & \\
& \quad , \mathit{apply} \ := [\ x \ ? \ \mathit{set}\,(s\_1); \ a \ ? \ s\_1 & \\
& \qquad\qquad \vdash [a \in x \vdash \mathit{atm}(F, x) \nabla a = F(a)] & \\
& \qquad\qquad] & \\
& \quad , \mathit{exten} \ := [\ G \ ? [s\_1 \vdash s\_2]; \ x \ ? \ \mathit{set}\,(s\_1) & \\
& \qquad\qquad \vdash [[\ a \ ? \ s\_1 \ ; a \in x \vdash F(a) = G(a)\ ] \vdash \mathit{atm}(F, x) = \mathit{atm}(G, x)\ ] & \\
& \qquad\qquad] & \\
& \quad |\!\rangle & \\
&] & \\
; \ \mathit{mfilter\_subset} & : [\ D, R \ ? \ \mathit{sort} & \\
& \ ; \ m \quad ? \ D \xrightarrow{m} R & \\
& \ ; \ P \quad ? \ [D; R \vdash \mathit{prop}\,] & \\
& \vdash \mathbf{dom}(P \rhd m) \subseteq \mathbf{dom}\ m & \\
&] & \\
]\!] & &
\end{array}
$$

B.9.9 Trees

$\langle$ Trees B.9.9 $\rangle \equiv$

context *Trees* $:=$

$[\![$ $\langle$ Tree Construction B.9.9.1 $\rangle$

; $\langle$ Basic Laws of Trees B.9.9.2 $\rangle$

; $\langle$ Operations upon Trees B.9.9.3 $\rangle$

; $\langle$ Derived Laws of Trees B.9.9.8 $\rangle$

$]\!]$

B.9.9.1.

$\langle$ Tree Construction B.9.9.1 $\rangle \equiv$

$$
\begin{array}{ll}
[\![\ \mathit{tree} & : [\mathit{sort} \vdash \mathit{sort}\,] \\
;\ \tau & : [\, s \mathbin{?} \mathit{sort}\ \vdash \mathit{tree}(s)\,] \\
;\ \mathit{node} & : [\, s \mathbin{?} \mathit{sort}\,; s;\ \mathit{seq}(\mathit{tree}(s)) \vdash \mathit{tree}(s)\,] \\
]\!]
\end{array}
$$

B.9.9.2.

$\langle$ Basic Laws of Trees B.9.9.2 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ (\cdot)\ \text{okon}\ (\cdot) := [\, s\ ?\ \mathit{sort}\,;\ P\ : [\, \mathit{tree}\,(s) \vdash \mathit{prop}\,];\ \mathit{br}\ :\ \mathit{seq}\,(\mathit{tree}(s)) \\
\qquad\qquad\qquad\quad \vdash P \rhd \mathit{br} = \mathit{br} \\
\qquad\qquad\qquad\quad] \\
;\ \textbf{opspec left}\ \ \text{okon}\ \ \dot{=}= \\
;\ \mathit{tree\_induct}\ \ :\ \ [\, s\ ?\ \mathit{sort}\,;\ P\ : [\, \mathit{tree}\,(s) \vdash \mathit{prop}\,] \\
\qquad\qquad\qquad\quad \vdash \left| \begin{array}{l} P\,(\tau); \\ \qquad [\, a\ :\ s\,;\ \mathit{br}\ :\ \mathit{seq}\,(\mathit{tree}(s)) \\ \qquad \vdash [\, P\ \ \text{okon}\ \mathit{br} \vdash P(\mathit{node}(a, \mathit{br}))\,] \\ \qquad] \\ \hline [\, t : \mathit{tree}\,(s) \vdash P(t)\,] \end{array} \right. \\
\qquad\qquad\qquad\quad] \\
]\!]
\end{array}
$$

B.9.9.3.

$\langle$ Operations upon Trees B.9.9.3 $\rangle \equiv$

$[\![\langle$ Extracting Elements from a Tree B.9.9.4 $\rangle$

$;\langle$ Test of Duplicates B.9.9.5 $\rangle$

$;\langle$ Tree Insertion B.9.9.6 $\rangle$

$;\langle$ Path from Root to Element B.9.9.7 $\rangle$

$]\!]$

B.9.9.4.

$\langle$ Extracting Elements from a Tree B.9.9.4 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathit{info} \quad : [\, s \mathbin{?} \mathit{sort}\,; \mathit{tree}(s) \vdash \mathit{set}(s)\,] \\
;\ \mathit{def\_info} : [\, s\ ?\ \mathit{sort} \\
\qquad\qquad \vdash \langle\!|\ \mathit{empty}\ := \ \mathit{info}\,(\tau \mathbin{.\!\cdot\!.} \mathit{tree}(s)) = \{\,\} \\
\qquad\qquad\quad ,\ \mathit{recur}\ \ := [\, a\ ?\ s\ ;\ \mathit{br}\ ?\ \mathit{seq}\,(\mathit{tree}(s)) \\
\qquad\qquad\qquad\qquad \vdash \mathit{info}\,(\mathit{node}(a, \mathit{br})) = a \odot \bigcup \text{elems}\ \mathit{info} * \mathit{br} \\
\qquad\qquad\qquad\qquad] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
]\!]
\end{array}
$$

B.9.9.5.

$\langle$ Test of Duplicates B.9.9.5 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ nodup \quad : [\ s\ ?\ sort\ ;\ tree(s) \vdash prop\ ] \\
;\ def\_nodup : [\ s\ ?\ sort \\
\qquad \vdash \langle\!|\ empty\ := nodup\ (\tau \therefore tree(s)) = true \\
\qquad\quad ,\ recur\ := [\ a\ ?\ s\ ;\ br\ ?\ seq\ (tree(s)) \\
\qquad\qquad \vdash nodup\ (node(a, br)) \\
\qquad\qquad\quad \Leftrightarrow a \notin \bigcup \text{elems}\ info * br \wedge \bigcap \text{elems}\ info * br = \{\ \} \wedge nodup\ \text{okon}\ \ br \\
\qquad\qquad] \\
\qquad\quad |\!\rangle \\
\qquad] \\
]\!]
\end{array}
$$

B.9.9.6.

$\langle$ Tree Insertion B.9.9.6 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ insert \quad : [\ s\ ?\ sort\ ;\ s; s; tree(s) \vdash tree(s)\ ] \\
;\ def\_insert : [\ s\ ?\ sort\ ;\ a, b\ ?\ s\ ;\ br\ ?\ seq\ (tree(s)) \\
\qquad \vdash \langle\!|\ empty\ := insert\ (a, b, \tau) = \tau \\
\qquad\quad ,\ recur\ := [\ br\ ?\ seq\ (tree(s)) \\
\qquad\qquad \vdash \langle\!|\ insert\ (a, b, node(a, br)) = node(a, br +\!\!+ \langle\ node(b, \langle\rangle\ \rangle)) \\
\qquad\qquad\quad ,[\ c\ ?\ s \\
\qquad\qquad\quad ;\ a \neq c \\
\qquad\qquad\quad \vdash insert\ (a, b, node(c, br)) = node(c, insert(a, b) * br) \\
\qquad\qquad\quad] \\
\qquad\qquad\quad |\!\rangle \\
\qquad\qquad] \\
\qquad\quad |\!\rangle \\
\qquad] \\
]\!]
\end{array}
$$

B.9.9.7.

$\langle$ Path from Root to Element B.9.9.7 $\rangle \equiv$

$[\![\ init\_path \quad : [\ s\ ?\ sort\ ;\ s; tree(s) \vdash seq(s)\ ]$

$$
\begin{array}{l}
;\ \mathit{def\_init\_path}\ : [\, s\ ?\ \mathit{sort}\ ;\ a\ ?\ s \\
\qquad \vdash \langle\!| \ \mathit{empty}\ := \ \mathit{init\_path}\ (a, \tau \therefore \mathit{tree}(s)) = \langle\rangle \\
\qquad\quad ,\ \mathit{recur}\ := [\, \mathit{br}\ ?\ \mathit{seq}\ (\mathit{tree}(s)) \\
\qquad\qquad \vdash \langle\!|\ \mathit{init\_path}\ (a, \mathit{node}(a, \mathit{br})) = \langle\ a\ \rangle \\
\qquad\qquad\quad ,[\, b\ ?\ s \\
\qquad\qquad\quad ;\ a \neq b \\
\qquad\qquad\quad \vdash \mathit{init\_path}\ (a, \mathit{node}(b, \mathit{br})) \\
\qquad\qquad\qquad = \langle\ b\ \rangle +\!\!+ \mathrm{hd}\ \mathit{init\_path}\ (a) * ([\, t : \mathit{tree}\ (s) \vdash a \in \mathit{info}(t)\,] \rhd \mathit{br}) \\
\qquad\qquad\quad] \\
\qquad\qquad |\!\rangle \\
\qquad\qquad] \\
\qquad\quad |\!\rangle \\
\qquad] \\
]\!]
\end{array}
$$

B.9.9.8.

$\langle$ Derived Laws of Trees B.9.9.8 $\rangle \equiv$

$$
\begin{array}{l}
[\![\ \mathit{info\_prop}\ : [\, s\ ?\ \mathit{sort}\ ;\ a\ ?\ s \\
\qquad \vdash \langle\!|\ \mathit{single}\ := \ \mathit{info}\ (\mathit{node}(a, \langle\rangle)) = \{a\} \\
\qquad\quad ,\ \mathit{insert}\ := [\, b\ ?\ s\ ;\ t\ ?\ \mathit{tree}\ (s) \\
\qquad\qquad ;\ a \in \mathit{info}(t) \\
\qquad\qquad \vdash \mathit{info}\ (\mathit{insert}(a, b, t)) = b \odot \mathit{info}(t) \\
\qquad\qquad] \\
\qquad\quad |\!\rangle \\
\qquad] \\
;\ \mathit{nodup\_prop}\ : [\, s\ ?\ \mathit{sort}\ ;\ a\ ?\ s \\
\qquad \vdash \langle\!|\ \mathit{single}\ := \ \mathit{nodup}\ (\mathit{node}(a, \langle\rangle)) = \mathit{true} \\
\qquad\quad ,\ \mathit{insert}\ := [\, b\ ?\ s\ ;\ t\ ?\ \mathit{tree}\ (s) \\
\qquad\qquad \vdash [\ \mathit{nodup}\ (t) \\
\qquad\qquad\quad ;\ a \in \mathit{info}(t) \\
\qquad\qquad\quad ;\ b \notin \mathit{info}(t) \\
\qquad\qquad\quad \vdash \mathit{nodup}\ (\mathit{insert}(a, b, t)) \\
\qquad\qquad\quad] \\
\qquad\qquad] \\
\qquad\quad |\!\rangle \\
\qquad]
\end{array}
$$

$; \ init\_path\_prop \ : [\ s \ ? \ sort \ ; a, b \ ? \ s \ ; \ t \ ? \ tree\,(s)$
$\quad \vdash [\ nodup\ (t)$
$\quad\quad ; a \in info(t)$
$\quad\quad ; b \notin info(t)$
$\quad\quad \vdash \langle\!| \ last \ := \ init\_path\ (b, insert(a, b, t)) = init\_path(a, t) \mathbin{++} \langle \ b \ \rangle$
$\quad\quad\quad , \ inv \ := [\ c \ ? \ s$
$\quad\quad\quad\quad ; c \in info(t)$
$\quad\quad\quad\quad \vdash init\_path\ (a, insert(c, b, t)) = init\_path(a, t)$
$\quad\quad\quad\quad]$
$\quad\quad\quad |\!\rangle$
$\quad\quad]$
$\quad]$
$; \ path\_incl \ : [\ s \ ? \ sort \ ; \ a \ ? \ s \ ; \ tr \ ? \ tree\,(s)$
$\quad \vdash \mathrm{elems}\ init\_path\ (a, tr) \subseteq info(tr)$
$\quad]$
$]\!]$

B.10 Putting it all together

$[\![$ *sort* : **prim**
; **context** *BasicDatatypes* :=
$[\![$ ⟨ Equality B.9.1 ⟩
; **import** *Equality*
; ⟨ Propositions B.9.2 ⟩
; **import** *Propositions*
; ⟨ Ordered Pairs B.9.3 ⟩
; **import** *OrderedPairs*
; ⟨ Quantifiers B.9.4 ⟩
; **import** *Quantifiers*
; ⟨ Natural Numbers B.9.5 ⟩
; **import** *NaturalNumbers*
; ⟨ Finite Sets B.9.6 ⟩
; **import** *FiniteSets*
; ⟨ Sequences B.9.7 ⟩
; **import** *Sequences*
; ⟨ Finite Maps B.9.8 ⟩
; **import** *FiniteMaps*
; ⟨ Trees B.9.9 ⟩
; **import** *Trees*
$]\!]$
; **import** *BasicDatatypes*
; ⟨ Methodology. B.8 ⟩
; **import** *ReificationMethodology*
; **import** *Module_Interface_Library*

; ⟨ Deltas. B.7.2 ⟩

; ⟨ Files. B.7.1 ⟩

; **import** *Files*

; ⟨ The devlopment of a revision management system. B.1.1 ⟩

⟧

B.11 Table of Deva sections

⟨ 1st. data reification (delta technique). B.1.4 ⟩ This code is used in section B.1.1.

⟨ 2nd. data reification (global array). B.1.5 ⟩ This code is used in section B.1.1.

⟨ Abstract post-condition (*OPEN*). B.4.4.4 ⟩ This code is used in section B.4.4.2.

⟨ Abstract specification of the kernel system. B.1.3 ⟩ This code is used in section B.1.1.

⟨ Accessing an Element in a Sequence B.9.7.11 ⟩ This code is used in section B.9.7.3.

⟨ Application of a Delta Unit to a Sequence B.7.2.6 ⟩ This code is used in section B.7.2.3.

⟨ Application of a Delta to a List B.7.2.13 ⟩ This code is used in section B.7.2.11.

⟨ Application of a Sequence of Deltas to a List B.7.2.14 ⟩ This code is used in section B.7.2.11.

⟨ Apply a Map to a Deltas Inserts B.7.2.16 ⟩ This code is used in section B.7.2.11.

⟨ Apply map to object in domain. B.8.5.8 ⟩ This code is used in section B.8.5.3.

⟨ Assumed inverse of retrieve function. B.1.4.11 ⟩ This code is used in section B.1.4.10.

⟨ Auxiliary definitions. B.2.4.1 ⟩ This code is used in section B.2.4.

⟨ Auxiliary lemma. B.5.5.4 ⟩ This code is used in section B.5.5.1.

⟨ Auxiliary lemmas (preservation). B.4.2.1 ⟩ This code is used in section B.4.2.

⟨ Axioms of equality. B.8.4.15 ⟩ This code is used in section B.8.4.1.

⟨ Basic Arithmetical Operations B.9.5.3 ⟩ This code is used in section B.9.5.

⟨ Basic Laws of Finite Maps B.9.8.3 ⟩ This code is used in section B.9.8.

⟨ Basic Laws of Finite Sets B.9.6.2 ⟩ This code is used in section B.9.6.

⟨ Basic Laws of Propositions B.9.2.4 ⟩ This code is used in section B.9.2.

⟨ Basic Laws of Quantifiers B.9.4.2 ⟩ This code is used in section B.9.4.

⟨ Basic Laws of Sequences B.9.7.2 ⟩ This code is used in section B.9.7.

⟨ Basic Laws of Trees B.9.9.2 ⟩ This code is used in section B.9.9.

⟨ Basic Quantifiers B.9.4.1 ⟩ This code is used in section B.9.4.

⟨ Basic Relations B.9.5.6 ⟩ This code is used in section B.9.5.

⟨ Basic sorts and constants. B.1.2 ⟩ This code is used in section B.1.1.

⟨ Body of the result case. B.4.6.5 ⟩ This code is used in section B.4.6.3.

⟨ Calculus of operations. B.8.4 ⟩ This code is used in section B.8.1.

⟨ Cardinality of a Set B.9.6.8 ⟩ This code is used in section B.9.6.3.

⟨ Case distinction is complete (CHECKIN). B.2.4.8 ⟩ This code is used in section B.2.4.7.

⟨ Case distinction is complete (D:CHECKIN). B.3.4.8 ⟩ This code is used in section B.3.4.7.

⟨ Case $r = new$ (CHECKIN). B.2.4.9 ⟩ This code is used in section B.2.4.7.

⟨ Case $r = nr$. B.3.4.9 ⟩ This code is used in section B.3.4.7.

⟨ Case r unequal to new (CHECKIN). B.2.4.10 ⟩ This code is used in section B.2.4.7.

⟨ Case r unequal to nr. B.3.4.11 ⟩ This code is used in section B.3.4.7.

⟨ Check in a file (delta technique). B.1.4.6 ⟩ This code is used in section B.1.4.3.

⟨ Check in a file (global array). B.1.5.7 ⟩ This code is used in section B.1.5.4.

⟨ Check in a file (locks). B.1.6.7 ⟩ This code is used in section B.1.6.2.

⟨ Check in a file. B.1.3.5 ⟩ This code is used in section B.1.3.2.

⟨ Check out a file (delta technique). B.1.4.7 ⟩ This code is used in section B.1.4.3.

⟨ Check out a file (global array). B.1.5.8 ⟩ This code is used in section B.1.5.4.

⟨ Check out a file (locks). B.1.6.8 ⟩ This code is used in section B.1.6.2.

⟨ Check out a file. B.1.3.6 ⟩ This code is used in section B.1.3.2.

⟨ Complementation of precondition. B.8.4.13 ⟩ This code is used in section B.8.4.2.

⟨ Composition Laws of Equality B.9.1.4 ⟩ This code is used in section B.9.1.2.

⟨ Compute length of sequence. B.8.5.31 ⟩ This code is used in section B.8.5.27.
⟨ Compute path from tree-root to tree-object. B.8.5.21 ⟩ This code is used in section B.8.5.16.
⟨ Compute the size of map. B.8.5.10 ⟩ This code is used in section B.8.5.3.
⟨ Conjunction B.9.2.7 ⟩ This code is used in section B.9.2.5.
⟨ Construction and Selection Operations B.9.3.1 ⟩ This code is used in section B.9.3.
⟨ Construction of Delta Units B.7.2.2 ⟩ This code is used in section B.7.2.1.
⟨ Construction of Deltas B.7.2.10 ⟩ This code is used in section B.7.2.9.
⟨ Construction of Propositions B.9.2.1 ⟩ This code is used in section B.9.2.
⟨ Converting an Abstraction into a Map B.9.8.11 ⟩ This code is used in section B.9.8.5.
⟨ Create empty map. B.8.5.4 ⟩ This code is used in section B.8.5.3.
⟨ Create empty sequence. B.8.5.28 ⟩ This code is used in section B.8.5.27.
⟨ Create empty tree. B.8.5.17 ⟩ This code is used in section B.8.5.16.
⟨ Create root. B.8.5.18 ⟩ This code is used in section B.8.5.16.
⟨ Cut and Paste of a Specified Subsequence B.9.7.13 ⟩ This code is used in section B.9.7.3.
⟨ Defined Quantifiers B.9.4.3 ⟩ This code is used in section B.9.4.
⟨ Defining Axioms B.9.5.5 ⟩ This code is used in section B.9.5.3.
⟨ Definition of input initialisation. B.8.4.8 ⟩ This code is used in section B.8.4.7.
⟨ Definition of signature extension. B.8.4.10 ⟩ This code is used in section B.8.4.9.
⟨ Definition of signature initialisation. B.8.4.7 ⟩ This code is used in section B.8.4.6.
⟨ Definition of state extension. B.8.4.11 ⟩ This code is used in section B.8.4.10.
⟨ Delete association pair from map. B.8.5.9 ⟩ This code is used in section B.8.5.3.
⟨ Delta Units B.7.2.1 ⟩ This code is used in section B.7.2.
⟨ Deltas. B.7.2 ⟩ This code is used in section B.10.
⟨ Derivation of the evaluated postcondition of *CHECKIN* B.5.6.3 ⟩ This code is used in section B.5.6.1.
⟨ Derivation of the evaluated postcondition of *CHECKOUT* B.5.7.2 ⟩ This code is used in section B.5.7.1.
⟨ Derivation of the evaluated postcondition of *FREE* B.5.5.5 ⟩ This code is used in section B.5.5.1.
⟨ Derivation of the evaluated postcondition of *OPEN* B.5.3.2 ⟩ This code is used in section B.5.3.1.
⟨ Derivation of the evaluated postcondition of *RESET* B.5.2.2 ⟩ This code is used in section B.5.2.1.
⟨ Derivation of the evaluated postcondition of *SET* B.5.4.5 ⟩ This code is used in section B.5.4.1.
⟨ Derivation of the evaluated precondition of *SET* B.5.4.4 ⟩ This code is used in section B.5.4.1.
⟨ Derivation of the post-content (*CHECKIN*). B.5.6.2 ⟩ This code is used in section B.5.6.1.
⟨ Derivation of the post-content (*OPEN*). B.5.3.3 ⟩ This code is used in section B.5.3.1.
⟨ Derived Arithmetical Laws B.9.5.7 ⟩ This code is used in section B.9.5.
⟨ Derived Laws of Delta Units B.7.2.8 ⟩ This code is used in section B.7.2.1.
⟨ Derived Laws of Deltas B.7.2.17 ⟩ This code is used in section B.7.2.9.
⟨ Derived Laws of Equality B.9.1.2 ⟩ This code is used in section B.9.1.
⟨ Derived Laws of Finite Maps B.9.8.13 ⟩ This code is used in section B.9.8.
⟨ Derived Laws of Finite Sets B.9.6.14 ⟩ This code is used in section B.9.6.
⟨ Derived Laws of Propositions B.9.2.5 ⟩ This code is used in section B.9.2.
⟨ Derived Laws of Quantifiers B.9.4.4 ⟩ This code is used in section B.9.4.
⟨ Derived Laws of Sequences B.9.7.16 ⟩ This code is used in section B.9.7.
⟨ Derived Laws of Trees B.9.9.8 ⟩ This code is used in section B.9.9.
⟨ Derived properties of equality. B.8.4.16 ⟩ This code is used in section B.8.4.1.
⟨ Derived properties of modules. B.8.2.4 ⟩ This code is used in section B.8.2.
⟨ Derived properties of reification. B.8.3.5 ⟩ This code is used in section B.8.3.
⟨ Derived properties of the calculus of operations. B.8.4.14 ⟩ This code is used in section B.8.4.

⟨ Developments units: operations. B.8.1 ⟩ This code is used in section B.8.
⟨ Difference between two Sets B.9.6.7 ⟩ This code is used in section B.9.6.3.
⟨ Disjunction B.9.2.8 ⟩ This code is used in section B.9.2.5.
⟨ Distinguished Numbers B.9.5.2 ⟩ This code is used in section B.9.5.
⟨ Domain case ($CHECKIN$). B.4.6.2 ⟩ This code is used in section B.4.6.
⟨ Domain case ($OPEN$). B.4.4.1 ⟩ This code is used in section B.4.4.
⟨ Domain lemma ($CHECKOUT$). B.4.5.2 ⟩ This code is used in section B.4.5.
⟨ Domain of a Map B.9.8.2 ⟩ This code is used in section B.9.8.
⟨ Equality B.9.1 ⟩ This code is used in section B.10.
⟨ Equivalence Relation Laws of Equality B.9.1.3 ⟩ This code is used in section B.9.1.
⟨ Evaluation of the operations (locks). B.1.6.11 ⟩ This code is used in section B.1.6.2.
⟨ Evaluation of $CHECKIN$. B.1.6.16 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation of $CHECKOUT$. B.1.6.17 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation of $FREE$. B.1.6.15 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation of $OPEN$. B.1.6.13 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation of $RESET$. B.1.6.12 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation of SET. B.1.6.14 ⟩ This code is used in section B.1.6.11.
⟨ Evaluation proofs B.5.8 ⟩ This code is used in section B.1.6.10.
⟨ Extend map by new association pair. B.8.5.5 ⟩ This code is used in section B.8.5.3.
⟨ Extension by user-held locks. B.1.6 ⟩ This code is used in section B.1.1.
⟨ Extension lemma for $OPEN$ B.5.3.1 ⟩ This code is used in section B.5.3.
⟨ Extension lemma for $RESET$ B.5.2.1 ⟩ This code is used in section B.5.2.
⟨ Extension to robust operations. B.1.7 ⟩ This code is used in section B.1.1.
⟨ Extension: $CHECKIN$ B.5.6.1 ⟩ This code is used in section B.5.6.
⟨ Extension: $CHECKOUT$ B.5.7.1 ⟩ This code is used in section B.5.7.
⟨ Extension: $FREE$ B.5.5.1 ⟩ This code is used in section B.5.5.
⟨ Extension: SET B.5.4.1 ⟩ This code is used in section B.5.4.
⟨ Extracting Elements from a Tree B.9.9.4 ⟩ This code is used in section B.9.9.3.
⟨ Extracting a Specified Subsequence B.9.7.12 ⟩ This code is used in section B.9.7.3.
⟨ Files. B.7.1 ⟩ This code is used in section B.10.
⟨ Filtering a Map by a Predicate B.9.8.9 ⟩ This code is used in section B.9.8.5.
⟨ Filtering a Sequence with a Predicate B.9.7.8 ⟩ This code is used in section B.9.7.3.
⟨ Filtering a Set with a Predicate B.9.6.11 ⟩ This code is used in section B.9.6.3.
⟨ Finite Maps B.9.8 ⟩ This code is used in section B.10.
⟨ Finite Sets B.9.6 ⟩ This code is used in section B.10.
⟨ First Position of an Element in a Sequence B.9.7.15 ⟩ This code is used in section B.9.7.3.
⟨ Flattening a Sequence of Sequences B.9.7.9 ⟩ This code is used in section B.9.7.3.
⟨ Full Deltas B.7.2.9 ⟩ This code is used in section B.7.2.
⟨ Generating a Sequence of Ascending Numbers B.9.7.14 ⟩ This code is used in section B.9.7.3.
⟨ Head and Tail B.9.7.5 ⟩ This code is used in section B.9.7.3.
⟨ Horizontal development: modules and the calculus of operations. B.8.1 ⟩ This code is used in section B.8.
⟨ Implication B.9.2.6 ⟩ This code is used in section B.9.2.5.
⟨ Inequality B.9.2.11 ⟩ This code is used in section B.9.2.5.
⟨ Insert object at the right end of sequence. B.8.5.29 ⟩ This code is used in section B.8.5.27.
⟨ Insert object into tree. B.8.5.19 ⟩ This code is used in section B.8.5.16.
⟨ Intersection of a Set of Sets B.9.6.13 ⟩ This code is used in section B.9.6.3.
⟨ Intersection of two Sets B.9.6.6 ⟩ This code is used in section B.9.6.3.
⟨ Invariant (global array). B.1.5.2 ⟩ This code is used in section B.1.5.1.
⟨ Kernel operations. B.1.3.2 ⟩ This code is used in section B.1.3.
⟨ Kernel state and invariant. B.1.3.1 ⟩ This code is used in section B.1.3.
⟨ Laws of Ordered Pairs B.9.3.2 ⟩ This code is used in section B.9.3.
⟨ Length of a Sequence B.9.7.6 ⟩ This code is used in section B.9.7.3.

⟨ Library of module interfaces. B.8.5 ⟩ This code is used in section B.8.
⟨ Lines changed by a Delta. B.7.2.18 ⟩ This code is used in section B.7.2.11.
⟨ Lock a file for a user. B.1.6.5 ⟩ This code is used in section B.1.6.2.
⟨ Logical Equivalence B.9.2.9 ⟩ This code is used in section B.9.2.5.
⟨ Map Application B.9.8.8 ⟩ This code is used in section B.9.8.5.
⟨ Map Construction B.9.8.1 ⟩ This code is used in section B.9.8.
⟨ Mapping a Map over a Sequence B.9.8.10 ⟩ This code is used in section B.9.8.5.
⟨ Mapping an Abstraction over a Sequence B.9.7.7 ⟩ This code is used in section B.9.7.3.
⟨ Mapping an Abstraction over a Set B.9.6.10 ⟩ This code is used in section B.9.6.3.
⟨ Methodology. B.8 ⟩ This code is used in section B.10.
⟨ Miscalleneous B.9.2.10 ⟩ This code is used in section B.9.2.5.
⟨ Module assembly (delta technique). B.1.4.8 ⟩ This code is used in section B.1.4.
⟨ Module assembly (global array). B.1.5.9 ⟩ This code is used in section B.1.5.
⟨ Module assembly (locks). B.1.6.9 ⟩ This code is used in section B.1.6.
⟨ Module assembly (mappings). B.8.5.12 ⟩ This code is used in section B.8.5.1.
⟨ Module assembly (sequences). B.8.5.33 ⟩ This code is used in section B.8.5.25.
⟨ Module assembly (total operations). B.1.7.1 ⟩ This code is used in section B.1.7.
⟨ Module assembly (trees). B.8.5.23 ⟩ This code is used in section B.8.5.14.
⟨ Module assembly. B.1.3.7 ⟩ This code is used in section B.1.3.
⟨ Module reification. B.8.3.4 ⟩ This code is used in section B.8.3.
⟨ Modules. B.8.2 ⟩ This code is used in section B.8.1.
⟨ Natural Numbers B.9.5 ⟩ This code is used in section B.10.
⟨ New part of the invariant (*FREE*). B.5.5.3 ⟩ This code is used in section B.5.5.1.
⟨ New part of the invariant (*SET*). B.5.4.3 ⟩ This code is used in section B.5.4.1.
⟨ Old part of the invariant (*FREE*). B.5.5.2 ⟩ This code is used in section B.5.5.1.
⟨ Old part of the invariant (*SET*). B.5.4.2 ⟩ This code is used in section B.5.4.1.
⟨ Op-Conjunction of two operations. B.8.4.3 ⟩ This code is used in section B.8.4.2.
⟨ Op-Disjunction of two operations. B.8.4.4 ⟩ This code is used in section B.8.4.2.
⟨ Op-Equality of operations. B.8.4.1 ⟩ This code is used in section B.8.4.
⟨ Open the system with a root file (delta technique). B.1.4.5 ⟩ This code is used in section B.1.4.3.
⟨ Open the system with a root file (global array). B.1.5.6 ⟩ This code is used in section B.1.5.4.
⟨ Open the system with a root file (locks). B.1.6.4 ⟩ This code is used in section B.1.6.2.
⟨ Open the system with a root file. B.1.3.4 ⟩ This code is used in section B.1.3.2.
⟨ Operation list reification. B.8.3.2 ⟩ This code is used in section B.8.3.
⟨ Operation lists. B.8.2.1 ⟩ This code is used in section B.8.2.
⟨ Operation reification. B.8.3.1 ⟩ This code is used in section B.8.3.
⟨ Operations (delta technique). B.1.4.3 ⟩ This code is used in section B.1.4.
⟨ Operations (global array). B.1.5.4 ⟩ This code is used in section B.1.5.
⟨ Operations (locks). B.1.6.2 ⟩ This code is used in section B.1.6.
⟨ Operations (mappings). B.8.5.3 ⟩ This code is used in section B.8.5.1.
⟨ Operations (sequences). B.8.5.27 ⟩ This code is used in section B.8.5.25.
⟨ Operations (trees). B.8.5.16 ⟩ This code is used in section B.8.5.14.
⟨ Operations upon Delta Units B.7.2.3 ⟩ This code is used in section B.7.2.1.
⟨ Operations upon Deltas B.7.2.11 ⟩ This code is used in section B.7.2.9.
⟨ Operations upon Finite Maps B.9.8.5 ⟩ This code is used in section B.9.8.
⟨ Operations upon Sequences B.9.7.3 ⟩ This code is used in section B.9.7.
⟨ Operations upon Sets B.9.6.3 ⟩ This code is used in section B.9.6.
⟨ Operations upon Trees B.9.9.3 ⟩ This code is used in section B.9.9.
⟨ Operations upon operations. B.8.4.2 ⟩ This code is used in section B.8.4.
⟨ Ordered Pairs B.9.3 ⟩ This code is used in section B.10.
⟨ Overwrite a Delta Inserts with Ascending Integers B.7.2.15 ⟩ This code is used in section B.7.2.11.

⟨ Overwrite a Delta Unit Inserts by Ascending Integers B.7.2.7 ⟩ This code is used in section B.7.2.3.
⟨ Path from Root to Element B.9.9.7 ⟩ This code is used in section B.9.9.3.
⟨ Peano Axioms B.9.5.1 ⟩ This code is used in section B.9.5.
⟨ Priorities B.9.2.2 ⟩ This code is used in section B.9.2.1.
⟨ Projection Functions B.7.2.4 ⟩ This code is used in section B.7.2.3.
⟨ Projection lemmas. B.4.1 ⟩ This code is used in section B.1.4.10.
⟨ Proof obligations for modules. B.8.2.3 ⟩ This code is used in section B.8.2.
⟨ Proof obligations for operations. B.8.1.2 ⟩ This code is used in section B.8.1.
⟨ Proof of invariant preservation (1). B.4.2.2 ⟩ This code is used in section B.4.2.
⟨ Proof of invariant preservation (2). B.4.2.3 ⟩ This code is used in section B.4.2.
⟨ Proof of invariant preservation (3). B.4.2.4 ⟩ This code is used in section B.4.2.
⟨ Proof of invariant preservation (4). B.4.2.5 ⟩ This code is used in section B.4.2.
⟨ Proof of invariant preservation (CHECKIN). B.2.4.3 ⟩ This code is used in section B.2.4.
⟨ Proof of invariant preservation (CHECKOUT). B.2.3.2 ⟩ This code is used in section B.2.3.
⟨ Proof of invariant preservation (D:CHECKIN). B.3.4.3 ⟩ This code is used in section B.3.4.
⟨ Proof of invariant preservation (D:CHECKOUT). B.3.3.2 ⟩ This code is used in section B.3.3.
⟨ Proof of invariant preservation (D:OPEN). B.3.2.2 ⟩ This code is used in section B.3.2.
⟨ Proof of invariant preservation (D:RESET). B.3.1.2 ⟩ This code is used in section B.3.1.
⟨ Proof of invariant preservation (OPEN). B.2.2.2 ⟩ This code is used in section B.2.2.
⟨ Proof of invariant preservation (RESET). B.2.1.2 ⟩ This code is used in section B.2.1.
⟨ Proof of lemma IV. B.4.6.6 ⟩ This code is used in section B.4.6.4.
⟨ Proof of lemma V. B.4.6.7 ⟩ This code is used in section B.4.6.4.
⟨ Proof of lemma VI. B.4.6.8 ⟩ This code is used in section B.4.6.4.
⟨ Proof of operation reification (*CHECKIN*). B.4.6 ⟩ This code is used in section B.1.4.12.
⟨ Proof of operation reification (*CHECKOUT*). B.4.5 ⟩ This code is used in section B.1.4.12.
⟨ Proof of operation reification (*OPEN*). B.4.4 ⟩ This code is used in section B.1.4.12.
⟨ Proof of operation reification (*RESET*). B.4.3 ⟩ This code is used in section B.1.4.12.
⟨ Proof of satisfiability (CHECKIN). B.2.4.2 ⟩ This code is used in section B.2.4.
⟨ Proof of satisfiability (CHECKOUT). B.2.3.1 ⟩ This code is used in section B.2.3.
⟨ Proof of satisfiability (D:CHECKIN). B.3.4.2 ⟩ This code is used in section B.3.4.
⟨ Proof of satisfiability (D:CHECKOUT). B.3.3.1 ⟩ This code is used in section B.3.3.
⟨ Proof of satisfiability (D:OPEN). B.3.2.1 ⟩ This code is used in section B.3.2.
⟨ Proof of satisfiability (D:RESET). B.3.1.1 ⟩ This code is used in section B.3.1.
⟨ Proof of satisfiability (OPEN). B.2.2.1 ⟩ This code is used in section B.2.2.
⟨ Proof of satisfiability (RESET). B.2.1.1 ⟩ This code is used in section B.2.1.
⟨ Proof of the first part of the invariant (CHECKIN). B.2.4.4 ⟩ This code is used in section B.2.4.3.
⟨ Proof of the first part of the invariant (D:CHECKIN). B.3.4.4 ⟩ This code is used in section B.3.4.3.
⟨ Proof of the first part of the invariant (D:OPEN). B.3.2.3 ⟩ This code is used in section B.3.2.2.
⟨ Proof of the first part of the invariant (D:RESET). B.3.1.3 ⟩ This code is used in section B.3.1.2.
⟨ Proof of the first part of the invariant (OPEN). B.2.2.3 ⟩ This code is used in section B.2.2.2.
⟨ Proof of the first part of the invariant (RESET). B.2.1.3 ⟩ This code is used in section B.2.1.2.
⟨ Proof of the fourth part of the invariant (CHECKIN). B.2.4.7 ⟩ This code is used in section B.2.4.3.

⟨ Proof of the fourth part of the invariant (D:CHECKIN). B.3.4.7 ⟩ This code is used in section B.3.4.3.
⟨ Proof of the fourth part of the invariant (D:OPEN). B.3.2.6 ⟩ This code is used in section B.3.2.2.
⟨ Proof of the fourth part of the invariant (D:RESET). B.3.1.6 ⟩ This code is used in section B.3.1.2.
⟨ Proof of the fourth part of the invariant (OPEN). B.2.2.6 ⟩ This code is used in section B.2.2.2.
⟨ Proof of the fourth part of the invariant (RESET). B.2.1.6 ⟩ This code is used in section B.2.1.2.
⟨ Proof of the operation reification conditions B.1.4.12 ⟩ This code is used in section B.1.4.10.
⟨ Proof of the second part of the invariant (CHECKIN). B.2.4.5 ⟩ This code is used in section B.2.4.3.
⟨ Proof of the second part of the invariant (D:CHECKIN). B.3.4.5 ⟩ This code is used in section B.3.4.3.
⟨ Proof of the second part of the invariant (D:OPEN). B.3.2.4 ⟩ This code is used in section B.3.2.2.
⟨ Proof of the second part of the invariant (D:RESET). B.3.1.4 ⟩ This code is used in section B.3.1.2.
⟨ Proof of the second part of the invariant (OPEN). B.2.2.4 ⟩ This code is used in section B.2.2.2.
⟨ Proof of the second part of the invariant (RESET). B.2.1.4 ⟩ This code is used in section B.2.1.2.
⟨ Proof of the third part of the invariant (CHECKIN). B.2.4.6 ⟩ This code is used in section B.2.4.3.
⟨ Proof of the third part of the invariant (D:CHECKIN). B.3.4.6 ⟩ This code is used in section B.3.4.3.
⟨ Proof of the third part of the invariant (D:OPEN). B.3.2.5 ⟩ This code is used in section B.3.2.2.
⟨ Proof of the third part of the invariant (D:RESET). B.3.1.5 ⟩ This code is used in section B.3.1.2.
⟨ Proof of the third part of the invariant (OPEN). B.2.2.5 ⟩ This code is used in section B.2.2.2.
⟨ Proof of the third part of the invariant (RESET). B.2.1.5 ⟩ This code is used in section B.2.1.2.
⟨ Proof of $val\_op\ (D_{op}.CHECKIN, D_{mod}.inv)$. B.3.4 ⟩ This code is used in section B.1.4.9.
⟨ Proof of $val\_op\ (D_{op}.CHECKOUT, D_{mod}.inv)$. B.3.3 ⟩ This code is used in section B.1.4.9.
⟨ Proof of $val\_op\ (D_{op}.OPEN, D_{mod}.inv)$. B.3.2 ⟩ This code is used in section B.1.4.9.
⟨ Proof of $val\_op\ (D_{op}.RESET, D_{mod}.inv)$. B.3.1 ⟩ This code is used in section B.1.4.9.
⟨ Proof of $val\_op\ (K_{op}.CHECKIN, K_{mod}.inv)$. B.2.4 ⟩ This code is used in section B.1.3.8.
⟨ Proof of $val\_op\ (K_{op}.CHECKOUT, K_{mod}.inv)$. B.2.3 ⟩ This code is used in section B.1.3.8.
⟨ Proof of $val\_op\ (K_{op}.OPEN, K_{mod}.inv)$. B.2.2 ⟩ This code is used in section B.1.3.8.
⟨ Proof of $val\_op\ (K_{op}.RESET, K_{mod}.inv)$. B.2.1 ⟩ This code is used in section B.1.3.8.
⟨ Proof of $val\_op\ (L_{op}.CHECKIN, L_{mod}.inv)$ B.5.6 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (L_{op}.CHECKOUT, L_{mod}.inv)$ B.5.7 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (L_{op}.FREE, L_{mod}.inv)$ B.5.5 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (L_{op}.OPEN, L_{mod}.inv)$ B.5.3 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (L_{op}.RESET, L_{mod}.inv)$ B.5.2 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (L_{op}.SET, L_{mod}.inv)$ B.5.4 ⟩ This code is used in section B.1.6.10.
⟨ Proof of $val\_op\ (T_{op}.CHECKIN, L_{mod}.inv)$. B.6.5 ⟩ This code is used in section B.1.7.2.
⟨ Proof of $val\_op\ (T_{op}.CHECKOUT, L_{mod}.inv)$. B.6.4 ⟩ This code is used in section B.1.7.2.
⟨ Proof of $val\_op\ (T_{op}.FREE, L_{mod}.inv)$. B.6.3 ⟩ This code is used in section B.1.7.2.
⟨ Proof of $val\_op\ (T_{op}.OPEN, L_{mod}.inv)$. B.6.1 ⟩ This code is used in section B.1.7.2.
⟨ Proof of $val\_op\ (T_{op}.RESET, L_{mod}.inv)$. B.6 ⟩ This code is used in section B.1.7.2.

⟨ Proof of $val\_op\ (T_{op}.SET, L_{mod}.inv)$. B.6.2 ⟩ This code is used in section B.1.7.2.
⟨ Proof of $val\_retr\ (D_{mod}.inv, K_{mod}.inv, retr_D)$. B.4.2 ⟩ This code is used in section B.1.4.10.
⟨ Propositions B.9.2 ⟩ This code is used in section B.10.
⟨ Quantifiers B.9.4 ⟩ This code is used in section B.10.
⟨ Range of a Map B.9.8.6 ⟩ This code is used in section B.9.8.5.
⟨ Read object from sequence. B.8.5.30 ⟩ This code is used in section B.8.5.27.
⟨ Relating Sequences and Maps B.9.8.12 ⟩ This code is used in section B.9.8.5.
⟨ Release a lock. B.1.6.6 ⟩ This code is used in section B.1.6.2.
⟨ Reset the system (delta technique). B.1.4.4 ⟩ This code is used in section B.1.4.3.
⟨ Reset the system (global array). B.1.5.5 ⟩ This code is used in section B.1.5.4.
⟨ Reset the system (locks). B.1.6.3 ⟩ This code is used in section B.1.6.2.
⟨ Reset the system. B.1.3.3 ⟩ This code is used in section B.1.3.2.
⟨ Result case (RESET). B.4.3.1 ⟩ This code is used in section B.4.3.
⟨ Result case (*CHECKIN*). B.4.6.3 ⟩ This code is used in section B.4.6.
⟨ Result case (*CHECKOUT*). B.4.5.1 ⟩ This code is used in section B.4.5.
⟨ Result case (*OPEN*). B.4.4.2 ⟩ This code is used in section B.4.4.
⟨ Result lemma I. B.4.6.9 ⟩ This code is used in section B.4.6.4.
⟨ Result lemma II. B.4.6.10 ⟩ This code is used in section B.4.6.4.
⟨ Result lemma III. B.4.6.11 ⟩ This code is used in section B.4.6.4.
⟨ Result lemmas (*CHECKIN*:result). B.4.6.4 ⟩ This code is used in section B.4.6.3.
⟨ Retrieve condition. B.8.3.3 ⟩ This code is used in section B.8.3.
⟨ Retrieve function (delta technique → files). B.1.4.2 ⟩ This code is used in section B.1.4.
⟨ Retrieve function (global array → delta technique). B.1.5.3 ⟩ This code is used in section B.1.5.
⟨ Retrieve lemma. B.4.1.1 ⟩ This code is used in section B.1.4.10.
⟨ Retrieve map lemma. B.4.4.3 ⟩ This code is used in section B.4.4.2.
⟨ Rewriting Laws of Equality B.9.1.5 ⟩ This code is used in section B.9.1.2.
⟨ Sequence Construction B.9.7.1 ⟩ This code is used in section B.9.7.
⟨ Sequences B.9.7 ⟩ This code is used in section B.10.
⟨ Sequential Join of two Sequences B.9.7.4 ⟩ This code is used in section B.9.7.3.
⟨ Set Construction B.9.6.1 ⟩ This code is used in section B.9.6.
⟨ Set Membership B.9.6.4 ⟩ This code is used in section B.9.6.3.
⟨ Set of Elements occuring in a Sequence B.9.7.10 ⟩ This code is used in section B.9.7.3.
⟨ Side deduction (OPEN). B.2.2.7 ⟩ This code is used in section B.2.2.6.
⟨ Side deduction A (D:CHECKIN) B.3.4.10 ⟩ This code is used in section B.3.4.9.
⟨ Side deduction A (D:OPEN). B.3.2.8 ⟩ This code is used in section B.3.2.6.
⟨ Side deduction B (D:CHECKIN) B.3.4.12 ⟩ This code is used in section B.3.4.11.
⟨ Side deduction B (D:OPEN). B.3.2.7 ⟩ This code is used in section B.3.2.6.
⟨ Side deduction. B.4.1.2 ⟩ This code is used in section B.4.1.1.
⟨ Side deductions (*CHECKIN*). B.4.6.1 ⟩ This code is used in section B.4.6.
⟨ Signature extension. B.8.4.9 ⟩ This code is used in section B.8.4.5.
⟨ Signature initialisation. B.8.4.6 ⟩ This code is used in section B.8.4.5.
⟨ Signature modifications. B.8.4.5 ⟩ This code is used in section B.8.4.2.
⟨ Signature of modules. B.8.2.2 ⟩ This code is used in section B.8.2.
⟨ Signature of operations. B.8.1.1 ⟩ This code is used in section B.8.1.
⟨ Signatures and Precedences B.9.5.4 ⟩ This code is used in section B.9.5.3.
⟨ Some auxiliary deductions (D:CHECKIN). B.3.4.1 ⟩ This code is used in section B.3.4.
⟨ Specification of labelled trees. B.8.5.14 ⟩ This code is used in section B.8.5.
⟨ Specification of mappings. B.8.5.1 ⟩ This code is used in section B.8.5.
⟨ Specification of sequences. B.8.5.25 ⟩ This code is used in section B.8.5.
⟨ State and invariant (delta technique). B.1.4.1 ⟩ This code is used in section B.1.4.
⟨ State and invariant (global array). B.1.5.1 ⟩ This code is used in section B.1.5.
⟨ State and invariant (locks). B.1.6.1 ⟩ This code is used in section B.1.6.
⟨ State and invariant (mappings). B.8.5.2 ⟩ This code is used in section B.8.5.1.

⟨ State and invariant (sequences). B.8.5.26 ⟩ This code is used in section B.8.5.25.
⟨ State and invariant (trees). B.8.5.15 ⟩ This code is used in section B.8.5.14.
⟨ Strengthening of precondition. B.8.4.12 ⟩ This code is used in section B.8.4.2.
⟨ Subset Relation B.9.6.9 ⟩ This code is used in section B.9.6.3.
⟨ Substitution Principle for Propositions B.9.2.3 ⟩ This code is used in section B.9.2.
⟨ Substitution Principle of Equality B.9.1.1 ⟩ This code is used in section B.9.1.
⟨ Test if map is empty. B.8.5.6 ⟩ This code is used in section B.8.5.3.
⟨ Test if object is in domain of map. B.8.5.7 ⟩ This code is used in section B.8.5.3.
⟨ Test if object occurs in tree. B.8.5.20 ⟩ This code is used in section B.8.5.16.
⟨ Test of Duplicates B.9.9.5 ⟩ This code is used in section B.9.9.3.
⟨ Test of Well-Formedness of Delta Units B.7.2.5 ⟩ This code is used in section B.7.2.3.
⟨ Test of Well-Formedness of Deltas B.7.2.12 ⟩ This code is used in section B.7.2.11.
⟨ The devlopment of a revision management system. B.1.1 ⟩ This code is used in section B.10.
⟨ Tree Construction B.9.9.1 ⟩ This code is used in section B.9.9.
⟨ Tree Insertion B.9.9.6 ⟩ This code is used in section B.9.9.3.
⟨ Trees B.9.9 ⟩ This code is used in section B.10.
⟨ Union of a Set of Sets B.9.6.12 ⟩ This code is used in section B.9.6.3.
⟨ Union of two Maps B.9.8.7 ⟩ This code is used in section B.9.8.5.
⟨ Union of two Sets B.9.6.5 ⟩ This code is used in section B.9.6.3.
⟨ Validity lemmas (locks). B.5.1.1 ⟩ This code is used in section B.1.6.10.
⟨ Validity of the data reification (delta technique). B.1.4.10 ⟩ This code is used in section B.1.4.
⟨ Validity of the data reification (global array). B.1.5.11 ⟩ This code is used in section B.1.5.
⟨ Validity of the kernel specification. B.1.3.8 ⟩ This code is used in section B.1.3.
⟨ Validity of the module (mappings). B.8.5.13 ⟩ This code is used in section B.8.5.1.
⟨ Validity of the module (sequences). B.8.5.34 ⟩ This code is used in section B.8.5.25.
⟨ Validity of the module (trees). B.8.5.24 ⟩ This code is used in section B.8.5.14.
⟨ Validity of the operations (mappings). B.8.5.11 ⟩ This code is used in section B.8.5.1.
⟨ Validity of the operations (sequences). B.8.5.32 ⟩ This code is used in section B.8.5.25.
⟨ Validity of the operations (trees). B.8.5.22 ⟩ This code is used in section B.8.5.14.
⟨ Validity of the specification (delta technique). B.1.4.9 ⟩ This code is used in section B.1.4.
⟨ Validity of the specification (global array). B.1.5.10 ⟩ This code is used in section B.1.5.
⟨ Validity of the specification (locks). B.1.6.10 ⟩ This code is used in section B.1.6.
⟨ Validity of the specification (total operations). B.1.7.2 ⟩ This code is used in section B.1.7.
⟨ Vertical development: reification. B.8.3 ⟩ This code is used in section B.8.

B.12 Index of variables

+: B.7.2.12, B.7.2.13, B.7.2.15, *B.9.5.4*, B.9.5.5, B.9.7.13, B.9.7.14, B.9.8.12.
-: B.7.2.13, *B.9.5.4*, B.9.5.5, B.9.7.12.
$-$: B.2.1.1, B.2.2.1, B.3.1.1, B.3.2.1, B.3.4.2.
$<$: B.1.5.7, B.1.6.14, B.5.4.1, B.7.2.12, B.8.5.5, B.8.5.30, *B.9.5.6*.
$=$: *B.9.1*.
$>$: B.1.3.5, B.1.4.6, B.1.6.16, B.4.6.1, B.7.2.5, *B.9.5.6*, B.9.5.7, B.9.7.12.
$\oplus$: B.4.6.6, B.7.1, B.7.2.13, B.7.2.14, B.7.2.17.
$\oplus_s$: B.1.4.2, B.4.1.1, B.4.4.3, B.4.6.6, B.7.2.14, B.7.2.17.
$\oplus_u$: B.4.4.3, B.7.2.6, B.7.2.8, B.7.2.13, B.7.2.17.
$\mapsto$: B.1.3.1, B.1.3.5, B.1.4.1, B.1.4.6, B.1.5.1, B.1.5.7, B.1.6.1, B.1.6.14, B.1.6.16, B.2.4.1, B.2.4.4, B.2.4.6, B.2.4.8, B.2.4.9, B.2.4.10, B.3.4, B.3.4.4, B.3.4.6, B.3.4.8, B.3.4.10,

B.3.4.12, B.4.1.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.8, B.5.4.1, B.5.4.3, B.5.6.1, B.5.6.2, B.7.2.2, B.7.2.13, B.8.5.5, *B.9.3.1*, B.9.3.2, B.9.6.1, B.9.6.14, B.9.7.1, B.9.8.1, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.11, B.9.8.12, B.9.8.13.

$\otimes$: B.1.3.1, B.1.3.4, B.1.3.5, B.1.4.1, B.1.4.5, B.1.4.6, B.1.5.1, B.1.5.6, B.1.5.7, B.1.6.1, B.1.6.4, B.1.6.5, B.1.6.7, B.1.6.13, B.1.6.14, B.1.6.16, B.1.7, B.2.2, B.2.4, B.3.2, B.3.4, B.4.4, B.4.6, B.5.3.1, B.5.4, B.5.4.1, B.5.6, B.5.6.1, B.5.8, B.7.2.2, B.8.4.3, B.8.4.9, B.8.4.11, B.8.4.14, B.8.5.5, B.8.5.19, *B.9.3.1*, B.9.6.14, B.9.8.1, B.9.8.12.

$(\cdot)^{CPRE}$: B.1.7, B.8.4.13, B.8.4.14.

$\wedge$: B.1.3.1, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.1, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.2, B.1.5.7, B.1.5.8, B.1.6.1, B.1.6.7, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.3.2, B.2.4.7, B.2.4.8, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.3.2, B.3.4.3, B.3.4.7, B.3.4.8, B.3.4.9, B.3.4.11, B.4.2, B.4.4, B.4.4.1, B.4.5.1, B.5.1.1, B.5.2.1, B.5.3.1, B.5.4.1, B.5.5.1, B.5.6, B.5.6.1, B.5.7.1, B.5.8, B.7.1, B.7.2.5, B.7.2.12, B.8.2.3, B.8.3.2, B.8.3.3, B.8.4.3, B.8.4.4, B.8.4.11, B.8.4.12, B.8.4.14, B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.10, B.8.5.20, B.8.5.21, B.8.5.30, B.8.5.31, *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.7, B.9.2.9, B.9.3.2, B.9.4.4, B.9.6.14, B.9.7.2, B.9.7.12, B.9.8.8, B.9.9.5.

$\bigcap$: B.9.6.13, B.9.9.5.

$\bigcup$: B.1.5.2, B.9.6.12, B.9.9.4, B.9.9.5.

$\cap$: B.9.6.6, B.9.6.7, B.9.6.10, B.9.6.13.

$\cup$: B.9.6.5, B.9.6.6, B.9.6.12, B.9.6.14.

$\exists_2$: B.2.1.1, B.2.2.1, B.2.3.1, B.2.4, B.2.4.2, B.3.1.1, B.3.2.1, B.3.3.1, B.3.4.2, B.8.1.2, *B.9.4.3*, B.9.4.4.

Δ: B.1.4.1, B.1.4.2, B.1.5.1, B.1.5.2, B.1.5.3, B.3.1.3, B.3.1.5, B.3.1.6, B.4.1.1, B.4.3.1, B.4.6.6, B.4.6.8, B.7.1, B.7.2.10, B.7.2.12, B.7.2.13, B.7.2.14, B.7.2.15, B.7.2.16, B.7.2.17, B.7.2.18.

$\Leftrightarrow$: B.2.1.4, B.2.1.5, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.4, B.3.1.5, B.3.2.4, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.11, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.5.1.1, B.7.2.5, B.8.4.14, B.8.4.16, B.8.5.6, B.8.5.7, B.8.5.20, *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.7, B.9.2.9, B.9.6.4, B.9.6.14, B.9.9.5.

$\exists$: B.4.2, B.8.3.3, B.9.4.1, B.9.4.2, B.9.4.3, B.9.4.4.

$\forall_2$: *B.9.4.3*.

$\forall$: B.1.3.1, B.1.4.1, B.1.5.2, B.2.1.2, B.2.1.6, B.2.2.2, B.2.4.3, B.2.4.7, B.3.1.2, B.3.1.6, B.3.2.2, B.3.4.3, B.3.4.7, B.7.1, B.7.2.12, *B.9.4.1*, B.9.4.2, B.9.4.3, B.9.4.4, B.9.6.9, B.9.6.14.

def_wff_{Δ}: B.7.2.12.

$def\_wff_{\Delta_u}$: B.7.2.5.

$\geq$: B.1.2, B.1.3.1, B.1.4.1, B.1.5.2, B.2.1.2, B.2.1.5, B.2.2.2, B.2.2.5, B.2.4.3, B.2.4.6, B.3.1.2, B.3.1.5, B.3.2.2, B.3.2.5, B.3.4.3, B.3.4.6, B.4.2.4, B.8.5.2, *B.9.5.6*, B.9.5.7, B.9.6.14.

$\Rightarrow$: B.1.3.1, B.1.4.1, B.1.5.2, B.2.1.2, B.2.1.6, B.2.2.2, B.2.2.6, B.2.4.3, B.2.4.7, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.4.3, B.3.4.7, B.4.2.5, B.7.1, B.7.2.5, B.7.2.12, B.8.5.7, B.8.5.20, *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.6, B.9.2.9, B.9.6.9.

$\in$: B.1.3.1, B.1.3.5, B.1.3.6, B.1.4.1, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.2, B.1.5.7, B.1.5.8, B.1.6.7, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.1.2, B.2.1.6, B.2.2.2, B.2.2.6, B.2.2.7, B.2.4.1, B.2.4.3, B.2.4.7, B.2.4.8, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.2.8, B.3.4.1, B.3.4.3, B.3.4.7, B.3.4.8, B.4.1.1, B.4.1.2, B.4.2.5, B.4.5, B.4.6.1, B.4.6.7, B.4.6.11, B.5.4.1, B.5.6, B.7.1, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.20, B.8.5.21, B.9.6.4, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.9, B.9.6.14, B.9.8.8, B.9.8.13, B.9.9.7, B.9.9.8.

$\leq$: B.7.1, B.7.2.12, B.8.5.30, *B.9.5.6*, B.9.7.12.

$(\cdot) \xrightarrow{m} (\cdot)$: B.1.3.1, B.1.4.1, B.1.4.2, B.1.5.1, B.1.5.3, B.1.6.1, B.2.1.3, B.2.1.5, B.2.1.6, B.3.1.3, B.3.1.5, B.3.1.6, B.4.1.1, B.4.3.1, B.4.6.8, B.5.8, B.7.2.16, B.8.5.2, B.8.5.4, B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10, B.8.5.12, *B.9.8.1*, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.10, B.9.8.11, B.9.8.12, B.9.8.13.

$\neq$: B.2.4.7, B.2.4.8, B.3.4.7, B.3.4.8, *B.9.2.11*, B.9.6.14, B.9.7.12, B.9.8.3, B.9.8.8, B.9.9.6, B.9.9.7.

$\not\Rightarrow$: *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.10.

$\notin$: B.1.3.5, B.1.6.5, B.1.6.14, B.2.4.1, B.3.4.1, B.4.1.1, B.4.6.1, B.4.6.4, B.4.6.11, B.5.4, B.5.4.1, B.8.5.5, B.8.5.7, B.8.5.19, B.8.5.20, B.9.6.4, B.9.6.6, B.9.6.7, B.9.6.8, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.9, B.9.8.13, B.9.9.5, B.9.9.8.

$\neg$: B.1.4.6, B.1.5.7, B.1.6.15, B.1.6.16, B.5.5.1, B.5.5.4, B.7.2.5, B.7.2.6, B.8.4.13, B.8.5.9, *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.6, B.9.2.11, B.9.5.1, B.9.5.6, B.9.6.4, B.9.6.11, B.9.7.8, B.9.8.9.

$\vee$: B.2.4.8, B.3.4.8, B.8.4.4, *B.9.2.1*, B.9.2.2, B.9.2.4, B.9.2.8, B.9.6.4, B.9.6.14, B.9.7.12.

$\subseteq$: B.1.6.1, B.5.1.1, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.3, B.5.5.1, B.5.5.3, B.5.5.4, B.5.6.1, B.5.7.1, B.9.6.9, B.9.6.14, B.9.8.13, B.9.9.8.

$\times$: *B.9.5.4*, B.9.5.5, B.9.7.6.

$\_$: B.1.3.3, B.1.3.4, B.1.3.5, B.1.4.4, B.1.4.5, B.1.4.6, B.1.5.5, B.1.5.6, B.1.5.7, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.2.1.1, B.2.2.1, B.2.4.2, B.3.1.1, B.3.2.1, B.3.4.2, B.8.1.1, B.8.4.8, B.8.5.4, B.8.5.6, B.8.5.10, B.8.5.17, B.8.5.18, B.8.5.28, B.8.5.29.

Δ_u: B.7.2.2, B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.10, B.7.2.13, B.7.2.15, B.7.2.16, B.7.2.17.

elems: B.1.5.2, B.4.1.1, B.4.1.2, B.4.6.4, B.4.6.11, B.7.1, *B.9.7.10*, B.9.8.13, B.9.9.4, B.9.9.5, B.9.9.8.

flatten: B.1.5.2, B.1.5.7, *B.9.7.9*.

$op_{fun}((\cdot),(\cdot))$: B.8.1.1, B.8.4.6.

hd: *B.9.7.5*, B.9.9.7.

$op_{in}((\cdot),(\cdot))$: B.1.3.4, B.1.3.5, B.1.4.5, B.1.4.6, B.1.5.6, B.1.5.7, B.1.6.4, B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.7, B.5.8, B.8.1.1, B.8.4.6, B.8.5.5, B.8.5.9, B.8.5.18, B.8.5.19, B.8.5.29.

len: B.1.5.6, B.1.5.7, B.7.1, B.7.2.7, B.7.2.12, B.7.2.13, B.7.2.15, B.8.5.30, B.8.5.31, *B.9.7.6*, B.9.7.12, B.9.7.13, B.9.8.12.

∇: B.1.3.1, B.1.3.6, B.1.4.1, B.1.5.2, B.1.5.3, B.1.6.7, B.1.6.16, B.1.6.17, B.2.1.2, B.2.1.6, B.2.2.2, B.2.2.6, B.2.3, B.2.4.3, B.2.4.7, B.2.4.9, B.2.4.10, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.2.7, B.3.4.3, B.3.4.7, B.3.4.10, B.3.4.11, B.3.4.12, B.4.2.5, B.4.5.1, B.5.6, B.5.7.1, B.8.5.8, *B.9.8.8*, B.9.8.10, B.9.8.13.

$\odot$: B.1.3.5, B.1.4.6, B.1.5.7, B.1.6.14, B.1.6.16, B.2.4.1, B.2.4.4, B.2.4.6, B.2.4.8, B.2.4.9, B.2.4.10, B.3.4, B.3.4.4, B.3.4.6, B.3.4.8, B.3.4.10, B.3.4.12, B.4.1.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.8, B.5.4.1, B.5.4.3, B.5.6.1, B.5.6.2, B.8.5.5, *B.9.8.1*, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.11, B.9.8.12, B.9.8.13.

$\langle\rangle_{(\cdot)}$: B.8.2.1, B.8.2.3, B.8.3.2.

$\rhd$: B.1.6.15, B.5.5.1, B.5.5.4, B.8.5.9, *B.9.8.9*, B.9.8.13.

$\langle\ (\cdot),(\cdot),(\cdot)\ \rangle_{\Delta_u}$: B.1.4.5, B.1.5.6, B.3.2, B.3.2.3, B.3.2.5, B.3.2.6, B.3.2.7, B.3.2.8, B.4.4.2, B.4.4.3, B.7.2.2, B.7.2.7, B.7.2.8, B.7.2.16, B.7.2.18.

$*$: B.1.4.2, B.1.5.3, B.4.1.1, B.4.4.3, B.4.6.6, B.7.2.16, *B.9.8.10*, B.9.8.13.

$\odot$: B.1.3.7, B.1.4.8, B.1.5.9, B.1.6.9, B.1.7.1, B.8.2.1, B.8.2.3, B.8.2.4, B.8.3.2, B.8.5.12, B.8.5.23, B.8.5.33.

$\langle(\cdot)\rangle$: B.1.3.7, B.1.4.8, B.1.4.12, B.1.5.9, B.1.6.9, B.1.7.1, B.8.2.1, B.8.2.3, B.8.2.4, B.8.3.2, B.8.5.12, B.8.5.23, B.8.5.33.

$(\cdot) \sqsubseteq^{mod}_{(\cdot)} (\cdot)$: B.8.3.2, B.8.3.4.

$\langle(\cdot) \mapsto (\cdot)\rangle$: B.1.3.4, B.1.4.5, B.1.5.6, B.1.6.13, B.2.2, B.2.2.3, B.2.2.5, B.2.2.6, B.2.2.7, B.3.2, B.3.2.3, B.3.2.5, B.3.2.7, B.3.2.8, B.4.4.2, B.4.4.3, B.4.4.4, B.5.3.1, B.5.3.3, *B.9.8.1*, B.9.8.13.

$\langle\rangle$: B.1.3.3, B.1.4.4, B.1.5.5, B.1.6.12, B.1.6.13, B.2.1, B.2.1.3, B.2.1.5, B.2.1.6, B.3.1, B.3.1.3, B.3.1.5, B.3.1.6, B.4.3, B.4.3.1, B.5.2.1, B.5.3.1, B.5.8, B.8.5.4, B.8.5.6, *B.9.8.1*, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.9, B.9.8.11, B.9.8.12.

$\langle\rangle$: B.1.3.4, B.1.4.2, B.1.4.5, B.1.5.5, B.1.5.6, B.1.6.13, B.2.2, B.2.2.3, B.2.2.4, B.3.2, B.3.2.3, B.3.2.4, B.4.1.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.6.6, B.5.3.1, B.5.3.3, B.7.2.5, B.7.2.8, B.7.2.13, B.7.2.14, B.7.2.15, B.7.2.17, B.8.5.18, B.8.5.28, *B.9.7.1*, B.9.7.2, B.9.7.4, B.9.7.6, B.9.7.7, B.9.7.8, B.9.7.9, B.9.7.10, B.9.7.12, B.9.7.14, B.9.8.10, B.9.8.12, B.9.9.6, B.9.9.7, B.9.9.8.

$\{\}$: B.1.3.4, B.1.4.5, B.1.5.6, B.1.6.13, B.2.1.3, B.2.1.5, B.2.1.6, B.3.1.3, B.3.1.5, B.3.1.6, B.4.3.1, B.4.4, B.4.4.1, B.5.2.1, B.5.3.1, B.9.6.1, B.9.6.2, B.9.6.4, B.9.6.5, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.10, B.9.6.11, B.9.6.12, B.9.6.13, B.9.6.14, B.9.7.10, B.9.8.2, B.9.8.6, B.9.8.11, B.9.9.4, B.9.9.5.

τ: B.1.3.3, B.1.4.4, B.1.5.5, B.1.6.12, B.2.1, B.2.1.3, B.2.1.4, B.3.1, B.3.1.3, B.3.1.4, B.4.3, B.4.3.1, B.5.2.1, B.5.8, B.8.5.17, *B.9.9.1*, B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.

$\cup$: *B.9.8.7*, B.9.8.8, B.9.8.9, B.9.8.10.

$number_{\Delta}$: B.1.5.7, B.7.2.15.

$number_{\Delta_*}$: B.7.2.7, B.7.2.15.

$\bigwedge$: B.1.6.3, B.1.6.4, B.5.8, B.8.4.3, B.8.4.14.

okon : *B.9.9.2*, B.9.9.5.

$op_{out}((\cdot),(\cdot))$: B.8.1.1, B.8.4.6, B.8.4.8, B.8.4.14, B.8.5.6, B.8.5.10, B.8.5.31.

$=_{op}$: B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.5.8, B.8.4.1, B.8.4.3, B.8.4.4, B.8.4.8, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.15, B.8.4.16.

$\bigvee$: B.1.7, B.8.4.4, B.8.4.14.

$(\cdot) \sqsubseteq^{oplist}_{(\cdot),DVarg} (\cdot)$: B.1.4.12, B.8.3.2.

$(\cdot) \sqsubseteq^{op}_{(\cdot),DVarg} (\cdot)$: B.1.4.12, B.4.3, B.4.4, B.4.5, B.4.6, B.8.3.1, B.8.3.2.

$(\cdot) \sqsubseteq_{(\cdot)} (\cdot)$: B.1.4.10, B.1.5.11, B.8.3.4.

rng : B.1.5.2, *B.9.8.6*.

∇: B.7.2.12, B.8.5.30, *B.9.7.11*.

$\odot$: B.7.2.13, B.7.2.14, B.7.2.15, *B.9.7.1*, B.9.7.2, B.9.7.4, B.9.7.5, B.9.7.6, B.9.7.7, B.9.7.8, B.9.7.9, B.9.7.10, B.9.7.11, B.9.8.10, B.9.8.12.

$\rhd$: B.7.2.12, *B.9.7.8*, B.9.8.13, B.9.9.2, B.9.9.7.

$+\!\!+$: B.1.5.7, B.4.6.6, B.7.2.17, B.8.5.29, *B.9.7.4*, B.9.7.6, B.9.7.7, B.9.7.9, B.9.7.12, B.9.7.13, B.9.7.14, B.9.8.12, B.9.8.13, B.9.9.6, B.9.9.7, B.9.9.8.

$*$: B.1.5.2, B.1.5.7, B.7.2.13, B.7.2.16, *B.9.7.7*, B.9.7.8, B.9.7.10, B.9.8.13, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.

$\langle (\cdot) \rangle$: B.1.4.5, B.1.5.6, B.3.2, B.3.2.3, B.3.2.5, B.3.2.6, B.3.2.7, B.3.2.8, B.4.4.2, B.4.4.3, B.4.6.6, B.7.2.17, B.7.2.18, B.8.5.29, *B.9.7.1*, B.9.7.12, B.9.7.14, B.9.8.12, B.9.8.13, B.9.9.6, B.9.9.7, B.9.9.8.

$\odot$: B.2.4.4, B.2.4.6, B.2.4.8, B.3.4.4, B.3.4.6, B.3.4.8, B.4.6.8, B.5.4.3, B.9.6.1, B.9.6.2, B.9.6.4, B.9.6.5, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.10, B.9.6.11, B.9.6.12, B.9.6.13, B.9.6.14, B.9.7.10, B.9.8.2, B.9.8.6, B.9.8.11, B.9.9.4, B.9.9.8.

$\backslash$: B.9.6.7, B.9.8.2, B.9.8.6.

$\rhd$: B.9.6.11, B.9.6.14.

$*$: B.1.5.2, B.9.6.10, B.9.6.11, B.9.6.14, B.9.7.10.

$\{(\cdot)\}$: B.2.2.3, B.2.2.5, B.2.2.7, B.3.2.3, B.3.2.5, B.3.2.8, B.4.4.2, B.4.4.3, B.4.4.4, B.9.6.1, B.9.6.14, B.9.8.13, B.9.9.8.

$op_{st}((\cdot))$: B.1.3.3, B.1.4.4, B.1.5.5, B.1.6.3, B.1.6.12, B.8.1.1, B.8.5.4, B.8.5.17, B.8.5.28.

$\wedge_{PRE}$: B.1.6.5, B.1.6.7, B.8.4.12, B.8.4.14.

tl : *B.9.7.5*.

$op((\cdot),(\cdot),(\cdot))$: B.1.3.6, B.1.4.7, B.1.5.8, B.1.6.8, B.1.6.17, B.1.7, B.8.1.1, B.8.1.2, B.8.2.1, B.8.2.3, B.8.2.4, B.8.3.1, B.8.3.2, B.8.4.1, B.8.4.3, B.8.4.4, B.8.4.6, B.8.4.9, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.15, B.8.4.16, B.8.5.7, B.8.5.8, B.8.5.20, B.8.5.21, B.8.5.30.

void: B.1.3.3, B.1.3.4, B.1.3.5, B.1.4.4, B.1.4.5, B.1.4.6, B.1.5.5, B.1.5.6, B.1.5.7, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.2.1, B.2.1.1, B.2.1.2, B.2.2.1, B.2.2.2, B.2.4, B.2.4.2, B.2.4.3, B.3.1, B.3.1.1, B.3.1.2, B.3.2.1, B.3.2.2, B.3.4.2, B.3.4.3, B.4.3, B.4.4.2, B.4.6.3, B.5.2.1, B.5.3.1, B.5.4.1, B.5.5.1, B.5.6.1, B.5.8, B.8.1.1, B.8.5.4, B.8.5.5, B.8.5.6, B.8.5.9, B.8.5.10, B.8.5.17, B.8.5.18, B.8.5.19,

B.8.5.28, B.8.5.29, B.8.5.31.
wff_{Δ}: B.1.4.1, B.1.5.2, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.4.3, B.3.4.7, B.3.4.9, B.3.4.11, B.7.1, B.7.2.12, B.7.2.17.
wff_{Δ_u}: B.3.2.6, B.7.2.5, B.7.2.8, B.7.2.12, B.7.2.17.
wff_F: B.1.3.1, B.1.3.4, B.1.3.5, B.1.4.1, B.1.4.5, B.1.4.6, B.1.4.9, B.2.1.2, B.2.1.6, B.2.2.2, B.2.2.6, B.2.4.3, B.2.4.7, B.2.4.9, B.2.4.10, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.4.1, B.3.4.3, B.3.4.7, B.3.4.9, B.3.4.11, B.4.2.5, B.4.4, B.4.4.1, B.7.1.
wff_L: B.7.1.
A_{inv}: B.1.5.1, B.1.5.9.
A_{mod}: B.1.5.9, B.1.5.10, B.1.5.11.
A_{op}: B.1.5.4, B.1.5.9, B.1.5.10.
A_{st}: B.1.5.1, B.1.5.2, B.1.5.3, B.1.5.5, B.1.5.6, B.1.5.7, B.1.5.8.
$A\_CHECKIN_{val}$: B.1.5.10.
$A\_CHECKOUT_{val}$: B.1.5.10.
A_OPEN_{val}: B.1.5.10.
A_RESET_{val}: B.1.5.10.
A_valid: B.1.5.10.
abs_to_map: B.4.3.1, B.4.6.8, *B.9.8.11*.
abs_to_map_prop: B.4.2.1, B.4.2.5, B.4.4.1, B.4.4.3, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.8, *B.9.8.13*.
absorp: B.9.6.2.
abstract: B.1.4.10, B.8.3.4.
access: *B.9.7.11*.
add: *B.9.5.5*, B.9.6.6, B.9.6.7.
ainv: B.8.3.3.
and: B.1.4.12, B.1.6.10, B.1.7.2, B.2.3.1, B.3.3.1, B.4.2, B.4.4.1, B.4.5.1, B.5.1.1, B.5.4.1, B.5.5.1, B.8.2.4, B.8.5.13, *B.9.2.4*.
and3: B.5.1.1, *B.9.2.7*.
and4: B.2.1.2, B.2.2.2, B.2.4.3, B.3.1.2, B.3.2.2, B.3.4.3, B.4.2, B.4.6.2, *B.9.2.7*.
any_imp_true: *B.9.2.6*.
app: B.2.2.6, B.2.4.9, B.2.4.10, B.3.2.7, B.3.4.10, B.3.4.12, *B.9.8.8*.
apply: B.4.2.5, B.4.4.3, B.4.5.1, B.7.2.8, B.9.8.13.
APPLY: B.8.5.3, B.8.5.11, B.8.5.12.
apply_to_inserts: B.1.5.3, B.7.2.16.
arr: B.1.5.1, B.1.5.2, B.1.5.3, B.1.5.7, B.1.5.8.
ast: B.8.3.3.
atm: B.1.4.2, B.1.5.3, B.4.1, B.4.2.1, B.4.2.5, B.4.3.1, B.4.4.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.4, B.4.6.5, B.4.6.8, *B.9.8.11*, B.9.8.13.
aux: B.2.4.10, B.4.2.1, B.4.2.2, B.4.2.4, B.4.2.5, B.4.3.1, B.4.4.1, B.5.5.1, B.5.5.3.
aux_1: B.3.4.11.
aux_2: B.3.4.11.
aux1: B.2.4.1, B.2.4.4, B.2.4.5, B.3.4.1, B.3.4.4, B.3.4.5.
aux2: B.2.4.1, B.2.4.5, B.3.4.1, B.3.4.5.
base: B.9.5.5, B.9.6.5, B.9.6.6, B.9.6.7, B.9.7.11.
BasicDatatypes: B.10.
begin: B.9.7.13.
br: B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.
card: B.1.3.1, B.1.3.5, B.1.4.1, B.1.4.6, B.1.5.2, B.1.5.7, B.1.6.14, B.1.6.16, B.2.1.2, B.2.1.5, B.2.2.2, B.2.2.5, B.2.4.3, B.2.4.6, B.3.1.2, B.3.1.5, B.3.2.2, B.3.2.5, B.3.4.3, B.3.4.6, B.4.2.4, B.4.6.1, B.5.4.1, B.8.5.2, B.8.5.5, B.8.5.10, B.9.6.8, B.9.6.14.
card_prop: B.5.1.1, B.9.6.14.
case_a: B.2.4.7, B.2.4.9, B.3.4.7, B.3.4.10.
case_b: B.2.4.7, B.2.4.10, B.3.4.7, B.3.4.11, B.3.4.12.
cased: B.2.4.7, B.3.4.7, *B.9.2.8*.
changed: B.1.4.1, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.4.3, B.3.4.7, B.3.4.9, B.3.4.11, B.7.1, B.7.2.18.
Char: B.7.1.
char: B.9.8.13.
CHECKIN: B.1.3.2, B.1.3.7, B.1.3.8, B.1.4.3, B.1.4.8, B.1.4.9, B.1.4.12, B.1.5.4, B.1.5.9, B.1.5.10, B.1.6.2, B.1.6.7, B.1.6.9, B.1.6.10, B.1.6.16, B.1.7, B.1.7.1, B.1.7.2, B.2.4, B.2.4.2, B.3.4, B.3.4.2, B.4.6, B.4.6.2, B.4.6.3, B.4.6.5, B.5.6, B.5.6.1, B.6.5.
$CHECKIN_{val}$: B.1.3.8, B.5.6.
CHECKOUT: B.1.3.2, B.1.3.7, B.1.3.8, B.1.4.3, B.1.4.8, B.1.4.9, B.1.4.12, B.1.5.4, B.1.5.9, B.1.5.10, B.1.6.2, B.1.6.8, B.1.6.9, B.1.6.10, B.1.6.17, B.1.7, B.1.7.1, B.1.7.2, B.2.3, B.2.3.1, B.2.4, B.3.3, B.3.3.1, B.4.5, B.4.5.1, B.5.7, B.5.7.1, B.6.4.
$CHECKOUT_{val}$: B.1.3.8, B.5.7.
co: B.1.4.2, B.1.5.3.

co_post: B.5.3.1, B.5.4.1, B.5.4.3, B.5.5.1, B.5.5.3, B.5.6.1, B.5.7.1.
commut: B.9.6.2.
commute_map: <u>*B.9.8.3*</u>.
comp: B.9.2.9.
compl: B.2.4.7, B.3.4.7.
completeness: B.4.2, B.8.3.3.
concrete: B.1.4.10, B.8.3.4.
cons: B.1.6.10, B.1.7.2, B.2.4.8, B.3.4.8, B.8.2.3, B.8.2.4, B.8.5.13, B.9.6.10, B.9.6.11, B.9.6.14, B.9.7.6, B.9.7.7, B.9.7.8, B.9.8.9.
cons_inj: <u>*B.9.7.2*</u>.
cont: B.1.3.1, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.1, B.1.4.2, B.1.4.5, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.1, B.1.5.2, B.1.5.3, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.1, B.1.6.5, B.1.6.13, B.1.6.14, B.1.6.16, B.1.6.17, B.2.1.2, B.2.1.3, B.2.1.5, B.2.1.6, B.2.2.2, B.2.2.3, B.2.2.5, B.2.2.6, B.2.2.7, B.2.3, B.2.4.1, B.2.4.3, B.2.4.4, B.2.4.6, B.2.4.7, B.2.4.8, B.2.4.9, B.2.4.10, B.3.1.2, B.3.1.3, B.3.1.5, B.3.1.6, B.3.2.2, B.3.2.3, B.3.2.5, B.3.2.6, B.3.2.7, B.3.2.8, B.3.3, B.3.4, B.3.4.1, B.3.4.3, B.3.4.4, B.3.4.6, B.3.4.7, B.3.4.8, B.3.4.10, B.3.4.11, B.3.4.12, B.4.1, B.4.2, B.4.2.1, B.4.2.2, B.4.2.4, B.4.2.5, B.4.3.1, B.4.4, B.4.4.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.5, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.6, B.4.6.7, B.4.6.8, B.4.6.9, B.5.1.1, B.5.2.1, B.5.3.1, B.5.3.3, B.5.4, B.5.4.1, B.5.4.3, B.5.5.1, B.5.5.3, B.5.6.1, B.5.6.2, B.5.7.1.
$cont_{Do}$: B.4.4.2, B.4.4.4, B.4.6.3, B.4.6.4, B.4.6.6, B.4.6.7, B.4.6.8, B.4.6.9, B.4.6.10.
contra: <u>*B.9.2.6*</u>.
convert: B.1.4.11, B.4.2.
count_up: B.1.5.6, B.7.2.7, <u>*B.9.7.14*</u>.
CREATE: B.1.6.3, B.1.6.4, B.5.8, B.8.5.3, B.8.5.11, B.8.5.12, B.8.5.16, B.8.5.22, B.8.5.23, B.8.5.27, B.8.5.32, B.8.5.33.
cst: B.8.3.3.
cut: B.7.2.6, <u>*B.9.7.13*</u>.
D_{inv}: B.1.4.1, B.1.4.8, B.1.4.9.
D_{mod}: B.1.4.8, B.1.4.9, B.1.4.10, B.1.4.11, B.1.4.12, B.1.5.11, B.3.1, B.3.1.2, B.3.2, B.3.2.2, B.3.3, B.3.3.2, B.3.4, B.3.4.3, B.4.2, B.4.3, B.4.4, B.4.5, B.4.6.
D_{op}: B.1.4.3, B.1.4.8, B.1.4.9, B.1.4.12, B.3.1, B.3.1.1, B.3.2, B.3.2.1, B.3.3, B.3.3.1, B.3.4, B.3.4.2, B.4.3, B.4.4, B.4.4.1, B.4.5, B.4.6, B.4.6.2.
D_{reif}: B.1.4.10, B.1.4.12.
D_{st}: B.1.4.1, B.1.4.2, B.1.4.4, B.1.4.5, B.1.4.6, B.1.4.7, B.1.4.9, B.1.4.11, B.3.1, B.3.1.1, B.3.1.2, B.3.2, B.3.2.1, B.3.2.2, B.3.3, B.3.3.1, B.3.3.2, B.3.4, B.3.4.2, B.3.4.3, B.4.1, B.4.2, B.4.3, B.4.4, B.4.4.2, B.4.5, B.4.5.1, B.4.6, B.4.6.3.
$D\_CHECKIN_{reif}$: B.1.4.12.
$D\_CHECKIN_{val}$: B.1.4.9.
$D\_CHECKOUT_{reif}$: B.1.4.12.
$D\_CHECKOUT_{val}$: B.1.4.9.
D_OPEN_{reif}: B.1.4.12.
D_OPEN_{val}: B.1.4.9.
D_RESET_{reif}: B.1.4.12.
D_RESET_{val}: B.1.4.9.
D_valid: B.1.4.9, B.1.4.10.
dom: B.1.3.1, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.1, B.1.4.2, B.1.4.5, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.2, B.1.5.3, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.1, B.1.6.5, B.1.6.7, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.1.2, B.2.1.3, B.2.1.5, B.2.1.6, B.2.2.2, B.2.2.3, B.2.2.5, B.2.2.6, B.2.2.7, B.2.4.1, B.2.4.3, B.2.4.4, B.2.4.6, B.2.4.7, B.2.4.8, B.3.1.2, B.3.1.3, B.3.1.5, B.3.1.6, B.3.2.2, B.3.2.3, B.3.2.5, B.3.2.6, B.3.2.8, B.3.4.1, B.3.4.3, B.3.4.4, B.3.4.6, B.3.4.7, B.3.4.8, B.4.1, B.4.2.1, B.4.2.2, B.4.2.4, B.4.2.5, B.4.3.1, B.4.4, B.4.4.1, B.4.4.4, B.4.5, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.4, B.4.6.5, B.4.6.7, B.4.6.8, B.5.1.1, B.5.2.1, B.5.3.1, B.5.4, B.5.4.1, B.5.4.3, B.5.5.1, B.5.5.3, B.5.5.4, B.5.6, B.5.6.1, B.5.7.1, B.8.5.2, B.8.5.5, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10, <u>*B.9.8.2*</u>, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.13.
decomp: B.9.2.9.
def_apply_delta: B.7.2.13.
def_apply_delta_seq: B.7.2.14.
def_apply_unit: B.7.2.6.
def_biginter: B.9.6.13.
def_bigunion: B.9.6.12.
def_card: B.2.1.5, B.2.4.6, B.3.1.5, B.3.4.6, B.9.6.8.

def_changed: B.3.2.6, B.7.2.18.
def_compl_pre: B.8.4.13.
def_count_up: *B.9.7.14*.
def_info: B.2.1.3, B.3.1.3, *B.9.9.4*.
def_init_in: B.5.8, B.8.4.8.
def_init_path: B.4.4.3, *B.9.9.7*.
def_insert: *B.9.9.6*.
def_join: B.5.8, B.8.4.3.
def_nodup: B.2.1.4, B.3.1.4, *B.9.9.5*.
def_number_delta: B.7.2.15.
def_oplist_reif: B.1.4.12, B.8.3.2.
def_opor: B.8.4.4.
def_strengthen_pre: B.8.4.12.
def_subseq: *B.9.7.12*.
def_val_oplist: B.1.6.10, B.1.7.2, B.8.2.3, B.8.2.4, B.8.5.13.
def_xtnd_stl: B.8.4.11.
del: B.1.4.1, B.1.4.6, B.1.5.7, B.3.1.2, B.3.1.6, B.3.2.2, B.3.2.6, B.3.2.7, B.3.4, B.3.4.4, B.3.4.6, B.3.4.8, B.3.4.9, B.3.4.10, B.3.4.12, B.4.6.3, B.4.6.6, B.7.2.2, B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.12, B.7.2.13, B.7.2.16.
del_{num}: B.1.5.7.
del_1: B.3.4.3, B.3.4.7, B.3.4.9, B.3.4.10, B.3.4.11, B.3.4.12.
del_wff: B.3.4.1, B.3.4.9.
DELETE: B.1.6.6, B.8.5.3, B.8.5.11, B.8.5.12.
Deltas: B.1.4.1.
dep: B.1.3.1, B.1.3.5, B.1.4.1, B.1.4.2, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.1, B.1.5.2, B.1.5.3, B.1.5.7, B.1.5.8, B.1.6.1, B.1.6.16, B.2.1.2, B.2.1.3, B.2.1.4, B.2.2.2, B.2.2.3, B.2.2.4, B.2.4.1, B.2.4.3, B.2.4.4, B.2.4.5, B.3.1.2, B.3.1.3, B.3.1.4, B.3.2.2, B.3.2.3, B.3.2.4, B.3.3, B.3.4, B.3.4.1, B.3.4.3, B.3.4.4, B.3.4.5, B.4.1, B.4.2, B.4.2.2, B.4.2.3, B.4.3.1, B.4.4.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.5, B.4.6.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.6, B.4.6.7, B.4.6.8, B.4.6.10, B.4.6.11, B.5.6.1, B.5.6.2.
dep_{Do}: B.4.4.2, B.4.4.4, B.4.6.3, B.4.6.4, B.4.6.7, B.4.6.8, B.4.6.9, B.4.6.10.
diff: B.1.4.6, B.1.5.7, B.3.4, B.4.6.3, B.7.1, B.9.6.7.
dom: B.4.2.1, B.4.4.1, B.4.5.2, B.4.6.1, B.9.8.13.
dom_equal: B.4.6.1.
dom_lemma: B.4.5.
dom_prop: B.2.2.3, B.2.2.5, B.2.2.7, B.3.2.3, B.3.2.5, B.3.2.8, B.4.4.4, B.5.6.1, *B.9.8.13*.
domain: B.2.1.3, B.2.1.5, B.2.1.6, B.2.4.4, B.2.4.6, B.2.4.8, B.3.1.3, B.3.1.5, B.3.1.6, B.3.4.4, B.3.4.6, B.3.4.8, B.4.3, B.4.3.1, B.4.4, B.4.5, B.4.6, B.4.6.8, B.5.2.1, B.5.3.1, B.5.4.3, B.8.3.1, *B.9.8.2*.
down: B.2.1.4, B.2.1.5, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.4, B.3.1.5, B.3.2.4, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.11, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.5.2.2, B.5.3.2, B.5.4.4, B.5.4.5, B.5.5.5, B.5.6.3, B.5.7.2.
dp: B.1.4.2, B.1.5.3.
ds: B.7.2.14, B.7.2.17.
du: B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.13, B.7.2.15, B.7.2.16, B.7.2.17.
dummy: B.9.7.16.
*du*1: B.7.2.13.
elems: *B.9.7.10*.
empty: B.2.1.3, B.2.1.4, B.2.1.5, B.2.1.6, B.3.1.3, B.3.1.4, B.3.1.5, B.3.1.6, B.4.3.1, B.5.2.1, B.5.3.1, B.7.2.13, B.7.2.14, B.7.2.15, B.8.2.3, B.8.3.2, B.9.6.4, B.9.6.8, B.9.6.10, B.9.6.11, B.9.6.12, B.9.6.13, B.9.7.4, B.9.7.6, B.9.7.7, B.9.7.8, B.9.7.9, B.9.7.10, B.9.7.12, B.9.7.14, B.9.8.2, B.9.8.6, B.9.8.7, B.9.8.9, B.9.8.10, B.9.8.11, B.9.8.12, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.
EMPTY: B.8.5.3, B.8.5.11, B.8.5.12.
end: B.9.7.13.
Equality: B.9.1, B.10.
equiv: B.5.1.1, B.5.2.2, B.5.3.2, B.5.4.4, B.5.4.5, B.5.5.5, B.5.6.3, B.5.7.2, *B.9.2.4*.
equiv_prop: *B.9.2.9*.
eval_post: B.5.2.1, B.5.3.1, B.5.3.3, B.5.4.1, B.5.4.2, B.5.4.3, B.5.5.1, B.5.5.2, B.5.5.4, B.5.6.1, B.5.6.2, B.5.7.1.
eval_pre: B.5.4.1, B.5.4.3.
ex: *B.9.4.2*.
exists_pr: *B.9.4.3*.
ext: B.8.4.13.
exten: B.4.6.8, B.9.8.13.
extend: B.5.4.3, B.9.6.14.

ex2_eq_intro: B.2.1.1, B.2.2.1, B.2.4.2, B.3.1.1, B.3.2.1, B.3.4.2, <u>*B.9.4.4*</u>.
ex2_eq2_intro: B.2.3.1, B.3.3.1, <u>*B.9.4.4*</u>.
false: B.2.1.6, B.3.1.6, B.4.1.2, B.4.6.11, B.8.5.7, B.8.5.20, <u>*B.9.2.1*</u>, B.9.2.4, B.9.7.8, B.9.8.9.
false_out: B.2.1.6, B.3.1.6, <u>*B.9.2.4*</u>.
File: B.1.3.1, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.1, B.1.5.3, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.4, B.1.6.7, B.1.6.8, B.1.6.13, B.1.6.16, B.1.6.17, B.1.7, B.2.1.3, B.2.1.5, B.2.1.6, B.2.2, B.2.3.1, B.2.3.2, B.2.4, B.3.2, B.3.3.1, B.3.3.2, B.3.4, B.4.4, B.4.5.1, B.4.6, B.5.3.1, B.5.6, B.5.6.1, B.5.7.1, B.5.8, B.7.1.
Files: B.7.1, B.10.
filter: B.9.6.11, B.9.6.14.
FiniteMaps: B.9.8, B.10.
FiniteSets: B.9.6, B.10.
first: B.2.2.6, B.2.4.9, B.3.2.7, B.3.4.10, B.9.8.8.
flatten: <u>*B.9.7.9*</u>.
fold: B.2.4.4, B.3.4.4, B.4.5.1, B.4.6.5, B.4.6.8, <u>*B.9.1.5*</u>.
fold_left: <u>*B.9.1.5*</u>.
FREE: B.1.6.2, B.1.6.9, B.1.6.10, B.1.6.15, B.1.7, B.1.7.1, B.1.7.2, B.5.5, B.5.5.1, B.6.3.
f1: B.7.1.
f2: B.7.1.
geq_prop: B.2.1.5, B.3.1.5, <u>*B.9.5.7*</u>.
get_fd_cont: B.4.1, B.4.2.1, B.4.4.1, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.8.
get_fd_dep: B.4.1, B.4.2.2, B.4.2.3, B.4.6.5.
def_sel$_1$: B.2.1.3, B.2.1.5, B.2.1.6, B.2.2.3, B.2.2.5, B.2.2.6, B.2.2.7, B.2.4.4, B.2.4.6, B.2.4.8, B.2.4.9, B.2.4.10, B.3.1.3, B.3.1.5, B.3.1.6, B.3.2.3, B.3.2.5, B.3.2.7, B.3.2.8, B.3.4.4, B.3.4.6, B.3.4.8, B.3.4.10, B.3.4.12, B.4.1, B.4.2.5, B.4.3.1, B.4.4.3, B.4.4.4, B.4.6.9, B.5.3.3, B.5.6.2, <u>*B.9.3.2*</u>.
$def_s el_2$: B.2.1.3, B.2.1.4, B.2.2.3, B.2.2.4, B.2.4.4, B.2.4.5, B.3.1.3, B.3.1.4, B.3.2.3, B.3.2.4, B.3.4.4, B.3.4.5, B.4.1, B.4.3.1, B.4.4.3, B.4.4.4, B.4.6.10, <u>*B.9.3.2*</u>.
goal: B.2.1.4, B.2.1.5, B.2.1.6, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.7, B.3.1.4, B.3.1.5, B.3.1.6, B.3.2.4, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.7, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.4.5.1.
gth_prop: B.2.4.6, B.3.4.6, <u>*B.9.5.7*</u>.
head: B.9.7.5.
hide: B.4.2, <u>*B.9.4.4*</u>.
hyp: B.2.1.6, B.2.2.6, B.2.2.7, B.2.4.7, B.2.4.8, B.3.1.6, B.3.2.6, B.3.2.8, B.3.4.7, B.3.4.8, B.4.1.2, B.4.2.5, B.4.6.7, B.4.6.11, B.5.1.1.
hyp_aux: B.4.6.7.
hyp_inv: B.2.3, B.2.3.2, B.2.4, B.2.4.1, B.3.2, B.3.3, B.3.3.2, B.3.4, B.3.4.1, B.4.2, B.4.2.1, B.4.2.5, B.4.3, B.4.4, B.4.5, B.4.6, B.4.6.1, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.2, B.5.4.3, B.5.5.1, B.5.5.2, B.5.5.3, B.5.6.1, B.5.7.1.
hyp_inv_dom: B.2.4.1, B.2.4.4, B.3.4.1, B.3.4.4, B.4.2.1, B.4.2.2.
hyp_inv_max: B.4.2.1, B.4.2.4.
hyp_inv_unique: B.2.4.1, B.2.4.5, B.3.4.1, B.3.4.5, B.4.2.1, B.4.2.3.
hyp_inv_wff: B.2.4.1, B.2.4.10, B.3.4.1, B.3.4.11, B.4.2.1.
hyp_post: B.2.1.2, B.2.2.2, B.2.3.2, B.2.4.3, B.3.1.2, B.3.2.2, B.3.3.2, B.3.4.3, B.4.3, B.4.3.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.5.1, B.4.6.3, B.4.6.9, B.4.6.10, B.5.2.1, B.5.2.2, B.5.3.1, B.5.3.2, B.5.4.1, B.5.4.5, B.5.5.1, B.5.5.5, B.5.6.1, B.5.6.3, B.5.7.1, B.5.7.2.
hyp_pre: B.2.2, B.2.2.6, B.2.4, B.2.4.1, B.3.2, B.3.2.6, B.3.3, B.3.4, B.3.4.1, B.4.3, B.4.4, B.4.4.1, B.4.5, B.4.6, B.4.6.1, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.4, B.5.5.1, B.5.6.1, B.5.7.1.
hyp_pre_aux: B.4.5, B.4.5.1.
hyp_pre_new: B.2.4.1, B.2.4.6, B.3.4.1, B.3.4.6.
hyp_pre_old: B.2.4.1, B.3.4.1.
hyp_pre_safe: B.2.4.1, B.2.4.6, B.3.4.1, B.3.4.6.
hyp_pre_wff: B.2.4.1, B.2.4.9, B.3.4.1, B.3.4.9.
hypu: B.4.1.1.
hyp1: B.4.1.1.
hyp2: B.4.1.1, B.4.2.5.
hyp3: B.4.1.1, B.4.1.2.
iLeft: <u>*B.9.2.8*</u>.
imp: B.2.1.6, B.2.2.6, B.2.4.7, B.2.4.10, B.3.1.6, B.3.2.6, B.3.4.7, B.3.4.11,

B.4.2.5, <u>*B*.9.2.4</u>.
in: B.1.3.4, B.1.3.5, B.1.4.5, B.1.4.6, B.1.4.12, B.1.5.6, B.1.5.7, B.1.6.4, B.1.6.5, B.1.6.7, B.1.6.10, B.1.6.13, B.1.6.14, B.1.6.16, B.1.7.2, B.2.1, B.2.1.1, B.2.1.6, B.2.2, B.2.2.1, B.2.2.6, B.2.3.1, B.2.4, B.2.4.1, B.2.4.2, B.2.4.7, B.3.1, B.3.1.1, B.3.1.6, B.3.2, B.3.2.1, B.3.2.6, B.3.3.1, B.3.4, B.3.4.2, B.3.4.7, B.4.1.2, B.4.2, B.4.2.5, B.4.3, B.4.4, B.4.4.1, B.4.4.2, B.4.5.1, B.4.6, B.4.6.2, B.4.6.3, B.4.6.5, B.4.6.11, B.5.1.1, B.5.4, B.5.4.1, B.5.5.1, B.5.6, B.5.8, B.8.1.1, B.8.1.2, B.8.2.1, B.8.2.3, B.8.2.4, B.8.3.1, B.8.3.2, B.8.4.1, B.8.4.3, B.8.4.4, B.8.4.6, B.8.4.8, B.8.4.9, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.15, B.8.4.16, B.8.5.5, B.8.5.13, B.8.5.19, B.9.2.4, B.9.4.2, B.9.4.4.
in_1: B.8.2.4.
in_2: B.8.2.4.
in_3: B.8.2.4.
in_4: B.8.2.4.
indirect: <u>*B*.9.1.4</u>.
indirect_product: B.2.1.3, B.2.2.3, B.2.4.4, B.3.1.3, B.3.2.3, B.3.4.4, <u>*B*.9.1.4</u>.
info: B.1.3.1, B.1.4.1, B.1.5.2, B.2.1.2, B.2.1.3, B.2.2.2, B.2.2.3, B.2.4.1, B.2.4.3, B.2.4.4, B.3.1.2, B.3.1.3, B.3.2.2, B.3.2.3, B.3.4.1, B.3.4.3, B.3.4.4, B.4.1.1, B.4.1.2, B.4.2.2, B.4.6.1, B.4.6.7, B.4.6.11, B.8.5.19, B.8.5.20, B.8.5.21, <u>*B*.9.9.4</u>, B.9.9.5, B.9.9.7, B.9.9.8.
info_prop: B.2.2.3, B.2.4.4, B.3.2.3, B.3.4.4, <u>*B*.9.9.8</u>.
init: B.1.6.4, B.5.8, B.8.4.6, B.8.4.8, B.8.4.14.
init_path: B.1.4.2, B.4.1.1, B.4.1.2, B.4.4.3, B.4.6.4, B.4.6.6, B.4.6.11, B.8.5.21, <u>*B*.9.9.7</u>, B.9.9.8.
init_path_prop: B.4.1.1, B.4.6.6, <u>*B*.9.9.8</u>.
ins: B.1.5.2, B.1.5.7, B.7.2.2, B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.13, B.7.2.15, B.7.2.16.
INSERT: B.1.6.5, B.8.5.3, B.8.5.11, B.8.5.12, B.8.5.16, B.8.5.22, B.8.5.23, B.8.5.27, B.8.5.32, B.8.5.33.
insert: B.1.3.5, B.1.4.6, B.1.5.7, B.1.6.16, B.2.4.1, B.2.4.4, B.2.4.5, B.3.4, B.3.4.4, B.3.4.5, B.4.1.1, B.4.6.3, B.4.6.5, B.4.6.6, B.4.6.8, B.5.6.1, B.5.6.2, B.8.5.19, <u>*B*.9.9.6</u>, B.9.9.8.
inter: B.9.6.6.
inv: B.1.3.7, B.1.3.8, B.1.4.8, B.1.4.9, B.1.4.10, B.1.4.11, B.1.4.12, B.1.5.9, B.1.5.10, B.1.6.1, B.1.6.9, B.1.6.10, B.1.7.1, B.1.7.2, B.2.1, B.2.1.2, B.2.2, B.2.2.2, B.2.3, B.2.3.2, B.2.4, B.2.4.3, B.3.1, B.3.1.2, B.3.2, B.3.2.2, B.3.3, B.3.3.2, B.3.4, B.3.4.3, B.4.1.1, B.4.2, B.4.3, B.4.4, B.4.5, B.4.6, B.5.1.1, B.5.2, B.5.3, B.5.4, B.5.4.1, B.5.4.2, B.5.5, B.5.5.1, B.5.5.2, B.5.6, B.5.7, B.6, B.6.1, B.6.2, B.6.3, B.6.4, B.6.5, B.8.1.2, B.8.2.2, B.8.2.3, B.8.2.4, B.8.3.2, B.8.3.4, B.8.4.14, B.8.5.12, B.8.5.23, B.8.5.33, B.9.6.8, B.9.9.8.
inv_c: B.8.3.1, B.8.3.2, B.8.3.3.
inv_E: B.5.1.1, B.5.2, B.5.2.1, B.5.3, B.5.3.1, B.5.6, B.5.7.
inv_{KMAP}: B.5.1.1, B.5.2, B.5.3.
inv_{KMAPE}: B.5.1.1, B.5.2, B.5.3.
inv_{LMAP}: B.5.1.1, B.5.4, B.5.5.
inv_1: B.5.2, B.5.3, B.5.4, B.5.5, B.5.6, B.5.7, B.8.4.14.
inv_2: B.5.2, B.8.4.14.
inv_dom: B.4.6.1, B.4.6.7.
*inv_eqv*1: B.5.1.1, B.5.2, B.5.3.
*inv_eqv*2: B.5.1.1, B.5.4, B.5.5.
inv_hyp: B.4.2.
inv_new_D: B.4.6.1, B.4.6.6, B.4.6.7, B.4.6.11.
inv_old_D: B.4.6.1, B.4.6.6, B.4.6.7.
inv_unique: B.4.6.1, B.4.6.6, B.4.6.7.
invar: B.1.4.11, B.4.2.
inverse: B.1.4.11, B.4.2.
iRight: <u>*B*.9.2.8</u>.
join: B.1.4.12, B.4.6.6, B.8.3.2, <u>*B*.9.7.4</u>, B.9.8.13.
K_{inv}: B.1.3.1, B.1.3.7, B.2.4.
K_{mod}: B.1.3.7, B.1.3.8, B.1.4.10, B.1.4.11, B.1.4.12, B.1.6.1, B.2.1, B.2.1.2, B.2.2, B.2.2.2, B.2.3, B.2.3.2, B.2.4, B.2.4.3, B.4.2, B.5.1.1, B.5.4.1, B.5.4.2, B.5.5.1, B.5.5.2, B.5.6, B.5.7.
K_{op}: B.1.3.2, B.1.3.7, B.1.3.8, B.1.4.12, B.1.6.3, B.1.6.4, B.1.6.7, B.1.6.8, B.2.1, B.2.1.1, B.2.2, B.2.2.1, B.2.3, B.2.3.1, B.2.4, B.2.4.2, B.4.3,

B.4.4, B.4.4.2, B.4.5, B.4.5.1, B.4.6, B.4.6.3, B.4.6.5, B.5.8.
K_{st}: B.1.3.1, B.1.3.3, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.11, B.1.6.1, B.2.1, B.2.1.1, B.2.1.2, B.2.2, B.2.2.1, B.2.2.2, B.2.3, B.2.3.1, B.2.3.2, B.2.4, B.2.4.2, B.2.4.3, B.4.2.
K_valid: B.1.3.8, B.1.4.10.
L_{inv}: B.1.6.1, B.1.6.9, B.1.7.
L_{mod}: B.1.6.9, B.1.6.10, B.1.7.1, B.1.7.2, B.5.1.1, B.5.2, B.5.3, B.5.4, B.5.4.1, B.5.5, B.5.5.1, B.5.6, B.5.7, B.6, B.6.1, B.6.2, B.6.3, B.6.4, B.6.5.
L_{op}: B.1.6.2, B.1.6.9, B.1.6.10, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.1.7, B.5.2, B.5.2.1, B.5.3, B.5.3.1, B.5.4, B.5.4.1, B.5.5, B.5.5.1, B.5.6, B.5.6.1, B.5.7, B.5.7.1, B.5.8.
L_{st}: B.1.6.1, B.1.6.3, B.1.6.4, B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.8, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.1.7, B.5.1.1, B.5.2.1, B.5.3.1, B.5.4, B.5.4.1, B.5.5.1, B.5.6, B.5.6.1, B.5.7, B.5.7.1, B.5.8.
$L\_CHECKIN_{val}$: B.1.6.10, B.6.5.
L_CHECKIN_eval: B.1.6.16, B.5.6.3.
$L\_CHECKOUT_{val}$: B.1.6.10, B.6.4.
L_CHECKOUT_eval: B.1.6.17, B.5.7.2.
L_FREE_{val}: B.1.6.10, B.6.3.
L_FREE_eval: B.1.6.15, B.5.5.5.
L_OPEN_{val}: B.1.6.10, B.6.1.
L_OPEN_eval: B.1.6.13.
L_OPEN_eval_pr: B.5.3.2, B.5.8.
L_RESET_{val}: B.1.6.10, B.6.
L_RESET_eval: B.1.6.12.
L_RESET_eval_pr: B.5.2.2, B.5.8.
L_SET_{val}: B.1.6.10, B.6.2.
L_SET_eval: B.1.6.14, B.5.4.4, B.5.4.5.
L_valid: B.1.6.10.
l_1: B.9.7.2, B.9.7.4, B.9.8.13.
l_2: B.9.7.2, B.9.7.4, B.9.8.13.
last: B.4.6.6, B.9.9.8.
sel_1: B.1.3.1, B.1.3.4, B.1.3.5, B.1.4.1, B.1.4.5, B.1.4.6, B.1.5.1, B.1.5.6, B.1.5.7, B.1.6.1, B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.13, B.1.6.14, B.1.6.16, B.2.2, B.2.4.1, B.3.2, B.3.4, B.4.4, B.4.6, B.5.1.1, B.5.3.1, B.5.4, B.5.4.1, B.5.5, B.5.6, B.5.6.1, B.7.2.4, B.8.4.3, B.8.4.9, B.8.4.11, B.8.4.14, B.8.5.5, B.8.5.19, *B.9.3.1*, B.9.3.2, B.9.6.14.
lemma: B.4.1.1.
$lemma_{III}$: B.4.6.4, B.4.6.6.
$lemma_{IV}$: B.4.6.4, B.4.6.8.
$lemma_{V}$: B.4.6.4, B.4.6.8.
$lemma_{VI}$: B.4.6.4, B.4.6.5.
length: *B.9.7.6*.
LENGTH: B.8.5.27, B.8.5.32, B.8.5.33.
less_than: *B.9.5.6*.
let: B.9.6.6, B.9.6.7.
lhs: B.2.1.3, B.2.2.3, B.2.4.4, B.3.1.3, B.3.2.3, B.3.4.4, B.4.1.1, B.4.2.1, B.4.4.1, B.4.5.2, B.4.6.1, B.4.6.8.
Line: B.1.4.1, B.3.1.3, B.3.1.5, B.3.1.6, B.4.1.1, B.4.3.1, B.4.6.6, B.4.6.8, B.7.1.
LineLength: B.7.1.
ll: B.9.7.9.
locks: B.1.6.1, B.1.6.7, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.5.1.1, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.3, B.5.5.1, B.5.5.3, B.5.5.4, B.5.6, B.5.6.1, B.5.7.1, B.5.8.
l1: B.9.7.12, B.9.7.13.
l2: B.9.7.12, B.9.7.13.
m_i: B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10.
m_o: B.8.5.4, B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10.
m_1: B.9.8.7.
m_2: B.9.8.7.
map: B.9.6.14.
MAP_{inv}: B.8.5.2, B.8.5.11, B.8.5.12.
MAP_{mod}: B.5.1.1, B.8.5.12, B.8.5.13.
MAP_{op}: B.1.6.3, B.1.6.4, B.1.6.5, B.1.6.6, B.5.8, B.8.5.3, B.8.5.11, B.8.5.12.
MAP_{st}: B.8.5.2.
MAP_APPLY_val: B.8.5.11, B.8.5.13.
MAP_CREATE_val: B.5.2, B.5.3, B.8.5.11, B.8.5.13.
MAP_DELETE_val: B.5.5, B.8.5.11, B.8.5.13.
MAP_EMPTY_val: B.8.5.11, B.8.5.13.
map_induction: *B.9.8.3*.
MAP_INSERT_val: B.5.4, B.8.5.11, B.8.5.13.
map_lemma: B.4.4.2, B.4.4.4.
map_map: *B.9.8.10*.
map_map_prop: B.4.1.1, B.4.4.3, B.4.6.6, *B.9.8.13*.

MAP_SIZE_val: B.8.5.11, B.8.5.13.
MAP_TEST_val: B.8.5.11, B.8.5.13.
mapfilter: *B.9.8.9*.
MAPPINGS_int: B.1.6.2, B.8.5.1.
mapunion: *B.9.8.7*.
max: B.1.6.5, B.8.5.2, B.8.5.5, B.8.5.11, B.8.5.12, B.8.5.13.
member: B.2.1.6, B.3.1.6, B.9.6.4.
member_prop: B.2.2.7, B.2.4.8, B.3.2.8, B.3.4.8, B.9.6.14.
mfilter_subset: B.5.5.4, B.9.8.13.
mk: B.1.3.1, B.1.3.3, B.1.3.4, B.1.3.5, B.1.4.1, B.1.4.2, B.1.4.4, B.1.4.5, B.1.4.6, B.1.5.1, B.1.5.3, B.1.5.5, B.1.5.6, B.1.5.7, B.1.6.1, B.1.6.12, B.1.6.13, B.1.6.16, B.2.1, B.2.2, B.2.4.1, B.3.1, B.3.2, B.3.4, B.4.3, B.4.3.1, B.4.4.2, B.4.4.4, B.4.6.3, B.4.6.5, B.4.6.9, B.4.6.10, B.5.2.1, B.5.3.1, B.5.3.3, B.5.6.1, B.5.6.2, B.5.8.
mod: B.8.2.3.
mod_a: B.8.3.2, B.8.3.4.
mod_c: B.8.3.2, B.8.3.4.
mod_valid: B.1.3.8, B.1.4.9, B.1.5.10, B.1.6.10, B.8.2.3, B.8.2.4, B.8.3.4, B.8.5.13, B.8.5.24, B.8.5.34.
module: B.1.4.10, B.8.2.2, B.8.2.3, B.8.3.2, B.8.3.4.
Module_Interface_Library: B.8.5, B.10.
mult: *B.9.5.5*.
mutual: B.9.6.14.
nat: B.1.2, B.1.5.1, B.1.5.2, B.1.5.3, B.7.1, B.7.2.2, B.7.2.7, B.7.2.8, B.7.2.12, B.7.2.15, B.7.2.18, B.8.5.2, B.8.5.5, B.8.5.10, B.8.5.11, B.8.5.12, B.8.5.13, B.8.5.30, B.8.5.31, *B.9.5*, *B.9.5.1*, B.9.5.4, B.9.5.5, B.9.5.6, B.9.5.7, B.9.6.8, B.9.6.14, B.9.7.6, B.9.7.11, B.9.7.12, B.9.7.13, B.9.7.14, B.9.7.15, B.9.8.12.
nat_induction: *B.9.5.1*.
NaturalNumbers: B.9.5, B.10.
neg: B.9.5.6.
nequiv: *B.9.2.4*.
new: B.1.3.5, B.1.4.6, B.1.5.7, B.1.6.16, B.2.1, B.2.1.1, B.2.1.2, B.2.1.3, B.2.1.4, B.2.1.5, B.2.1.6, B.2.2, B.2.2.1, B.2.2.2, B.2.2.3, B.2.2.4, B.2.2.5, B.2.2.6, B.2.2.7, B.2.4.1, B.2.4.4, B.2.4.5, B.2.4.6, B.2.4.7, B.2.4.8, B.2.4.9, B.2.4.10, B.3.1, B.3.1.1, B.3.1.2, B.3.1.3, B.3.1.4, B.3.1.5, B.3.1.6, B.3.2, B.3.2.1, B.3.2.2, B.3.2.3, B.3.2.4, B.3.2.5, B.3.2.6, B.3.2.7, B.3.2.8, B.3.4, B.3.4.2, B.3.4.3, B.3.4.4, B.3.4.5, B.3.4.6, B.3.4.7, B.3.4.8, B.5.1.1, B.5.4.3, B.8.4.9, B.8.4.11, B.8.4.14, B.9.6.8, B.9.6.14.
newr: B.1.4.5, B.5.3.1, B.5.3.3.
no: B.9.6.11.
node: B.1.3.4, B.1.4.5, B.1.5.6, B.1.6.13, B.2.2, B.2.2.3, B.2.2.4, B.3.2, B.3.2.3, B.3.2.4, B.4.4.2, B.4.4.3, B.4.4.4, B.5.3.1, B.5.3.3, B.8.5.18, *B.9.9.1*, B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7, B.9.9.8.
nodup: B.1.3.1, B.1.4.1, B.1.5.2, B.2.1.2, B.2.1.4, B.2.2.2, B.2.2.4, B.2.4.3, B.2.4.5, B.3.1.2, B.3.1.4, B.3.2.2, B.3.2.4, B.3.4.3, B.3.4.5, B.4.1.1, B.4.2.3, B.8.5.15, *B.9.9.5*, B.9.9.8.
nodup_prop: B.2.2.4, B.2.4.5, B.3.2.4, B.3.4.5, *B.9.9.8*.
not: B.2.1.6, B.3.1.6, B.4.1.2, B.4.6.11, *B.9.2.4*.
nr: B.3.4, B.3.4.1, B.3.4.4, B.3.4.5, B.3.4.6, B.3.4.7, B.3.4.8, B.3.4.10, B.3.4.12, B.4.6, B.4.6.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.6, B.4.6.8, B.4.6.11, B.5.6.1, B.5.6.2.
observ: B.9.6.14.
ok_delta: B.1.4.1.
ol_{a1}: B.8.3.2.
ol_{a2}: B.8.3.2.
ol_{c1}: B.8.3.2.
ol_{c2}: B.8.3.2.
ol_1: B.8.2.3.
ol_2: B.8.2.3.
oldr: B.5.6.
op: B.5.8, B.8.1.2, B.8.2.3, B.8.4.8, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.15.
op_a: B.8.3.1, B.8.3.2.
op_c: B.8.3.1, B.8.3.2.
op_1: B.8.2.4, B.8.4.3, B.8.4.4, B.8.4.14, B.8.4.15, B.8.4.16.
op_2: B.8.2.4, B.8.4.3, B.8.4.4, B.8.4.14, B.8.4.15, B.8.4.16.
op_3: B.8.2.4, B.8.4.15.
op_4: B.8.2.4.
OPEN: B.1.3.2, B.1.3.7, B.1.3.8, B.1.4.3, B.1.4.8, B.1.4.9, B.1.4.12, B.1.5.4, B.1.5.9, B.1.5.10, B.1.6.2, B.1.6.4, B.1.6.9, B.1.6.10, B.1.6.13, B.1.7, B.1.7.1, B.1.7.2, B.2.2,

B.2.2.1, B.3.2, B.3.2.1, B.4.4, B.4.4.1, B.4.4.2, B.5.3, B.5.3.1, B.5.8, B.6.1.
$OPEN_{val}$: B.1.3.8, B.5.3.
oplist: B.8.2.1, B.8.2.2, B.8.2.3, B.8.3.2, B.8.5.12, B.8.5.23, B.8.5.33.
ops: B.1.3.7, B.1.4.8, B.1.4.12, B.1.5.9, B.1.6.9, B.1.7.1, B.1.7.2, B.8.2.2, B.8.2.3, B.8.2.4, B.8.3.2, B.8.5.12, B.8.5.23, B.8.5.33.
opval: B.8.2.4.
or: B.2.4.1, B.2.4.4, B.2.4.5, B.3.4, B.3.4.1, B.3.4.4, B.3.4.5, B.4.6, B.4.6.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.6, B.4.6.8, B.4.6.11, B.5.6.1, B.5.6.2, *B.9.2.4*.
OrderedPairs: B.9.3, B.10.
out: B.2.1.1, B.2.1.2, B.2.1.6, B.2.2.1, B.2.2.2, B.2.4, B.2.4.2, B.2.4.3, B.2.4.10, B.3.1.1, B.3.1.2, B.3.1.6, B.3.2.1, B.3.2.2, B.3.4.2, B.3.4.3, B.3.4.11, B.4.1.2, B.4.3, B.4.4.2, B.4.6.3, B.4.6.5, B.4.6.11, B.5.2.2, B.5.3.2, B.5.4.4, B.5.4.5, B.5.5.5, B.5.6.3, B.5.7.2, B.8.1.1, B.8.1.2, B.8.2.1, B.8.2.3, B.8.3.1, B.8.3.2, B.8.4.1, B.8.4.3, B.8.4.4, B.8.4.6, B.8.4.8, B.8.4.9, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.15, B.8.4.16, B.9.2.4, B.9.4.2, B.9.4.4.
out_1: B.8.2.4.
out_2: B.8.2.4.
out_3: B.8.2.4.
out_4: B.8.2.4.
overwrite_map: *B.9.8.3*.
p_1: B.9.2.3.
p_2: B.9.2.3.
pair_inj: *B.9.3.2*.
pair_prop: B.9.6.14.
paste: B.7.2.6, *B.9.7.13*.
PATH: B.8.5.16, B.8.5.22, B.8.5.23.
path_incl: B.4.1.2, B.4.6.11, *B.9.9.8*.
pLeft: B.2.4.1, B.3.4.1, B.3.4.9, B.3.4.11, B.4.2.1, B.4.4.1, B.4.5.1, B.4.6.1, B.5.1.1, B.5.3.3, B.5.4.1, B.5.4.2, B.5.5.1, B.5.5.2, B.5.6.2, B.5.7.1, *B.9.2.7*.
pos: B.7.2.2, B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.12, B.7.2.13, B.7.2.16, B.9.5.6, B.9.7.13.
pos_of: B.9.7.15.
post: B.1.3.3, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.4, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.5, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.1.1, B.2.2.1, B.2.3.1, B.2.4, B.2.4.2, B.3.1.1, B.3.2.1, B.3.3.1, B.3.4.2, B.4.3, B.4.4.2, B.4.5.1, B.4.6.3, B.4.6.5, B.5.2.1, B.5.2.2, B.5.3.1, B.5.3.2, B.5.4.1, B.5.4.5, B.5.5.1, B.5.5.5, B.5.6.1, B.5.6.3, B.5.7.1, B.5.7.2, B.5.8, B.8.1.1, B.8.1.2, B.8.3.1, B.8.4.3, B.8.4.4, B.8.4.8, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.16, B.8.5.4, B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10, B.8.5.17, B.8.5.18, B.8.5.19, B.8.5.20, B.8.5.21, B.8.5.28, B.8.5.29, B.8.5.30, B.8.5.31.
post_cont: B.4.6.4, B.4.6.8.
post_dep: B.4.6.4, B.4.6.5, B.4.6.8.
pre: B.1.3.3, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.4, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.5, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.2, B.2.3, B.2.4, B.3.2, B.3.3, B.3.4, B.4.3, B.4.4.1, B.4.5, B.4.6, B.4.6.2, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.4, B.5.5.1, B.5.6.1, B.5.7.1, B.5.8, B.8.1.1, B.8.1.2, B.8.3.1, B.8.4.3, B.8.4.4, B.8.4.8, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.16, B.8.5.4, B.8.5.5, B.8.5.6, B.8.5.7, B.8.5.8, B.8.5.9, B.8.5.10, B.8.5.17, B.8.5.18, B.8.5.19, B.8.5.20, B.8.5.21, B.8.5.28, B.8.5.29, B.8.5.30, B.8.5.31.
pre_card: B.4.6.1.
pre_card_D: B.4.6.1, B.4.6.2.
pre_new: B.4.6.1.
pre_new_D: B.4.6.1, B.4.6.2.
pre_old: B.4.6.1.
pre_old_D: B.4.6.1, B.4.6.2.
pre_rev: B.4.6.1.
pre_wff: B.4.6.1, B.4.6.2.
prefl: *B.9.2.9*.
preservation: B.2.1, B.2.2, B.2.3, B.2.4, B.3.1, B.3.2, B.3.3, B.3.4, B.4.2, B.8.1.2, B.8.3.3.
prev: B.1.3.5, B.1.4.6, B.1.5.7, B.1.6.7, B.1.6.16.
pRight: B.2.2.6, B.2.3.2, B.2.4.1, B.2.4.10, B.3.2.6, B.3.3.2, B.3.4.1, B.3.4.11, B.4.2.1, B.4.4.1, B.4.5.1, B.4.6.1, B.4.6.6, B.5.1.1, B.5.2.1,

B.5.3.1, B.5.4.3, B.5.5.3, B.5.5.4, B.5.6.1, B.5.7.1, <u>*B.9.2.7*</u>.
proj_opeq: B.5.2.2, B.5.3.2, B.5.4.4, B.5.4.5, B.5.5.5, B.5.6.3, B.5.7.2, B.8.4.16.
prop: B.7.2.5, B.7.2.12, B.8.1.1, B.8.1.2, B.8.2.2, B.8.2.3, B.8.2.4, B.8.3.1, B.8.3.2, B.8.3.3, B.8.4.1, B.8.4.12, B.8.4.14, B.8.4.15, B.8.5.6, B.8.5.7, B.8.5.20, <u>*B.9.1*</u>, B.9.1.1, B.9.1.5, B.9.2.1, B.9.2.3, B.9.2.4, B.9.2.6, B.9.2.7, B.9.2.8, B.9.2.9, B.9.2.10, B.9.4.1, B.9.4.2, B.9.4.3, B.9.4.4, B.9.5.1, B.9.5.6, B.9.6.2, B.9.6.4, B.9.6.9, B.9.6.11, B.9.6.14, B.9.7.2, B.9.7.8, B.9.8.3, B.9.8.9, B.9.8.13, B.9.9.2, B.9.9.5.
prop_apply_delta_seq: B.4.4.3, B.7.2.17.
*prop_apply_delta_seq*2: B.4.6.6, B.7.2.17.
prop_convert: B.1.4.11, B.4.2.
prop_diff: B.3.4.9, B.4.6.6, B.7.1.
prop_diff_wff: B.3.4.9, B.7.1.
prop_ok_delta: B.3.2.6, B.7.2.17.
prop_subst: <u>*B.9.2.3*</u>.
prop_unit: B.3.2.6, B.4.4.3, B.7.2.8.
Propositions: B.9.2, B.10.
prsubst: <u>*B.9.2.9*</u>.
psubst: <u>*B.9.2.9*</u>.
psym: <u>*B.9.2.9*</u>.
ptrans: <u>*B.9.2.9*</u>.
Quantifiers: B.9.4, B.10.
r_1: B.1.6.15.
range: <u>*B.9.8.6*</u>.
READ: B.8.5.27, B.8.5.32, B.8.5.33.
recur: B.2.4.4, B.2.4.6, B.2.4.8, B.2.4.10, B.3.4.4, B.3.4.6, B.3.4.8, B.3.4.12, B.4.4.3, B.4.6.8, B.5.4.3, B.7.2.13, B.7.2.14, B.7.2.15, B.9.5.5, B.9.5.6, B.9.6.4, B.9.6.5, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.12, B.9.6.13, B.9.7.4, B.9.7.9, B.9.7.10, B.9.7.11, B.9.7.12, B.9.7.14, B.9.8.2, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.10, B.9.8.11, B.9.8.12, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.
reduce: B.4.1.1, B.4.6.6, B.9.8.13.
refl: B.2.1.1, B.2.1.3, B.2.1.4, B.2.1.5, B.2.2.1, B.2.2.3, B.2.2.4, B.2.2.5, B.2.2.6, B.2.3.1, B.2.4.2, B.2.4.4, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.1, B.3.1.3, B.3.1.4, B.3.1.5, B.3.2.1, B.3.2.3, B.3.2.4, B.3.2.5, B.3.2.6, B.3.2.7, B.3.3.1, B.3.4.2, B.3.4.4, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.10, B.3.4.11, B.3.4.12, B.4.1, B.4.1.1, B.4.2.1, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.4.3.1, B.4.4.1, B.4.4.3, B.4.4.4, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.5, B.4.6.6, B.4.6.8, B.4.6.9, B.4.6.10, B.5.3.3, B.5.4.1, B.5.5.1, B.5.6.2, B.5.7.1, <u>*B.9.1.3*</u>.
refl_imp: <u>*B.9.2.6*</u>.
refl_opeq: B.5.8, B.8.4.15.
$reif_A$: B.1.5.11.
$reif_D$: B.1.4.10.
reification: B.1.4.10, B.8.3.4.
ReificationMethodology: B.8, B.10.
res: B.2.3, B.2.3.1, B.2.3.2, B.3.3, B.3.3.1, B.3.3.2.
RESET: B.1.3.2, B.1.3.7, B.1.3.8, B.1.4.3, B.1.4.8, B.1.4.9, B.1.4.12, B.1.5.4, B.1.5.9, B.1.5.10, B.1.6.2, B.1.6.3, B.1.6.9, B.1.6.10, B.1.6.12, B.1.7, B.1.7.1, B.1.7.2, B.2.1, B.2.1.1, B.2.1.2, B.2.2.2, B.3.1, B.3.1.1, B.4.3, B.5.2, B.5.2.1, B.5.8, B.6.
$RESET_{val}$: B.1.3.8, B.5.2.
result: B.4.3, B.4.4, B.4.5, B.4.6, B.8.3.1.
retr: B.4.2, B.4.2.1, B.4.2.5, B.8.3.1, B.8.3.2, B.8.3.3, B.8.3.4.
$retr_A$: B.1.5.3, B.1.5.11.
$retr_D$: B.1.4.2, B.1.4.10, B.1.4.11, B.1.4.12, B.4.1, B.4.2, B.4.2.5, B.4.3, B.4.4, B.4.4.2, B.4.5, B.4.5.1, B.4.6, B.4.6.3.
$retr_{Di}$: B.4.5, B.4.5.1, B.4.5.2, B.4.6.3, B.4.6.6, B.4.6.8.
$retr_{Do}$: B.4.4.2, B.4.4.3, B.4.4.4, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.8.
$retr_{DoR}$: B.4.6.3, B.4.6.4, B.4.6.6, B.4.6.8.
$retr_{map}$: B.4.4.2, B.4.4.3.
retr_abs: B.1.4.2, B.1.5.3.
retr_lemma: B.4.1.1, B.4.6.7.
$retr\_rev_A$: B.1.5.3, B.1.5.7, B.1.5.8.
$retr\_rev_D$: B.1.4.2, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.3, B.3.3, B.3.4, B.3.4.1, B.4.1, B.4.1.1, B.4.2, B.4.3.1, B.4.4.1, B.4.4.2, B.4.5, B.4.6.1, B.4.6.3, B.4.6.7, B.4.6.8.
retrieval: B.1.4.10, B.8.3.4.
RevMax: B.1.2, B.1.3.1, B.1.3.5, B.1.4.1, B.1.4.6, B.1.5.2, B.1.5.7, B.1.6.5, B.1.6.14, B.1.6.16, B.2.1.2,

B.2.1.5, B.2.2.2, B.2.2.5, B.2.4.3, B.2.4.6, B.3.1.2, B.3.1.5, B.3.2.2, B.3.2.5, B.3.4.3, B.3.4.6, B.4.2.4, B.4.6.1, B.5.1.1, B.5.4.1.

$RevMax_{pos}$: B.1.2, B.2.1.5, B.2.2.5, B.3.1.5, B.3.2.5.

rg: B.1.3.1, B.1.4.1, B.1.4.2, B.1.4.9, B.1.4.11, B.1.5.1, B.1.5.2, B.1.5.3, B.1.6.1, B.1.6.5, B.1.6.7, B.4.1, B.4.2, B.4.2.1, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.5.1.1, B.5.4, B.5.6, B.5.7.

rg_D: B.4.2, B.4.2.1, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5.

$\overleftarrow{rg_D}$: B.4.3, B.4.4, B.4.4.1, B.4.5, B.4.5.1, B.4.5.2, B.4.6, B.4.6.1, B.4.6.2, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.6, B.4.6.7, B.4.6.8, B.4.6.11.

rg_D: B.4.3, B.4.3.1, B.4.4.2, B.4.4.3, B.4.4.4, B.4.5.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.8, B.4.6.9, B.4.6.10.

$\overleftarrow{rg}$: B.1.3.3, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.4, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.5, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.1, B.2.1.1, B.2.2, B.2.2.1, B.2.3, B.2.3.1, B.2.3.2, B.2.4, B.2.4.1, B.2.4.2, B.2.4.4, B.2.4.5, B.2.4.6, B.2.4.7, B.2.4.8, B.2.4.9, B.2.4.10, B.3.1, B.3.1.1, B.3.2, B.3.2.1, B.3.3, B.3.3.1, B.3.3.2, B.3.4, B.3.4.1, B.3.4.2, B.3.4.4, B.3.4.5, B.3.4.6, B.3.4.7, B.3.4.8, B.3.4.10, B.3.4.11, B.3.4.12, B.4.4, B.4.4.1, B.4.4.2, B.4.5, B.4.5.1, B.4.5.2, B.4.6, B.4.6.1, B.4.6.3, B.4.6.4, B.4.6.5, B.4.6.8, B.5.2.1, B.5.3.1, B.5.4.1, B.5.4.2, B.5.4.3, B.5.5.1, B.5.5.2, B.5.5.3, B.5.5.4, B.5.6.1, B.5.6.2, B.5.7.1.

rg: B.1.3.3, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.4, B.1.4.5, B.1.4.6, B.1.4.7, B.1.5.5, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.12, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.2.1.1, B.2.1.2, B.2.2.1, B.2.2.2, B.2.3.1, B.2.3.2, B.2.4, B.2.4.2, B.2.4.3, B.3.1.1, B.3.1.2, B.3.2.1, B.3.2.2, B.3.3.1, B.3.3.2, B.3.4.2, B.3.4.3, B.4.3, B.4.3.1, B.4.4.2, B.4.4.4, B.4.5.1, B.4.6.3, B.4.6.5, B.5.2.1, B.5.3.1, B.5.3.3, B.5.4.1, B.5.4.2, B.5.4.3, B.5.5.1, B.5.5.2, B.5.5.3, B.5.5.4, B.5.6.1, B.5.6.2, B.5.7.1, B.5.8.

rhs: B.2.1.3, B.2.2.3, B.2.4.4, B.3.1.3, B.3.2.3, B.3.4.4.

rhs_{Do}: B.4.4.2, B.4.4.3, B.4.4.4.

Rid: B.1.2, B.1.3.1, B.1.3.4, B.1.3.5, B.1.3.6, B.1.4.1, B.1.4.2, B.1.4.5, B.1.4.6, B.1.4.7, B.1.4.9, B.1.5.1, B.1.5.2, B.1.5.3, B.1.5.6, B.1.5.7, B.1.5.8, B.1.6.1, B.1.6.4, B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.8, B.1.6.13, B.1.6.14, B.1.6.15, B.1.6.16, B.1.6.17, B.1.7, B.2.1.2, B.2.1.3, B.2.1.4, B.2.1.5, B.2.1.6, B.2.2, B.2.2.2, B.2.2.6, B.2.3, B.2.4, B.2.4.3, B.2.4.7, B.3.1.2, B.3.1.3, B.3.1.4, B.3.1.5, B.3.1.6, B.3.2, B.3.2.2, B.3.2.6, B.3.3, B.3.4, B.3.4.3, B.3.4.7, B.4.1.1, B.4.2.5, B.4.3.1, B.4.4, B.4.5, B.4.6, B.4.6.7, B.4.6.8, B.5.1.1, B.5.3.1, B.5.4, B.5.4.1, B.5.5.1, B.5.5.4, B.5.6, B.5.6.1, B.5.7.1, B.5.8.

sel_2: B.1.3.1, B.1.3.4, B.1.3.5, B.1.4.1, B.1.4.5, B.1.4.6, B.1.5.1, B.1.5.6, B.1.5.7, B.1.6.1, B.1.6.5, B.1.6.7, B.1.6.8, B.1.6.13, B.1.6.14, B.1.6.16, B.2.2, B.2.4.1, B.3.2, B.3.4, B.4.4, B.4.6, B.5.1.1, B.5.2, B.5.3, B.5.3.1, B.5.4.1, B.5.6, B.5.6.1, B.5.7, B.7.2.4, B.8.4.3, B.8.4.9, B.8.4.11, B.8.4.14, B.8.5.5, B.8.5.19, *B.9.3.1*, B.9.3.2, B.9.6.14.

$ROOT$: B.8.5.16, B.8.5.22, B.8.5.23.

$rsubst$: B.2.1.2, B.2.2.2, B.2.3.2, B.2.4.3, B.3.1.2, B.3.2.2, B.3.3.2, B.3.4.3, B.5.2.1, B.5.3.1, B.5.4.2, B.5.4.3, B.5.5.2, B.5.5.3, B.5.5.4, B.5.6.1, B.5.7.1, *B.9.1.5*.

$r1$: B.2.2.6, B.2.2.7, B.3.2.6, B.3.2.7, B.3.2.8, B.4.1.1, B.4.1.2, B.5.5.1, B.5.5.4.

$r2$: B.4.1.1.

$r3$: B.4.1.1, B.4.1.2.

s_1: B.9.1.5, B.9.3.1, B.9.3.2, B.9.4.3, B.9.4.4, B.9.6.10, B.9.6.14, B.9.8.1, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.10, B.9.8.11, B.9.8.12, B.9.8.13.

s_2: B.9.1.5, B.9.3.1, B.9.3.2, B.9.4.3, B.9.4.4, B.9.6.10, B.9.6.14, B.9.8.1, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.10, B.9.8.11, B.9.8.12, B.9.8.13.

sam: B.1.5.2, B.1.5.3, _B.9.8.12_.
satisfiability: B.2.1, B.2.2, B.2.3, B.2.4, B.3.1, B.3.2, B.3.3, B.3.4, B.8.1.2.
seq: B.4.6.6, B.7.1, B.7.2.2, B.7.2.6, B.7.2.8, B.7.2.10, B.7.2.13, B.7.2.14, B.7.2.17, B.7.2.18, B.8.5.21, B.8.5.26, B.8.5.28, B.8.5.29, B.8.5.30, B.8.5.31, B.8.5.33, _B.9.7.1_, B.9.7.2, B.9.7.4, B.9.7.5, B.9.7.6, B.9.7.7, B.9.7.8, B.9.7.9, B.9.7.10, B.9.7.11, B.9.7.12, B.9.7.13, B.9.7.14, B.9.7.15, B.9.8.10, B.9.8.12, B.9.8.13, B.9.9.1, B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7.
SEQ_{inv}: B.8.5.26, B.8.5.32, B.8.5.33.
SEQ_{mod}: B.8.5.33, B.8.5.34.
SEQ_{op}: B.8.5.27, B.8.5.32, B.8.5.33.
SEQ_{st}: B.8.5.26.
seq_as_map: _B.9.8.12_.
SEQ_CREATE_{val}: B.8.5.32.
seq_induction: _B.9.7.2_.
SEQ_INSERT_{val}: B.8.5.32.
SEQ_LENGTH_{val}: B.8.5.32.
seq_pair_as_map: _B.9.8.12_.
SEQ_READ_{val}: B.8.5.32.
SEQ_val: B.8.5.34.
seqfilter: _B.9.7.8_.
seqmap: _B.9.7.7_.
Sequences: B.9.7, B.10.
SEQUENCES_int: B.8.5.25.
set: B.2.1.5, B.3.1.5, B.9.6.1, B.9.6.2, B.9.6.4, B.9.6.5, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.9, B.9.6.10, B.9.6.11, B.9.6.12, B.9.6.13, B.9.6.14, B.9.7.10, B.9.8.2, B.9.8.6, B.9.8.11, B.9.8.13, B.9.9.4.
SET: B.1.6.2, B.1.6.9, B.1.6.10, B.1.6.14, B.1.7, B.1.7.1, B.1.7.2, B.5.4, B.5.4.1, B.6.2.
set_induction: B.9.6.2.
setmap: B.9.6.10.
side: B.2.2.6, B.3.4.9, B.3.4.11.
$side_A$: B.3.2.6, B.3.2.7.
$side_B$: B.3.2.6.
simp_andR: B.3.2.6, B.3.4.9, B.3.4.11, B.9.2.7.
sing: B.1.4.12, B.1.6.10, B.1.7.2, B.2.2.5, B.3.2.5, B.8.2.3, B.8.2.4, B.8.3.2, B.8.5.13, B.9.6.14.
single: B.2.2.3, B.2.2.4, B.2.2.5, B.2.2.7, B.3.2.3, B.3.2.4, B.3.2.5, B.3.2.8, B.4.4.3, B.4.4.4, B.4.6.6, B.7.2.17, B.9.6.14, B.9.7.12, B.9.8.13, B.9.9.8.
SIZE: B.8.5.3, B.8.5.11, B.8.5.12.
snd: B.5.6.1, B.9.6.14.
sort: B.1.2, B.1.4.2, B.1.6.1, B.7.1, B.7.2.2, B.7.2.4, B.7.2.5, B.7.2.6, B.7.2.7, B.7.2.8, B.7.2.10, B.7.2.12, B.7.2.13, B.7.2.14, B.7.2.15, B.7.2.16, B.7.2.17, B.7.2.18, B.8.1.1, B.8.1.2, B.8.2.1, B.8.2.2, B.8.2.3, B.8.2.4, B.8.3.1, B.8.3.2, B.8.3.3, B.8.3.4, B.8.4.1, B.8.4.3, B.8.4.4, B.8.4.6, B.8.4.8, B.8.4.9, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.15, B.8.4.16, B.8.5.2, B.8.5.3, B.8.5.11, B.8.5.12, B.8.5.13, B.8.5.15, B.8.5.16, B.8.5.22, B.8.5.23, B.8.5.24, B.8.5.26, B.8.5.27, B.8.5.32, B.8.5.33, B.8.5.34, B.9.1, B.9.1.1, B.9.1.3, B.9.1.4, B.9.1.5, B.9.2.11, B.9.3.1, B.9.3.2, B.9.4.1, B.9.4.2, B.9.4.3, B.9.4.4, B.9.5, B.9.6.1, B.9.6.2, B.9.6.4, B.9.6.5, B.9.6.6, B.9.6.7, B.9.6.8, B.9.6.9, B.9.6.10, B.9.6.11, B.9.6.12, B.9.6.13, B.9.6.14, B.9.7.1, B.9.7.2, B.9.7.4, B.9.7.5, B.9.7.6, B.9.7.7, B.9.7.8, B.9.7.9, B.9.7.10, B.9.7.11, B.9.7.12, B.9.7.13, B.9.7.15, B.9.8.1, B.9.8.2, B.9.8.3, B.9.8.6, B.9.8.7, B.9.8.8, B.9.8.9, B.9.8.10, B.9.8.11, B.9.8.12, B.9.8.13, B.9.9.1, B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7, B.9.9.8, _B.10_.
spam: _B.9.8.12_.
sq_i: B.8.5.28, B.8.5.29, B.8.5.30, B.8.5.31.
sq_o: B.8.5.28, B.8.5.29, B.8.5.30, B.8.5.31.
st: B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.8, B.8.4.6, B.8.4.9, B.8.4.11, B.8.4.14, B.8.4.15, B.8.5.2, B.8.5.26.
st_{ai}: B.8.3.1.
st_{ao}: B.8.3.1.
st_{ci}: B.8.3.1.
st_{co}: B.8.3.1.
st_i: B.8.1.2, B.8.4.3, B.8.4.4, B.8.4.8, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.16.
st_o: B.8.1.2, B.8.4.3, B.8.4.4, B.8.4.8, B.8.4.11, B.8.4.13, B.8.4.14, B.8.4.16.
state: B.8.1.1, B.8.1.2, B.8.2.1, B.8.2.2, B.8.2.3, B.8.2.4, B.8.4.1, B.8.4.4,

B.8.4.6, B.8.4.8, B.8.4.9, B.8.4.11, B.8.4.12, B.8.4.13, B.8.4.14, B.8.4.16.
$state_a$: B.8.3.1, B.8.3.2, B.8.3.3, B.8.3.4.
$state_c$: B.8.3.1, B.8.3.2, B.8.3.3, B.8.3.4.
$state_1$: B.8.4.3, B.8.4.14.
$state_2$: B.8.4.3, B.8.4.14.
sub: <u>*B.9.5.5*</u>.
subgoal: B.2.2.6, B.2.4.7, B.2.4.9, B.2.4.10, B.3.2.6, B.3.4.7, B.3.4.9, B.3.4.11.
subseq: <u>*B.9.7.12*</u>, B.9.7.13.
subset: B.5.6.1, B.9.6.9, B.9.8.13.
subset_empty: B.5.2.1, B.5.3.1, B.9.6.14.
subset_prop: B.4.1.2, B.4.6.11, B.5.4.3, B.5.5.3, B.5.6.1, B.9.6.14.
subst: B.2.1.6, B.2.2.7, B.2.4.1, B.2.4.8, B.3.1.6, B.3.2.8, B.3.4.1, B.3.4.8, B.4.2.5, B.4.5, B.4.6.1, B.4.6.7, <u>*B.9.1.1*</u>.
subst_opeq: B.5.8, B.8.4.15.
succ: B.2.4.6, B.3.4.6, B.7.2.8, <u>*B.9.5.1*</u>, <u>*B.9.5.2*</u>, B.9.5.5, B.9.5.6, B.9.5.7, B.9.6.8, B.9.7.6, B.9.7.11, B.9.7.14.
succ_new: <u>*B.9.5.1*</u>.
succ_one_to_one: <u>*B.9.5.1*</u>.
sym: B.4.4.1, B.4.5.1, <u>*B.9.1.3*</u>.
sym_imp: <u>*B.9.2.6*</u>.
sym_neq: <u>*B.9.2.11*</u>.
sym_not_equiv: <u>*B.9.2.10*</u>.
sym_opeq: B.8.4.15.
T_{inv}: B.1.7.
T_{mod}: B.1.7, B.1.7.2.
T_{op}: B.1.7, B.1.7.1, B.1.7.2, B.6, B.6.1, B.6.2, B.6.3, B.6.4, B.6.5.
T_{st}: B.1.7.
$T\_CHECKIN_{val}$: B.1.7.2.
$T\_CHECKOUT_{val}$: B.1.7.2.
T_FREE_{val}: B.1.7.2.
T_OPEN_{val}: B.1.7.2.
T_RESET_{val}: B.1.7.2.
T_SET_{val}: B.1.7.2.
T_valid: B.1.7.2.
tail: B.9.7.5.
TEST: B.8.5.3, B.8.5.11, B.8.5.12, B.8.5.16, B.8.5.22, B.8.5.23.
tnd: <u>*B.9.2.4*</u>.
tr: B.9.9.8.
tr_i: B.8.5.19, B.8.5.20, B.8.5.21.
tr_o: B.8.5.17, B.8.5.18, B.8.5.19, B.8.5.20, B.8.5.21.
trans: B.4.4.1, B.5.5.3, B.5.6.1, <u>*B.9.1.3*</u>, B.9.6.14.
trans_imp: <u>*B.9.2.6*</u>.
trans_opeq: B.8.4.15.
trans_product: <u>*B.9.1.4*</u>.
tree: B.1.3.1, B.1.4.1, B.1.4.2, B.1.5.1, B.1.5.3, B.2.1.4, B.3.1.4, B.4.1.1, B.8.5.15, B.8.5.17, B.8.5.18, B.8.5.19, B.8.5.20, B.8.5.21, B.8.5.23, <u>*B.9.9.1*</u>, B.9.9.2, B.9.9.4, B.9.9.5, B.9.9.6, B.9.9.7, B.9.9.8.
$TREE_{inv}$: B.8.5.15, B.8.5.22, B.8.5.23.
$TREE_{mod}$: B.8.5.23, B.8.5.24.
$TREE_{op}$: B.8.5.16, B.8.5.22, B.8.5.23.
$TREE_{st}$: B.8.5.15.
$TREE\_CREATE_{val}$: B.8.5.22.
tree_induct: <u>*B.9.9.2*</u>.
$TREE\_INSERT_{val}$: B.8.5.22.
$TREE\_PATH_{val}$: B.8.5.22.
$TREE\_ROOT_{val}$: B.8.5.22.
$TREE\_TEST_{val}$: B.8.5.22.
TREE_val: B.8.5.24.
Trees: B.9.9, B.10.
TREES_int: B.8.5.14.
true: B.1.3.3, B.1.4.4, B.1.5.5, B.1.6.12, B.1.6.13, B.2.1, B.2.1.4, B.2.1.5, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1, B.3.1.4, B.3.1.5, B.3.2.4, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.11, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.4.3, B.5.8, B.7.2.8, B.8.5.4, B.8.5.6, B.8.5.7, B.8.5.10, B.8.5.17, B.8.5.18, B.8.5.20, B.8.5.26, B.8.5.28, B.8.5.29, B.8.5.31, <u>*B.9.2.1*</u>, B.9.2.4, B.9.2.6, B.9.2.7, B.9.2.9, B.9.2.10, B.9.7.8, B.9.8.9, B.9.9.5, B.9.9.8.
true_in: <u>*B.9.2.4*</u>.
true_is_true: B.4.3, <u>*B.9.2.10*</u>.
uid: B.1.6.16, B.5.6, B.5.6.1.
Uid: B.1.6.1, B.1.6.5, B.1.6.7, B.1.6.14, B.1.6.15, B.1.6.16, B.1.7, B.5.1.1, B.5.4, B.5.4.1, B.5.5.1, B.5.5.4, B.5.6, B.5.6.1, B.5.8.
unfold: B.2.1.3, B.2.1.4, B.2.1.5, B.2.2.3, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.4, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.3, B.3.1.4, B.3.1.5, B.3.2.3, B.3.2.4, B.3.2.5, B.3.2.6, B.3.2.7, B.3.4.4, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.10, B.3.4.11, B.3.4.12, B.4.1, B.4.1.1, B.4.2.1, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.4.3.1,

B.4.4.1, B.4.4.3, B.4.4.4, B.4.5.1, B.4.5.2, B.4.6.1, B.4.6.5, B.4.6.6, B.4.6.8, B.4.6.9, B.4.6.10, B.5.3.3, B.5.4.1, B.5.5.1, B.5.6.2, B.5.7.1, B.9.1.5.
unfold_left: B.9.1.5.
union: B.9.6.5, B.9.6.14.
unit: B.3.2.6, B.7.2.13, B.7.2.18.
univ: B.2.1.6, B.2.2.6, B.2.4.7, B.2.4.10, B.3.1.6, B.3.2.6, B.3.4.7, B.3.4.11, B.4.2.5, B.9.4.2.
univ_imp: B.9.4.4.
up: B.1.4.12, B.1.6.10, B.1.7.2, B.2.1.5, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.5, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.11, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.8.2.4, B.8.5.13.
update: B.2.4.1, B.2.4.2, B.2.4.3, B.2.4.4, B.2.4.5, B.2.4.6, B.2.4.7, B.2.4.8, B.2.4.9, B.2.4.10.
$val\_assembly_4$: B.1.3.8, B.1.4.9, B.1.5.10, B.1.6.10, B.1.7.2, B.8.2.4.
val_complpre: B.6.1, B.6.2, B.6.3, B.6.4, B.6.5, B.8.4.14.
val_init_in: B.5.3, B.8.4.14.
val_oconj: B.5.2, B.5.3, B.8.4.14.
val_odisj: B.6.1, B.6.2, B.6.3, B.6.4, B.6.5, B.8.4.14.
val_op: B.1.3.8, B.1.4.9, B.1.5.10, B.1.6.10, B.1.7.2, B.2.1, B.2.2, B.2.3, B.2.4, B.3.1, B.3.2, B.3.3, B.3.4, B.5.2, B.5.3, B.5.4, B.5.5, B.5.6, B.5.7, B.6, B.6.1, B.6.2, B.6.3, B.6.4, B.6.5, B.8.1.2, B.8.2.3, B.8.2.4, B.8.4.14, B.8.5.11, B.8.5.22, B.8.5.32.
val_oplist: B.1.7.2, B.8.2.3.
val_retr: B.1.4.10, B.4.2, B.8.3.3, B.8.3.4.
val_retr_D: B.1.4.10.
val_strpre: B.5.4, B.5.6, B.8.4.14.
val_subst_inv: B.5.2, B.5.3, B.5.4, B.5.5, B.8.4.14.
val_xtnd_in: B.5.6, B.8.4.14.
val_xtnd_inv: B.5.2, B.5.3, B.5.4, B.5.5, B.5.6, B.5.7, B.8.4.14.
val_xtnd_st: B.5.4, B.5.5, B.5.6, B.5.7, B.8.4.14.
valid: B.2.1.4, B.2.1.5, B.2.2.4, B.2.2.5, B.2.2.6, B.2.4.5, B.2.4.6, B.2.4.9, B.2.4.10, B.3.1.4, B.3.1.5, B.3.2.4, B.3.2.5, B.3.2.6, B.3.4.5, B.3.4.6, B.3.4.9, B.3.4.11, B.4.2.2, B.4.2.3, B.4.2.4, B.4.2.5, B.9.2.9.
valid_MAP: B.8.5.13.
weaken: B.4.1.2, B.4.6.11, B.9.6.14.
wff: B.3.2.6, B.7.2.8, B.7.2.12.
wff_lemma: B.1.4.9, B.3.4.1, B.4.2.5.
w4: B.8.3.5.
x_1: B.9.8.3.
x_2: B.9.8.3.
xtnd: B.1.6.5, B.1.6.6, B.1.6.7, B.1.6.8, B.8.4.9, B.8.4.11, B.8.4.14.
xx: B.9.6.12, B.9.6.13.
yes: B.9.6.11.