Journal of Automated Reasoning (2020) 64:1093–1121
https://doi.org/10.1007/s10817-019-09537-9

Exploring the Structure of an Algebra Text with Locales

Clemens Ballarin[1]

Received: 12 April 2019 / Accepted: 11 November 2019 / Published online: 30 November 2019

Abstract
Locales, the module system of the theorem prover Isabelle, were designed so that developments in abstract algebra could be represented faithfully and concisely. Whether these goals were met is assessed through a case study. Parts of an algebra textbook, Jacobson's *Basic Algebra*, that are challenging structurally were formalised. Key parts of the formalisation are presented in greater detail. An analysis of the work from both qualitative and quantitative perspectives substantiates that the design goals were met. In particular, the size ratio of formal to "pen and paper" text does not increase when going further into the book. The analysis also yields guidance on locales including patterns of use, which are identified and described.

Keywords Abstract algebra · Case study · Isabelle · Locales · Module system · Theorem prover

1 Introduction

Locales are the module system of the theorem prover Isabelle. Since their conception in the late nineties [18] locales have seen a considerable evolution from the initial design. Today, locales are widely used. A count in January 2019 yielded that 219 of the 455 entries of the Archive of Formal Proofs (www.isa-afp.org, development version) use locales.[1] The proportion of entries declaring locales is roughly 50% across the domains of Computer Science, Logic and Mathematics, which are distinguished in the archive. Only for Tools the proportion is significantly lower with only three of the 13 entries containing locale declarations.

The services provided by locales go beyond those of module systems known from programming languages. They are integrated with Isabelle's proof language Isar and are designed to provide adequate support for reasoning about algebraic structures. The objective of the study presented here is to gain confidence that this is the case. A fragment of a mathematics textbook is translated into a formal, machine-checkable document in a faithful manner. That is, definitions in the formal document should reflect those of the mathematical text and likewise the formal proofs follow their informal counterparts closely. The challenge sought

[1] Counted were the entries containing locale declarations—more precisely, where at least one theory file contained a line matching the regular expression `^locale␣`.

✉ Clemens Ballarin
ballarin@in.tum.de

[1] Danziger Str. 6a, 76199 Karlsruhe, Germany

is finding an adequate module structure for the "pen-and-paper" text reflecting the concepts found there—with the purpose of understanding the capabilities of locales. For the module structure to be considered adequate, objects of discourse, algebraic structures and their elements, should be similarly concise and no significant additional proof work should be required in the formal text. On the other hand, we do not worry about the precise notation of the objects of discourse and also take into consideration that proof scripts tend to be longer than informal proofs.

What would be a suitable text for such a study? It should, of course, be from the domain of abstract algebra. For practical reasons it should be self-contained and of moderate size. This suggests fairly elementary material from an algebra textbook. The first volume of Jacobson's *Basic Algebra* [16] was chosen, mostly due to the author's familiarity with the book. The text is demanding from the outset (especially for a computer scientist) and is probably not the preferred choice of text for a regular formalisation effort, where a low-level style is of advantage.

The formalised fragment stretches over a bit more than the first one hundred pages of the book. As a continuous body it would amount to 11.4 pages. It covers monoids, groups and rings and arrives at the fundamental theorems of homomorphisms of each of these structures. The choice enables studying how constructions for algebraic structures build on top of each other. For the purpose of this study, it does not matter that similar material has been formalised before.

The report starts with an introduction to locales. A walk through the formalised material follows. Space constraints do not permit going through the entire body. Key parts are presented, and the focus is on the role of locales. In the subsequent review the formalisation is assessed based on the goals set out above. Changes to the locale implementation that were triggered by the work are briefly explained, and usage patterns of locales identified in the work are presented. A discussion of what can be learnt from the case study concludes the analysis.

The discussed formal proof documents are available in the Archive of Formal Proofs [7].

2 Locales

Locales are an extension of Isabelle's Isar proof language [25] by means of manipulating "knowledge containers" or modules, which were designed for representing algebraic structures. The central concept is the *locale*, a theory functor that maps parameter operations (or simply parameters) and a specification to defined operations (involving the parameters) and theorems (implied by the specification). A sketch of locales follows. My detailed account on the semantics of locales is the exhaustive reference [6]. There, locales are also related to ML-style module systems [15] and other means of reuse in both provers and programming languages.

2.1 Declaring Locales

In its simplest form, a locale declaration consists of a **fixes** and an **assumes** clause:

$$\textbf{locale}\ n = \ \textbf{fixes}\ \overline{y}\ +\ \textbf{assumes}\ a_1 : A_1 \ldots a_j : A_j$$

The $\overline{y}$ are the parameters and $A_1 \wedge \cdots \wedge A_j$ is the specification of the locale. The A_i are versions of the user input where free variables except parameters are universally closed. They are also called axioms or assumptions of the locale. For example, a locale for monoids may be declared like this:[2]

[2] Readers wishing to reproduce the examples in Isabelle should use bold, not regular, "1" (input token `\<one>`).

```
locale monoid =
  fixes M and composition (infixl "·" 70) and unit ("1")
  assumes composition_closed: "⟦ a ∈ M; b ∈ M ⟧ ⟹ a · b ∈ M"
 and unit_closed: "1 ∈ M"
 and associative: "⟦ a ∈ M; b ∈ M; c ∈ M ⟧ ⟹ (a · b) · c = a · (b · c)"
 and left_unit: "a ∈ M ⟹ 1 · a = a"
 and right_unit: "a ∈ M ⟹ a · 1 = a"
```

Types, in particular types of the parameters, may be left implicit. They are inferred automatically. The theory of a locale n is elaborated in *context blocks*:

$$\textbf{context}\ n\ \textbf{begin}\ \ldots\ \textbf{end}$$

The locale n is the *target* of the block. These commands are available in context blocks:

$$\begin{aligned}&\textbf{definition}\ c\ \ \textbf{where}\ c \equiv t\\ &\textbf{notation}\ c\\ &\textbf{theorem}\ b : B\end{aligned}$$

The first command, **definition**, declares a new defined operation c with defining equation $c \equiv t$ and optional concrete syntax; **notation** enables changing the concrete syntax of an existing operation; **theorem** introduces a theorem B named b. Declarations in a context block are persistent—that is, they are present in subsequent context blocks of the same target. Parameters and specification of a locale are the *header*, the collections of declarations made in its context blocks form the *body*.

Locales are hierarchic and a graph of interdependent locales is maintained by the system. Dependencies may be given when declaring a locale through a *locale expression* at the beginning of the declaration,

$$\textbf{locale}\ n = \mathit{expression} + \cdots,$$

or between existing locales through a sublocale declaration:

$$\textbf{sublocale}\ n \subseteq \mathit{expression}\ \langle \mathit{proof} \rangle$$

Dependencies declared via the **sublocale** command are derived. The system creates a proof obligation, which must be discharged by supplying a proof. A locale expression consists of a sequence of *locale instances* followed by an optional **for** clause:

$$I_1 + \cdots + I_k\ \ \textbf{for}\ \overline{x}$$

A locale instance is either positional:

$$q : n\,\overline{t}\ \ \textbf{rewrites}\ \overline{eq}$$

or by name:

$$q : n\ \ \textbf{where}\ \overline{y} = \overline{t}\ \ \textbf{rewrites}\ \overline{eq}$$

Here, q is an optional qualifier, n the instantiated locale, $\overline{y}$ are its parameters, $\overline{t}$ terms from the target and $\overline{eq}$ optional equations. Terms are substituted for the parameters of the instantiated locale—in the positional case, in declaration order of the parameters. The qualifier q, if present, is prepended to names of derived operations and of theorems. Qualifiers are used to distinguish multiple instances of the same locale. The purpose of the equations is explained below.

In a locale declaration, the **for** clause defines the parameters $\overline{x}$ of the locale expression and the context of the locale instances.[3] The expression denotes locale instances that are *imported* to the declared locale. That is, the specifications of the imported instances contribute to the specification of the declared locale, and their declarations (definitions and theorems) are available in its body. The following declaration involves an expression consisting of a single instance and a for clause:

```
locale submonoid = monoid M "(·)" 1
  for N and M and composition (infixl "·" 70) and unit ("1") +
  assumes subset: "N ⊆ M"
    and sub_composition_closed: "⟦ a ∈ N; b ∈ N ⟧ ⟹ a · b ∈ N"
    and sub_unit_closed: "1 ∈ N"
```

The notation `(·)` refers to the `composition` operation. As a notational convenience uninstantiated parameters in locale instances are implicitly added to the for clause. This is useful when constructing linear locale hierarchies—for example, the above declaration may be abbreviated to

```
locale submonoid = monoid +
  fixes N
  assumes subset: "N ⊆ M"
    and sub_composition_closed: "⟦ a ∈ N; b ∈ N ⟧ ⟹ a · b ∈ N"
    and sub_unit_closed: "1 ∈ N"
```

albeit yielding the different parameter order `M composition unit N`.

The sublocale command lets one change existing locale hierarchies: for a locale n, **sublocale** $n \subseteq$ *expression* modifies the graph of locales maintained by the system as if the declaration of the target n imported *expression*. This makes the bodies of the instantiated locales available in n (the *target* of the operation). For example,

```
sublocale submonoid ⊆ sub: monoid N "(·)" 1   ⟨proof⟩
```

asserts that the carrier set `N` of the submonoid and the monoid operations indeed form a monoid. The notational variant **sublocale** *expression* ⟨*proof*⟩ is available in context blocks, and the command applies to the target of the block. The specification of the target is required to imply the specifications of the instantiated locales. Proof obligations are generated and must be discharged.

The sublocale command is not restricted to instantiating locale parameters; rewrites clauses in locale instances enable specifying rewrite rules for replacing defined operations of a locale by terms from the target. In combination with instantiation they provide signature morphisms [11] on locales. The rewrite rules yield additional proof obligations.

[3] In locale expressions outside locale declarations, the for clause retains its usual semantics as a binder.

2.2 Reasoning with Locales

Locales are extra-logical. The functors they represent are not encoded in Isabelle's logic. Instead, whenever a locale is visited—that is, a context block entered—the locale is *activated*:

- The locale graph is traversed and all reachable locale instances are activated recursively.
- The body of the locale is added to the context, making its declarations available.

The precise working of activation is described in my detailed account [6].

Activation of a locale initialises a context, so it can be worked in. It is also possible to activate additional locale instances in the current context, which can be a theory, a context block or a proof. We will only encounter the command applicable in context blocks:

interpretation *expression* ⟨*proof*⟩

It enriches the current block by the information on the instances identified by *expression* and stored in the respective locales. The operation is also known as *theory interpretation* [10]. The difference to the sublocale command is that its effect is temporary and limited to that block. The command does not change the locale hierarchy permanently.

We also need to briefly look at how the proof obligations created by **sublocale** and **interpretation** are represented. Each locale n is accompanied by a *locale predicate* n, which reflects the specification of n in the logic. The locale instance $n\,\bar{t}$ is represented by the proposition $n\,\bar{t}$. The `unfold_locales` method (which is integrated with Isabelle's default method, so it rarely occurs explicitly in proof scripts) provides backward reasoning on locale predicates and is used for refining proof obligations. It reduces atomic propositions on locale predicates to locale axioms, on which then further reasoning can take place. For a locale that imports a hierarchy of locales, this includes corresponding instances of axioms of the imported locales. The method is aware of the locale instances that are active in the current context and discharges matching subgoals during the reduction process automatically. The latter enables a technique called *bootstrapping*, which will be discussed in the course of the paper.

3 The Formalisation

The formalised material is from the first three chapters of Jacobson's *Basic Algebra I* [16] and covers all that the author undertakes to arrive at the fundamental theorem of homomorphisms of rings, also known as the universal property of ring homomorphisms. Cosets are introduced as orbits of translations, and so we can afford a short detour to Cayley's theorems for monoids and groups.

3.1 Concepts from Set Theory

In Isabelle/HOL `S` $\rightarrow_E$ `T` denotes the set of functions with domain `S` and co-domain `T`. Extensionality is achieved by mapping values outside `S` to an arbitrary but fixed value about which nothing can be proved [12]. Elements of `S` $\rightarrow_E$ `T` are obtained with bounded abstraction `λs∈S. t`. Composition `compose S` β α is defined as `λs∈S.` β `(`α `s)` and α ‘ `A` is the image of `A` under α. Since an element of `S` $\rightarrow_E$ `T` does not encode its domain, occasionally it needs to be made explict, such as in `compose S` β α. Rather than developing maps from scratch the existing concepts are used. The declaration

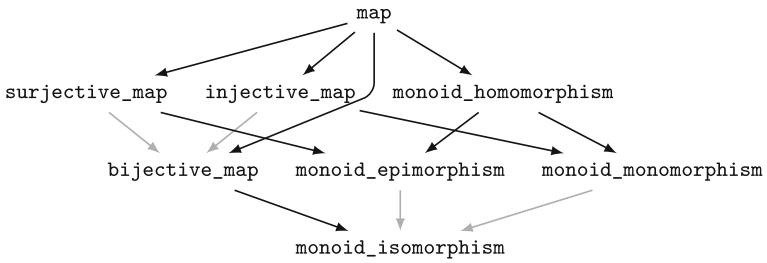


Fig. 1 Locale hierarchy of maps and monoid homomorphisms. $x \rightarrow y$ means that y imports x, $x \dashrightarrow y$ that the relationship is established through a sublocale declaration. Bijective maps are directly based on maps rather than on surjective and injective maps. This is to make better use of knowledge on bijective functions already available in Isabelle

```
locale map = fixes α and S and T assumes "α ∈ S →E T"
```

enables working with maps at the level of locales. Locales for injective, surjective and bijective maps are declared as well. Figure 1 shows the hierarchy of these locales, along with those of monoid homomorphisms, which are introduced later.

Jacobson's treatment of equivalence classes [16, p. 11] is interesting enough to warrant its formalisation. He simultaneously develops the concepts of equivalence relation E on and partition P of a set S and then shows that they are equivalent. Both are straightforwardly translated to locales, `equivalence` and `partition`, respectively. In the context of `equivalence` the natural map is defined as

```
Class = (λa∈S. {b ∈ S. (b, a) ∈ E})
```

and the quotient set S/E is the associated partition into classes:

```
Partition = {Class a |a. a ∈ S}
```

That `equivalence` and `partition` are the same thing is asserted through suitable mutual sublocale declarations.

The next topic is factoring a map through an equivalence relation. Every map α gives rise to an equivalence relation E_α in S where $a E_\alpha b$ if and only if $\alpha(a) = \alpha(b)$. It is convenient to analyse this situation in a dedicated context:

```
locale fiber_relation = map
```

Both locales have the same parameters and specification but their bodies differ; `fiber_relation` is a *clone* of `map`. While `fiber_relation` inherits the declarations of `map` the latter does not inherit the declarations of the former. The context is declared a sublocale of an instance of `equivalence` with E_α as the relation.

The map α gives rise to the induced map $\bar{\alpha}$ on the quotient set such that $\bar{\alpha}(\bar{a}) = \alpha(a)$. With the above sublocale declaration the definition of the induced map is

```
induced = (λA∈Partition. THE b. ∃a∈A. b = α a)
```

where `Partition` now represents the quotient set S/E_α and `THE` is the definite selection operator. Reasoning about functions defined on equivalence classes can be technically challenging [21]. The theorem

$$\llbracket \texttt{A} \in \texttt{Partition};\ \bigwedge \texttt{a}.\ \texttt{a} \in \texttt{S} \Longrightarrow \texttt{P (Class a)} \rrbracket \Longrightarrow \texttt{P A}$$

leads to concise proof scripts. Used as an elimination rule, a goal about an arbitrary element `A` of the quotient set is reduced to a goal about an element `a` of the underlying set. Theorems on the factorisation of α into natural and induced maps follow.

$$\texttt{compose S induced Class} = \alpha$$
$$\llbracket \beta \in \texttt{Partition} \to_E \texttt{T};\ \texttt{compose S } \beta \texttt{ Class} = \alpha \rrbracket \Longrightarrow \beta = \texttt{induced}$$

Sublocale declarations assert surjectivity and injectivity of the involved maps:

sublocale `equivalence ⊆ natural: surjective_map Class S Partition` ⟨*proof*⟩
sublocale `fiber_relation ⊆ induced: injective_map induced Partition T` ⟨*proof*⟩

This completes the statement of the universal property of the map α.

3.2 Abstract Monoids and Groups

Jacobson's definitions of monoids and groups and the associated substructures can be expressed with locales in a straightforward manner. For the corresponding monoid locales, see the examples in Sect. 2.2. The notions of invertibility and inverse are defined in the context of monoids:

```
context monoid begin
  definition "u ∈ M ⟹ invertible u ⟷ (∃v ∈ M. u · v = 1 ∧ v · u = 1)"
  definition "inverse = (λu ∈ M. THE v. v ∈ M ∧ u · v = 1 ∧ v · u = 1)"
end
```

Then, following Jacobson, "a group G (or $(G, p, 1)$) is a monoid all of whose elements are invertible" [16, Def. 1.2]:

```
locale group =
  monoid G "(·)" 1 for G and composition (infixl "·" 70) and unit ("1") +
  assumes invertible: "u ∈ G ⟹ invertible u"
```

Further "a submonoid of a monoid M [is] a subgroup if, regarded as a monoid, it is a group" [16, p. 31]:

```
locale subgroup = submonoid where N = G + sub: group G "(·)" 1 for G
```

The inverse operations of the group and the subgroup are different, even though Jacobson does not make this explicit. The lemma

$$\texttt{u} \in \texttt{G} \Longrightarrow \texttt{inverse u} = \texttt{sub.inverse u}$$

helps simplify expressions in subgroups involving both group and subgroup operations.

Jacobson's unusual decision to define the inverse operation rather than make it a parameter of the structure poses no problems to locales. In a sublocale declaration or interpretation the operation can be mapped to an operation from the target context via a rewrites clause.

3.3 Monoids and Groups of Transformations

For monoids and groups of transformations, which are monoids and groups whose elements are maps (or *transformations*) on some set S, the translation into locales is less obvious. For a set S the set $M(S)$ of maps of S into itself forms a monoid. This is the monoid of *all transformations*. Composition is the binary operation and the identity map the unit element. A submonoid of $M(S)$ is called a *monoid of transformations* [16, p. 29].

The formalisation starts with a locale that postulates the set S:[4]

```
locale transformations = fixes S :: "'a set"
```

The type constraint ensures that `S` is a set. The monoid of all transformations is identified in the context:

```
sublocale transformations ⊆
monoid "S →E S" "compose S" "identity S" ⟨proof⟩
```

The locale for a transformation monoid M of S follows:

```
locale transformation_monoid =   transformations S +
 submonoid M "S →E S" "compose S" "identity S" for M and S
```

Similarly, the invertible elements of the monoid of all transformations $M(S)$ form a group, the *symmetric group* Sym S:

```
sublocale transformations ⊆
 symmetric: group "Sym" "compose S" "identity S"  ⟨proof⟩
```

`Sym` is defined in the context of `transformations` and denotes the invertible elements of `S` $\to_E$ `S`. In the formalisation, the parameter `S` of `Sym` is implicit. The locale for a transformation group G of S follows:

```
locale transformation_group = transformations S +
  symmetric: subgroup G Sym "compose S" "identity S" for G and S
```

3.4 Isomorphism. Cayley's Theorem

"Two monoids $(M, p, 1)$ and $(M, p', 1')$ are said to be isomorphic if there exists a bijective map η of M to M' such that $\eta(1) = 1'$, $\eta(xy) = \eta(x)\eta(y)$, $x, y \in M$" [16, Def. 1.3]. The definition straightforwardly translates into a locale declaration:[5]

[4] Jacobson requires S to be non-vacuous, but this was not required in the formalisation.

[5] Isabelle requires escaping the single quote character in syntax declarations; "`''`" yields "`'`".

```
locale monoid_isomorphism =      bijective_map η M M' +
  source: monoid M "(·)" 1 + target: monoid M' "(·')" "1'"
  for η and M and composition (infixl "·" 70) and unit ("1")
    and M' and composition' (infixl "·''" 70) and unit' ("1''") +
  assumes commutes_with_composition:
    "⟦ x ∈ M; y ∈ M ⟧ ⟹ η x ·' η y = η (x · y)"
    and commutes_with_unit: "η 1 = 1'"
```

The monoid locales are distinguished by suitable qualifiers, and adequate notation is provided.

"Any monoid is isomorphic to a monoid of transformations" and "any group is isomorphic to a group of transformations" [16, Cayley's theorem]. In the proof, an isomorphism from the monoid $(M, p, 1)$ (or group $(G, p, 1)$) to a monoid (or group) of transformations of the set M (or G) is defined. The isomorphism takes an element a to the map $a_L : x \to ax$, called the *left translation* of a.

The formalisation takes place in the contexts of dedicated clones of locales of monoids and groups for left translations, in the monoid case,

```
locale left_translations_of_monoid = monoid begin
```

and the subsequent statements will be in this context. First,

```
translation = (λa∈M. λx∈M. a · x)
```

is defined, and `(a)`$_L$ will be used to denote `translation a`. The proof that this function is the required isomorphism is prepared in a sequence of three sublocale declarations, where each enriches the context to simplify the subsequent proofs.

```
sublocale transformation: transformations M  ⟨proof⟩
sublocale transformation: transformation_monoid "translation ` M" M  ⟨proof⟩
```

The line of reasoning follows that of Jacobson. First the context is extended by the monoid of all transformations of `M`. The proof is trivial since `transformations` has no assumptions. It is then shown that `translation ` M"` the image of `M` under `translation`, is a submonoid of the monoid of all transformations and therefore a monoid of transformations. That `translation` is an isomorphism comes next, and the third sublocale declaration asserts that a translation is a map.

```
sublocale map translation M "translation ` M"  ⟨proof⟩
theorem translation_isomorphism:
  "monoid_isomorphism translation M (·) 1
  (translation ` M) (compose M) (identity M)"
  ⟨proof⟩
end
```

This concludes reasoning in `left_translations_of_monoid`. Building up the context in several steps, here through sublocale declarations, I call *bootstrapping*.

In the final step, the result is transferred to `monoid` by means of a temporary interpretation:

```
context monoid begin
interpretation left_translations_of_monoid ⟨proof⟩
theorem cayley_monoid: "∃M' composition' unit'.
  transformation_monoid M' M ∧ (M, (·), 𝟭) ≅M (M', composition', unit')"
  ⟨proof⟩
end
```

With this technique, which I call *loose coupling*, left translations are kept separate from monoids.

The corresponding work for groups takes place in `left_translations_of_group` and builds on the monoids case:

```
locale left_translations_of_group = group begin
sublocale left_translations_of_monoid where M = G ⟨proof⟩
sublocale transformation: transformation_group "translation ` G" G ⟨proof⟩
end
```

In addition to the monoid case it needs to be shown that `translation ` G` is a transformation group. Since `(a)`$_L$ has the inverse `(inverse a)`$_L$ for all `a ∈ G` the `(a)`$_L$ are bijective and `translation ` G` is closed under inverses. No additional work on the isomorphism property is required. Cayley's theorem for groups follows, again by means of a temporary interpretation, in the `group` locale.

3.5 Orbits. Cosets of a Subgroup

For any transformation group G of a set S the relation $\sim_G$ on S, where $x \sim_G y$ if $y = \alpha(x)$ for some $\alpha \in G$, is an equivalence relation. The equivalence classes are called the G-orbits and constitute a partition of S [16, p. 51]. The correct context for the formalisation is `transformation_group`:

```
context transformation_group begin
  definition Orbit_Relation
    where "Orbit_Relation = {(x, y). x ∈ S ∧ y ∈ S ∧ (∃α ∈ G. y = α x)}"
  sublocale orbit: equivalence S Orbit_Relation ⟨proof⟩
end
```

With these declarations `orbit.Class x` denotes the orbit of an `x ∈ S`.

Right cosets $Hx = \{hx \mid h \in H\}$ are defined in the context of a subgroup H of a group G and are the orbits of left translations [16, p. 52]:

```
locale subgroup_of_group = subgroup H G "(·)" 𝟭 + group G "(·)" 𝟭
  for H and G and composition (infixl "·" 70) and unit ("𝟭")
begin
```

In the formalisation right cosets are denoted by `H |· x` rather than, for example, `Right_Coset x`. This is achieved by pulling the declaration up to an auxiliary locale `coset_notation` where `H` is not a parameter.

Now for the correspondence to orbits. Let G_L be the left translations of G. The subset $H_L(G)$ of left translations h_L (in G) for $h \in H$ is a subgroup of G_L and hence a group of transformations of G.

```
interpretation left: left_translations_of_group  ⟨proof⟩
interpretation transformation_group "left.translation ` H" G  ⟨proof⟩
```

Here `left_translation ` H` denotes $H_L(G)$. The $H_L(G)$-orbit of $x \in G$ is the right coset Hx

```
x ∈ G ⟹ H |· x = orbit.Class x.
```

The number of distinct cosets $[G : H]$ is called the index of H in G. For finite groups Lagrange's theorem $|G| = |H|[G : H]$ follows:

```
finite G ⟹ card G = card H * index
```

end

This concludes the section on right cosets. The index is defined as the cardinality of the partition `index = card orbit.Partition`. In the context of `subgroup_of_group` it has no parameters.

3.6 Congruences. Quotient Monoids and Groups

For a monoid $(M, \cdot, 1)$ a congruence E is an equivalence relation that respects composition—if $(a, a') \in E$ and $(b, b') \in E$ then $(a \cdot b, a' \cdot b') \in E$ [16, Def. 1.4]. Composition in M can be lifted to a binary operation $\bar{\cdot}$ of the quotient set M/E, yielding the quotient monoid $(M/E, \bar{\cdot}, \bar{1})$. If the monoid is a group the construction yields a group.

The formalisation takes place in two locales:

```
locale monoid_congruence = monoid + equivalence where S = M +
  assumes "⟦ (a, a') ∈ E; (b, b') ∈ E ⟧ ⟹ (a · b, a' · b') ∈ E"
locale group_congruence = group + monoid_congruence where M = G
```

The lifted binary operation is defined in `monoid_congruence` and denoted with square brackets:

```
([·]) = (λA∈Partition. λB∈Partition. THE C. ∃a∈A. ∃b∈B. C = Class (a · b))
```

`Class` denotes the natural map and `Partition` the quotient set M/E (see Sect. 3.1). The operation maps into the quotient set and commutes with the natural map:

```
⟦A ∈ Partition; B ∈ Partition⟧ ⟹ A [·] B ∈ Partition
⟦a ∈ G; b ∈ G⟧ ⟹ Class a [·] Class b = Class (a · b)
```

That the quotient construction yields a monoid is expressed through a sublocale declaration:

```
sublocale monoid_congruence ⊆
  quotient: monoid Partition "([·])" "Class 1"  ⟨proof⟩
```

In the group case additionally "every $\bar{a}$ is invertible and its inverse is $\overline{a^{-1}}$" [16, p. 55]

```
a ∈ G ⟹ quotient.inverse (Class a) = Class (inverse a)
```

and the construction yields a group:

```
sublocale group_congruence ⊆
  quotient: group Partition "([·])" "Class 1"  ⟨proof⟩
```

For groups, an alternative characterisation of congruences exists. "A subgroup K of G is said to be normal […] if $g^{-1}kg \in K$ for every $g \in G$ and $k \in K$" [16, Def. 1.5],

```
locale normal_subgroup =
  subgroup_of_group K G "(·)" 1
  for K and G and composition (infixl "·" 70) and unit ("1") +
  assumes normal: "⟦ g ∈ G; k ∈ K ⟧ ⟹ inverse g · k · g ∈ K"
```

and there is a one-to-one correspondence between group congruences and normal subgroups: "Let G be a group and $\equiv$ a congruence on G. Then the congruence class $K = \bar{1}$ of the unit is a normal subgroup of G and for any $g \in G$, $\bar{g} = Kg = gK$ […]. Conversely let K be any normal subgroup of G, then $\equiv$ defined by $a \equiv b$ if $a^{-1}b \in K$ is a congruence relation in G whose associated congruence classes are the left (or right) cosets gK" [16, Thm. 1.6]. As a consequence, results on normal subgroups are applicable to group congruences and vice versa. In the formalisation, results are made available to both contexts through mutual sublocale declarations.

Work for the direction from congruences to normal subgroups takes place in `group_congruence` where `Normal = Class 1` represents Jacobson's K. First

```
interpretation subgroup_of_group Normal G "(·)" 1  ⟨proof⟩
```

provides coset notation. Then `g ∈ G ⟹ Normal |· g = Class g` and `g ∈ G ⟹ g ·| Normal = Class g` are shown and

```
sublocale group_congruence ⊆ normal: normal_subgroup Normal G "(·)" 1
  ⟨proof⟩
```

follows. The other direction takes place in `normal_subgroup` where `Congruence` is defined as $a \equiv b$ if $a^{-1}b \in K$. The latter "is equivalent to saying that $b \in aK$, or that b is in the orbit of a relative to the transformation group $K_R(G)$" [16, p. 56].[6]

[6] $K_R(G)$ denotes the group of right translations $k_R : G \to G$ for $k \in K$. Jacobson leaves right translations as an exercise, which had to be formalised as well. For the corresponding work on left translations, see Sect. 3.5 above.

```
interpretation right_translations_of_group ⟨proof⟩
interpretation transformation_group "translation ‘ K" G
  rewrites "Orbit_Relation = Congruence" ⟨proof⟩
```

The rewrites clause effectuates the replacement of the transformation group's equivalence relation by `Congruence`.

The orbit of a transformation group is an equivalence, and since K is a normal subgroup it is also a congruence:

```
sublocale normal_subgroup ⊆ group_congruence where E = Congruence
  rewrites "Normal = K" ⟨proof⟩
```

This concludes the formalisation. Intermediate results temporarily provided by the interpretation commands are either no longer required (transformation groups) or are subsumed by later sublocale declarations (`Normal` is a subgroup).

3.7 Homomorphisms

The fundamental theorems of monoid, group and ring homomorphisms are variants of the factoring of a map α, through the equivalence relation E_α, into a surjective natural map and an injective induced map (Sect. 3.1). The involved sets are algebraic structures and the maps homomorphisms between them.

Homomorphism locales build on those of the respective maps. The locale for monoid homomorphisms is analogous to that for isomorphisms (see Sect. 3.4) but is based on `map` instead of `bijective_map`. Locales for monoid monomorphisms (injective) and epimorphisms (surjective) are declared as well. The hierarchy of the declared locales is shown in Fig. 1. The fundamental theorem of homomorphisms of monoids [16, p. 61] states that a homomorphism η of a monoid M into a monoid M' can be factored, in a unique manner, into an epimorphism ν and a monomorphism $\bar{\eta}$. The formalisation takes place in the dedicated context

```
locale monoid_homomorphism_fundamental = monoid_homomorphism
```

and consists of a sequence of sublocale declarations. The factoring is through the equivalence relation E_η and

```
sublocale fiber_relation η M M'  ⟨proof⟩
```

extends the context with the associated declarations and results. Recall from Sect. 3.1 that `Class` denotes the natural map ν, `induced` the induced map $\bar{\eta}$ and `Partition` the quotient set M/E_η. The equivalence, denoted by `E(η)`, is a congruence

```
sublocale monoid_congruence where E = "E(η)"  ⟨proof⟩
```

and therefore M/E_η a monoid. That the natural and induced maps are homomorphisms is now immediate:

```
sublocale natural:
  monoid_epimorphism Class M "(·)" 1 Partition "([·])" "Class 1"  ⟨proof⟩
sublocale induced:
```

```
monoid_monomorphism induced Partition "([·])" "Class 1"
  "M'" "(·')" "1'" ⟨proof⟩
```

The same reasoning is applicable to group homomorphisms, and Jacobson also works out the details for that. He then presents a second proof, without recourse to congruences, where the factor group G/L, for some normal subgroup L of G contained in the kernel $\ker \eta$ of η, replaces the quotient group. The formalisation for groups follows the second approach, and the locale `normal_subgroup_in_kernel` reflects the described situation:

```
locale normal_subgroup_in_kernel =
  group_homomorphism + contained: normal_subgroup L G "(·)" 1 for L +
  assumes subset: "L ⊆ Ker"
```

The fundamental theorem follows for $L = \ker \eta$:

```
sublocale group_homomorphism_fundamental ⊆
  normal_subgroup_in_kernel where L = Ker  ⟨proof⟩
```

3.8 Rings

Jacobson's definition of rings is: "A ring is a structure consisting of a non-vacuous set R together with two binary compositions $+, \cdot$ in R and two distinguished elements $0, 1 \in R$ such that 1. $(R, +, 0)$ is an abelian group. 2. $(R, \cdot, 1)$ is a monoid. 3. The distributive laws D $a(b + c) = ab + ac$ [and] $(b + c)a = ba + bc$ hold for all $a, b, c \in R$" [16, Def. 2.1]. It translates into the following locale:

```
locale ring =
  additive: abelian_group R "(+)" 0 + multiplicative: monoid R "(·)" 1
  for R and addition (infixl "+" 65) and multiplication (infixl "·" 70)
    and zero ("0") and unit ("1") +
  assumes "⟦ a ∈ R; b ∈ R; c ∈ R ⟧ ⟹ a · (b + c) = a · b + a · c"
    "⟦ a ∈ R; b ∈ R; c ∈ R ⟧ ⟹ (b + c) · a = b · a + c · a"
begin
  notation additive.inverse ("- _" [66] 65)
  notation multiplicative.inverse ("inverse _" [100] 100)
end
```

The additive and multiplicative structures are identified by the qualifiers `additive` and `multiplicative`, respectively. The declaration also shows how notation for unary minus and inverse are obtained.

Ring congruences and homomorphisms are likewise composed from the respective structures of additive group and multiplicative monoid. Jacobson repeats the construction discussed at the end of the previous section for an ideal I contained in the kernel of the ring homomorphism η. This situation is represented by the locale `ideal_in_kernel`. Finally, `ring_homomorphism_fundamental` is the context in which the fundamental theorem of ring homomorphisms is stated and shown. Figure 3 shows the hierarchy of the involved locales.

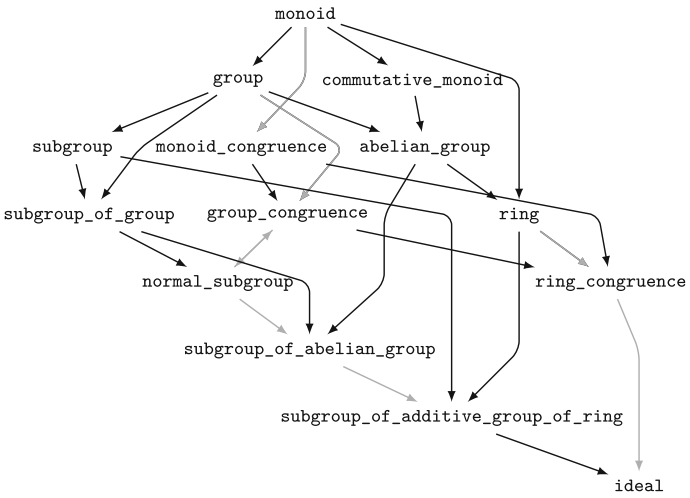


Fig. 2 Locale hierarchy of congruences, normal subgroups and ideals

4 Review

This study completes work the author had announced several years ago [5]. The formalisation effort is probably in the order of three person months.

Finding a suitable module structure for Jacobson's text has not been straightforward. The interrelations are complex. Frequently, decisions had to be revised when it came to using concepts in later sections. The main difficulty was that, of course, the mathematical text does not make it explicit how knowledge needs to be structured so it can be re-used mechanically. An example are translations, which are instances of monoids and groups of transformations. They are instrumental in the proof of Cayley's theorem and later in the characterisation of cosets as orbits. While translations are defined in the context of monoids, it turned out left and right translations better be declared in separate locales, so the knowledge specific to each could be activated individually when needed. On the other hand, the equivalence relation $\sim_G$ of a group G of transformations was defined in the locale for transformation groups and linked to the equivalence relation locale through a sublocale declaration, so this knowledge is automatically available for left and right translations. At times, several attempts to finding the right module structure were needed. To aid library design the concepts of *clone* and *loose coupling* were identified. They are explained below in Sect. 4.4.

Insufficient means for configuring notation were another source of difficulty. This was a design flaw, which was corrected (see also Sect. 4.3). In spite of these difficulties the outcome of the case study is satisfactory. The formalisation is readable, concise, even polished and appears natural.

Figures 1, 2 and 3 show parts of the sublocale graph. The entire graph consists of 51 locales and is too large to reproduce. Table 1 shows examples of Jacobson's notation and their correspondents in the formalisation. The latter is sufficiently concise, with one exception: the notion of invertibility for group transformations is `transformation.sub.invertible`. It occurs in the statement that left translations are invertible, which Jacobson spells out directly, without reference to this predicate. Since it occurs in a few isolated places only, there was no need for finding concise notation.

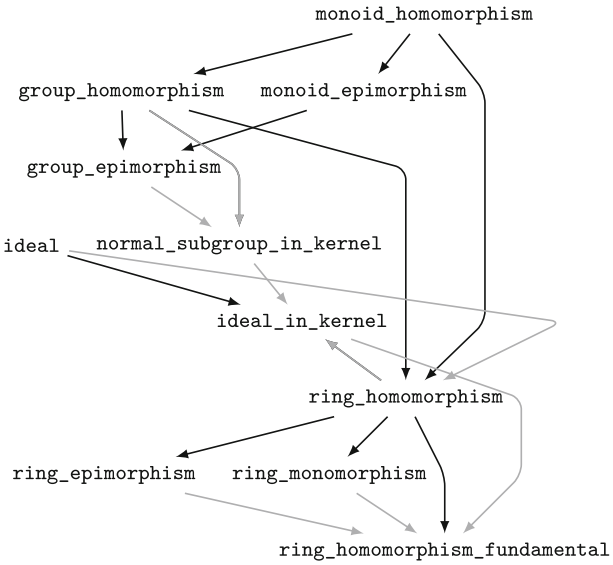


Fig. 3 Locale hierarchy of the fundamental theorem of ring homomorphisms

Table 1 Symbols in Jacobson's text and their representation in the formalisation code

Text	Code	Text	Code
$\bar{a}$	`Class a`	$\sim_G$	`Orbit_Relation`
S/E	`Partition`	$H_L(G)$	`left.translation ‘ H"`
$E(\alpha)$	`Fiber_Relation`	Hx	`H |· x`
$\bar{\alpha}$	`induced`	$\bar{a}\bar{b}$	`Class a [·] Class b`
u^{-1}	`inverse u`	$\equiv$	`Congruence`
a_L	`(a)`$_L$	G/K	`Partition`
G_L	`translation ‘ G"`	R/I	`additive.Partition`
Not used	`transformation.sub.invertible`	$a+I$	`a +| I`

4.1 Representing Algebraic Structures

In Isabelle simple algebraic structures can be represented as type classes [13]. But since a type class can only have a single parameter, expressiveness is restricted—for example, univariate polynomial rings can be represented but not vector spaces in general.

Other approaches to abstract algebra in higher-order logic are based on representing structures as terms rather than types. They are all variations of a technique first worked out by Elsa Gunter in the domain of group theory.

4.1.1 Foundations

In Gunter's formalisation [12] a group structure is a pair of carrier set and binary composition operation and *group* a predicate whose definition is the group axioms: *group* $(G, prod)$ if and only if $(G, prod)$ is a group. Defined operations—here, the identity element *id* and the

operation *inv* mapping an element to its inverse—have the group structure as an additional argument. The definition of *inv* illustrates this:

$$inv\,\mathcal{G}\,x \equiv \varepsilon\,y.\ y \in fst\,\mathcal{G} \wedge (snd\,\mathcal{G}\,y\,x = id\,\mathcal{G})$$

For making the definitions, Hilbert's indefinite choice operator ε is employed.[7]

Gunter derives the fundamental theorem of group homomorphisms and thereby demonstrates that the approach is workable in principle. Theorems in the theory of groups are of the form $group\ \mathcal{G} \Longrightarrow \cdots$, and the need for repeated explicit reference to the group structure makes statements unwieldy. For a limited scope of formalisation the problem can be addressed with *ad hoc* syntax translations quite successfully as demonstrated, for example, by work in the HOL4 system [9].

Locales are connected to Gunter's approach through the concept of the locale predicate. For a locale n with parameters $\overline{x}$ theorems are internally also of the form $n\,\overline{x} \Longrightarrow \cdots$, but the context information is suppressed. While tuples could be used with locales, records or individual parameters are usually preferred.

4.1.2 Record Types

Replacing tuples by records is a straightforward variation, which allows for more suggestive notation such as $prod\ \mathcal{G}$ instead of $snd\ \mathcal{G}$. Syntax such as $x \cdot_{\mathcal{G}} y$ for $prod\ \mathcal{G}\,x\,y$ becomes possible in a systematic manner. The use of records for representing algebraic structures in Isabelle was put forward by Kammüller [17]. Similar techniques for readable notation have also been employed early in type-theoretic provers [2].

Isabelle's records are extensible, and this is useful for working with hierarchies of algebraic structures. However, Isabelle's records only support single inheritance [20]—that is, a record declaration can only refer to one, not two or more, records for extension. Combining the ring structure from its additive group and multiplicative monoid substructures is not possible. This is a fundamental limitation. In Isabelle's algebra library HOL-Algebra [8] this is worked around by extending the multiplicative monoid record by additive operations yielding the ring structure. A record for additive groups is not declared. Instead these have the ring structure type. The unused multiplicative operation fields are left undefined. The declaration of rings in the present study (Sect. 3.8) does not suffer from this limitation.

When using records, concrete syntax such as infix notation is declared outside of locales. Locales merely provide support for concise syntax for records through a special parameter annotation [3, Section 5]. The first annotated parameter is distinguished and can be omitted in theorem statements; $x \cdot_{\mathcal{G}} y$ can be written as $x \cdot y$ if $\mathcal{G}$ is the distinguished parameter. This is similar to but less elaborate than Coq's canonical structures [19]. The notational improvement is limited. It is only effective for parameters. If *Quot* maps a group $\mathcal{G}$ to its quotient group relative to a congruence $\mathcal{C}$ the composition of two elements of the quotient group would be denoted by $x \cdot_{Quot\,\mathcal{G}\,\mathcal{C}} y$.

4.1.3 Local Theories

With the introduction of local theories [14] defined operations became available in locales themselves. Previously, definitions had to be made outside of locales, and records offered

[7] In reproducing Gunter's definition I have changed $(G, prod)$ to $\mathcal{G}$. Her original definition $inv\,(G, prod)\,x \equiv \varepsilon\,y.\ y \in G \wedge (prod\ y\,x = id\,(G, prod))$ does not involve the projections *fst* and *snd*, but many others of her definitions do. Further, since defined operations are well-defined on the carrier set definite choice is sufficient for making definitions. There is no need for the axiom of choice.

a workable means of doing so. With local theories, a definition $c \equiv t$ in a locale n with parameters $\overline{x}$ yields the global definition

$$n\,\overline{x} \Longrightarrow n.c\,\overline{x} \equiv t[\overline{x}]$$

of the qualified constant $n.c$. This is close to what Gunter did. Here, the definition takes place in the background. As in Gunter's approach, it is lifted over the parameters. Additionally, it is logically restricted to $n\,\overline{x}$. Concise notation is achieved by presenting $n.c\,\overline{x}$ as c in the locale context.

Local theories enable representing both signature and theory of an algebraic structure solely with the facilities of locales. The present study demonstrates this approach and shows how concise notation is achieved. In contrast to records, with local theories syntax can be defined on a per-operation basis. This is more flexible than syntax declarations at the record level, but it may also yield more verbose locale declarations. The additional investment pays off easily with more concise notation in propositions and goal states. Analysing goal states is the bread and butter of interactive proof. Any effort to keep notation concise will quickly lead to increased productivity.

In the study, the need for tailored syntax has led to a significant amount of locale declarations that do not represent algebraic structures as such but nevertheless correspond to situations found in the formalised text. Examples are the locales `fiber_relation` and `left_translations_of_group`, which are clones of `map` and `group`, respectively, and also `normal_subgroup_in_kernel` and `ideal_in_kernel`. In total, 51 locales were declared in 2589 lines of formal text (1478 lines when empty lines and comments are excluded). This is significantly more than for comparable work. For example, HOL-Algebra, which is more than ten times larger, contained only 82 locales as of November 2018.

4.1.4 Interpretation

Interpretation is essential for transferring abstract results to situations of use, be they concrete or abstract themselves. The need was already recognised by Gunter, who presents instances of her group theory development for the integers and modular arithmetic [12, Appendices D and E]. How the interpreted theories were arrived at is not recorded. Presumably appropriate code for creating them was provided in ML, the meta language of the HOL system.

The interpretation facilities of locales proved flexible enough for representing Jacobson's lines of argument faithfully through the formalised sections of his book. Notable examples of interpretation are encountered when the properties of quotient structures are established and when transformation groups are used. The latter are key to Cayley's theorem and, since cosets are the orbits of group elements under translation, also to Lagrange's theorem.

The initial design of locales [18] did not provide interpretation. Incidentally, Kammüller and Paulson show that the quotient construction over a group yields a group but do not make use of that result.[8] Their proof of Lagrange's theorem is elementary. It does not make the notion of transformation group explicit. The lack of interpretation is also apparent in their proof of Sylow's first theorem. The formalised proof, due to Wielandt, is based on the action of the group, by left translation, on a set of subsets of certain cardinality. Jacobson outlines the technique [16, p. 83]. The required concepts are encoded in the locales `sylow` and `sylow_central`, which describe the specific group action required in that proof. Only

[8] Proofs are preserved in Isabelle's source code repository: https://isabelle.in.tum.de/repos/isabelle/file/7e6cdcd113a2/src/HOL/GroupTheory/.

later, when locale interpretation was available, von Raumer [24] factored out the concept of group action in his formalisation of further Sylow theorems and applied it, via interpretation, in the various proofs.

4.2 Quantitive Analysis

Locales seek to aid formalisation of mathematics in two ways: through concise and readable syntax for the objects of discourse, and by facilitating reuse throughout a mathematical development. Both are expected to have an impact on the size of a formal development.

4.2.1 Notation

Locales succeed at hiding the complex nature of objects of discourse. For example, for a normal subgroup K multiplication of cosets enjoys the property $(gK)(hK) = ghK$. In the presented formalisation, within `normal_subgroup`, this statement is

```
⟦g ∈ G; h ∈ G⟧ ⟹ g ·| K [·] (h ·| K) = g · h ·| K
```

and consists of 49 characters including whitespace. In contrast, the fully expanded internal version is quite large:

```
⟦normal_subgroup K G composition unit; g ∈ G; h ∈ G⟧
⟹ monoid_congruence.quotient_composition G composition
     (normal_subgroup.Congruence K G composition unit)
     (coset_notation.Left_Coset composition g K)
     (coset_notation.Left_Coset composition h K) =
    coset_notation.Left_Coset composition (composition g h) K
```

This is partly due to the module system relying on qualified names for accommodating name spaces and long parameter names. Gunter's version of the same statement [12, p. 28, theorem `QUOT_PROD`], while being considerably shorter, has the same structure:

```
|- NORMAL(G,prod)N ==> (!x y. G x /\ G y ==>
    (quot-prod(G,prod)N (LEFT_COSET(G,prod)N x)
    (LEFT_COSET(G,prod)N y) =
    LEFT_COSET(G,prod)N(prod x y)))
```

By dropping qualifiers and using short parameter names a shortened statement can be arrived at. It is of similar size than Gunter's (150 vs. 146 characters):

```
⟦normal_subgroup K G c u; g ∈ G; h ∈ G⟧
⟹ quotient_composition G c (Congruence K G c u) (Left_Coset c g K)
     (Left_Coset c h K) =
    Left_Coset c (c g h) K
```

Table 2 Count of theorems and their reduction in size between non-locale and locale version

Section	Count	Ratio			
		(a)		(b)	
		Min	Max	Min	Max
Maps	3	1.14	1.87	1.14	1.87
Factoring a map	35	1.42	4.89	1.37	3.00
Abstract monoids	1	3.62	3.62	2.62	2.62
Monoids of transformations	2	1.84	2.12	1.84	2.12
Abstract groups	27	1.84	5.75	1.20	2.75
Groups of transformations	6	1.81	2.83	1.81	2.54
Isomorphism	1	2.30	2.30	1.41	1.41
Cayley's theorem for monoids	9	1.41	6.67	1.14	3.67
Cayley's theorem for groups	4	1.41	5.05	1.13	5.04
Right translations (exercise)	12	2.95	6.67	2.25	5.08
Commutativity	0	–	–	–	–
Orbits. cosets of a subgroup	20	1.65	9.00	1.23	4.90
Quotient monoids	3	2.50	3.94	1.52	2.27
Normal subgroups, factor groups	17	2.57	10.64	1.61	5.45
Homomorphisms of monoids	3	3.71	4.08	2.00	2.23
Homomorphisms of groups	20	2.31	7.72	1.40	5.71
Rings—elementary properties	4	4.56	4.71	2.06	2.17
Ideals, quotient rings	5	3.53	6.63	1.49	3.08
Homomorphisms of rings	3	2.82	3.21	1.35	1.53
Overall	175	1.14	10.64	1.13	5.71

(a) refers to the theorems in the formalisation, (b) to variants of these with with unqualified identifiers and single-character schematic variables; Min refers to the smallest reduction for a theorem in the section, Max to the greatest. Counted are the characters, including whitespace, of printed theorems. Symbols such as "α" and "$\Longrightarrow$" are considered single characters

An evaluation of the effect of locales on statement size by direct comparision of this study with Gunter's work is not possible—her work only covers group theory and further is based on a different textbook, so most theorem statements differ fundamentally—yet it emerges from the above consideration that for a direct formalisation of Jacobson's text using Gunter's approach theorem sizes can be expected to be roughly like those of the shortened internal versions of the present study. Table 2 gives an overview over the 175 theorems of the study. Column (a) lists the size reduction with respect to the full, column (b) with respect to the shortened internal versions. For a direct formalisation using Gunter's approach statements up to 5.7 times the size when using locales can be expected.

4.2.2 Structure

The ultimate benchmark is the mathematical text itself. For van Benthem Jutting's formalisation [23] of Landau's textbook *Grundlagen der Analysis* on complex arithmetic, the sizes of the Automath code and the formalised source were compared. According to Wiedijk [26],

de Bruijn calls what is lost in conciseness in the translation the *loss factor* and observes that it is constant: "it does not increase if we go further in the book". Wiedijk popularised de Bruijn's loss factor as *de Bruijn factor*.

For the present study, in the Isabelle theory files each declaration is annotated with the corresponding source location (page and line numbers) in the textbook. Further, statements found there explicitly are marked as theorems (unless they have been translated in locale constructs) others as lemmas. This makes the correspondence very explicit, and has been helpful in keeping organised. It also enables providing quantitative information. 400 lines of Jacobson's text, which corresponds to 11.4 pages of 35 lines each, were formalised, yielding about 1500 lines of formal development. Empty lines, source code comments and other passages of English text in the code are excluded from this count.

A precise breakdown is shown in Table 3. For the identified sections the ratio between code and text size (measured in lines) ranges from 1.0 to 6.5, except for right translations, which are left as an exercise to the reader, and for which the ratio therefore is much higher—28. It can be noted that the ratio is generally higher for sections on groups than on monoids. Certainly the need for numerous technical lemmas to simplify computation in abstract groups and the fact that the work on group homomorphisms is not based on that on monoid homomorphisms play a role here. On the other hand, the ratio drops again for the sections on rings, and we can conclude that as for van Benthem Jutting's formalisation the ratio does not increase further into the text and reuse of monoid and group parts for rings works well.[9]

4.3 Improvements to Locales

The case study triggered improvements to locales, and in comparison to my detailed account on locales [6] these changes were made:

- The keyword for rewrite clauses was changed, from **where** to **rewrites**. This improves the readability, in particular of named locale instances, which now use **where** exclusively.
- Handling of syntax was modified so that **notation** can be used for changing notation. Previously **abbreviation** was used. Syntax in locales is now more flexible. Notation can be more natural and conciser.
- It was necessary to make rewrite clauses part of locale instances. Previously, rewrite rules were processed after instantiation. The new approach is conceptually cleaner and provides means for avoiding syntax conflicts while locale expressions are parsed.

These changes had only a small impact on existing formalisations. The renamings required for the first change in the Isabelle distribution and the Archive of Formal Proofs could be handled in about one day. The other improvements are direct outcomes of the present study and were required for its successful completion. They were published with Isabelle 2016 and 2018 respectively. While the second change had no impact on existing developments in the Isabelle distribution and the Archive of Formal Proofs at all, it was important to ensure that it introduced no changes in the rendering of proof states. The third change did require surprisingly few isolated changes, which were not difficult to make.

[9] Wiedijk [26] supposes that the mathematical text is available in computer-readable form and suggests the factor be computed by comparing sizes of compressed files of the formalisation and its mathematical source. I have chosen to simply compare numbers of lines as is common practice when measuring code size in computer science.

Table 3 Line count of Isabelle code compared to Jacobson's text. Excludes empty lines, comments in source code etc

Section	Code	Text	Ratio
Maps	38	24	1.58
Factoring a map	241	58	4.16
Abstract monoids	27	8	3.38
Monoids of transformations	13	5	2.60
Abstract groups	130	20	6.50
Groups of transformations	29	8	3.63
Isomorphism	31	15	2.07
Cayley's theorem for monoids	48	15	3.20
Cayley's theorem for groups	31	8	3.88
Right translations (exercise)	85	3	28.33
Commutativity	3	3	1.00
Orbits. cosets of a subgroup	146	32	4.56
Quotient monoids	16	14	1.14
Normal subgroups, factor groups	192	48	4.00
Homomorphisms of monoids	51	32	1.59
Homomorphisms of groups	181	30	6.03
Rings—elementary properties	44	20	2.20
Ideals, quotient rings	88	28	3.14
Homomorphisms of rings	84	28	3.00
Total	1478	399	3.70

4.4 Reasoning Patterns

Certain locale constructions were used repeatedly in the study. They are identified here to provide guidance. In the diagrams, in addition to the arrows $x \twoheadrightarrow y$ for import and $x \rightarrow y$ for sublocale declarations, $x \dashrightarrow y$ denotes temporary interpretation.

4.4.1 Clone

A clone y of a locale x is obtained as follows:

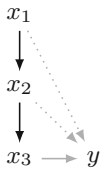
$$\textbf{locale } y = x$$

Both locales have the same parameters and specification, but their bodies differ. While y inherits all declarations of x, the latter does not inherit any of the former. Clones are useful for knowledge separation. Knowledge that is implied by a specification but does not belong to its main body, for example, might better be placed inside a clone. Clones may also be necessary for avoiding infinite chains of interpretation [4, Section 7.1].

The study contains several instances of the clone pattern. For example, `fiber_relation` (Sect. 3.1) is a clone of `map`, `left_translations_of_monoid` (Sect. 3.4) a clone of `monoid`,

and the locales for the fundamental theorems of homomorphisms (Sects. 3.7 and 3.8) are clones of the respective homomorphism locales.

4.4.2 Bootstrapping



Let $x_1, \ldots, x_n$ be a locale hierarchy and the goal to interpret x_n, via some morphism ϕ in y. Let ϕx_i denote the locale instance obtained by applying ϕ to the parameters (and defined operators) of x_i. Establishing the interpretations $\phi x_1, \ldots, \phi x_n$ one after the other may be significantly simpler then showing ϕx_n directly. Be it that notation is introduced that makes the subsequent proof more concise, be it that suitable knowledge is introduced or, as happens frequently, both. In analogy to the start up of a computer system, where simple software enables the loading of more complex software from an input device, this process is called *bootstrapping*. In the context of y

interpretation ϕx_i ⟨*proof*⟩

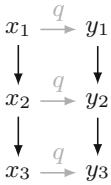
is established subsequently for $i = 1, \ldots, n-1$ and

sublocale ϕx_n ⟨*proof*⟩

follows eventually. All interpretations but the final one may be temporary.

In the study, bootstrapping occurs in several places. A prominent example is discussed in Sect. 3.6 where in the context of `left_translations_of_monoid` three sublocale declarations lead to the statement that left translations are monoid isomorphisms.

4.4.3 Functor



Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be hierarchies of algebraic structures represented by locales,

and let ϕ be a functor from (x_i) to (y_i). For example, let (x_i) and (y_i) be ring hierarchies and ϕ denote the construction of univariate polynomials. The connection of the hierarchies is achieved by a sequence of declarations

$$\textbf{sublocale } y_i \subseteq \phi x_i \ \langle proof \rangle.$$

All instances ϕx_i will share the same qualifier, which is indicated by the letter q in the diagram. In practice, declarations for the construction of ϕ lead to a hierarchy $y'_1, \ldots, y'_n$ of locales in which the sublocale declarations take place.

An instance of the functor pattern in the study is the quotient construction for monoids, groups and rings. The locales `monoid`, `group` and `ring` form the source hierarchy (x_i), and `monoid_congruence`, `group_congruence` and `ring_congruence` form the target hierarchy (y'_i); `abelian_group`, which is also part of the source hierarchy, has no corresponding target and is not part of the functor (Fig. 2).

4.4.4 Loose Coupling

Let x be a locale containing a construction leading to some result—for example, a theorem *th*. Let y be a locale in which the result from x is to be used (in the general case, via some morphism ϕ). If importing the entire context of ϕx into y permanently is not desirable, loose coupling between x and y can be achieved through a temporary interpretation of ϕx in y. After establishing the desired result *th* in y, based on ϕx, closing the context for y discards ϕx but *th* remains.

Loose couplings are used in the study to transfer Cayley's theorems back from `left_translations_of_monoid` to `monoid` and from `left_translations_of_group` to `group`. The monoid case is discussed in detail towards the end of Sect. 3.6

4.4.5 Mutual Sublocales

It occasionally happens that the same concept is arrived at through different paths. In the case of locales, the hierarchy can be consolidated through mutual sublocale declarations.

In the study this was made use of for the locale pairs `partition` and `equivalence` and `normal_subgroup` and `group_congruence`. Typically, defined operations of one locale are mapped to parameters of the other. Providing the correct mappings is essential for avoiding nontermination of the sublocale relation.

When I first realised this possibility [6, Section 5.3.2], I assumed it to be of theoretical interest only. Writers of mathematical libraries will conceivably be able to structure developments in a way that mutual sublocales are not required. Actually the feature has turned out to be relevant for being able to adequately reflect the structure of a mathematical text.

5 Discussion

What can be learnt from the study for the formalisation of algebra in Isabelle, and in simple type theory in general?

5.1 Complex Structures

Without any module support, syntax can quickly become unreadable and automation poor—for example, when groups themselves are denoted by large expressions.

Such situations were dealt with by dedicated locales, in which appropriate definitions were made. Examples are `monoid_congruence` and `group_congruence` where `Partition` is the quotient set, the structure's carrier set, and `[·]` the binary operation (see Sect. 3.6). Given that the definition of the binary operation

```
([·]) = (λA∈Partition. λB∈Partition. THE C. ∃a∈A. ∃b∈B. C = Class (a · b))
```

involves selection and bounded abstraction, working without a definition would have been unwieldy. Not only is the syntax concise, also the configuration of reasoning tools (simplifier and classical reasoner) inherited from the locales for monoids and groups is not interfered with by conflicting setups of these tools for bounded abstraction.

The loss factors of the sections on quotient monoids and factor groups are not higher than those of the sections on abstract monoids and groups, respectively (Table 3). This supports that automation worked well, and that was also my experience while working on those proofs.

Exceptions to the approach were made where definitions outside the locale yielding concise notation within the locale were already available. A prominent example are monoids and groups of transformations, where binary operation and unit element were denoted by `compose S` and `identity S`, respectively. Definitions in the locales would merely have hidden the set parameter `S`.

The functor pattern describes the general situation. The locales y_i' on the right-hand side should have appropriate definitions that the parameters of the x_i on the left-hand side can be mapped to. Additional definitions in y_i' for concepts corresponding to defined operations of x_i can be useful as well. These can be mapped via rewrites clauses.

The functor pattern also explains what is required for transferring the approach to other provers based on simple types such as the HOL4 system. Definitions in a locale correspond to global definitions (see Sect. 4.1.3) and are therefore not essential. Being able to lift knowledge from one context x_i to another y_i' is. The means for achieving this is, of course, theory interpretation, which is provided by locales through the sublocale and interpretation commands.

5.2 Conditional Definitions

Definitions in the study are conditional—that is, operations are only defined on the intended domain and not outside. This is in contrast to common practice in the HOL community, where often dummy values are chosen such that unnecessary proof obligations of the form "all values are in the carrier" can be avoided. For example, in a formalisation of homological

algebra I was involved with *completions* were chosen to represent group homomorphisms [1]. These map values outside the domain to the unit element of the co-domain.

The motivation for choosing conditional definitions in the present study was, besides being faithful to Jacobson's text, understanding whether they would pose problems with rewrites clauses, where conditional rewrite rules are not allowed. No such problems were encountered. Conditional definitions did cause additional work, but a small number of lemmas on undefinedness (three in set and four in group theory) helped resolve proof obligations largely through automation. Proof scripts did not grow significantly in size.

Conditional definitions do not necessarily lead to conditional theorems, as illustrated by the factorisation of a map or homomorphism α into induced and natural parts (Sect. 3.1):

```
compose S induced Class = α
```

This theorem holds for maps and homomorphisms, and Jacobson proves it at the most general level—for maps. Had completion semantics been adopted, `compose S` would have had to be replaced by an operator returning, outside `S`, the unit element of the co-domain. This would only have worked for homomorphisms, not maps, since in the context of the latter no distinguished element is available. Using dummy elements is appropriate for isolated applications but in libraries they pose the risk of precluding generalisation.

5.3 Limitations of Locales

Locales provide means for structuring formal developments, but they do not change the underlying logic, so they do not increase expressiveness. This is a fundamental limitation. On the other hand, locales work on top of any logic provided there is substitution [6, Section 3.1], so they could be provided for a wide range of provers.

Locales proved effective for reusing theorems, but there is room for improvement with regard to configuring notation. While it is often desirable to use the original notation for an operation from a dependency, sometimes it is not. For example, the notations for the binary operations of a homomorphism's domain and co-domain need to differ. Locales have a simple heuristic to avoid conflicts: notation is retained as long as no parameter is renamed or instantiated. This heuristic repeatedly necessitated redeclaring monoid syntax for groups as the carrier set parameter is renamed from `M` to `G`. This could be improved, by separating signature morphisms from locale expressions. A renaming or instantiation could then be declared once, along with notation, and reused in multiple locale expressions. Since locale declarations are infrequent (compared to theorems) this was considered a minor annoyance that could be lived with.

Locales aim at being flexible, and they succeed in many ways: a locale can be revisited and extended—for example, by definitions and theorems—at any point in a formal development. Even the locale hierarchy can be changed without modifying the locale declarations. What is currently not possible is amending an existing locale interpretation with additional rewrites clauses. This functionality would enable, when a definition in a locale is made, mapping it to an existing concept in the interpretation target. The feature is currently only available through internal APIs, for use by the class package [13], and only for what corresponds to the interpretation command, not for sublocale. Making it available in Isar for interpretation would be straightforward. For sublocale the matter is more involved: the sublocale graph

is labelled with morphisms, and I was never able to convince myself that amending these morphisms in the described manner would be robust enough. It is conceivable that such an amendment could break the locale hierarchy in unexpected ways. For symmetry, the feature is not provided for the interpretation command either.[10]

5.4 Abstract Algebra in Isabelle

Work on Isabelle's library of abstract algebra HOL-Algebra [8] started before the design of locales was complete. Definitions in locales only became available later [14]. Algebraic structures were therefore represented by record types (as outlined in Sect. 4.1.2). Achieving concise notation is awkward. To illustrate, let $Quot\ \mathcal{G}\ \mathcal{C}$ again denote the quotient group of $\mathcal{G}$ relative to the congruence $\mathcal{C}$. The locale for the quotient group situtation would extend the group locale by an additional parameter $\mathcal{Q}$ and an additional assumption $\mathcal{Q} \equiv Quot\ \mathcal{G}\ \mathcal{C}$, such that $x \cdot_{\mathcal{Q}} y$ could be written instead of $x \cdot_{Quot\ \mathcal{G}\ \mathcal{C}} y$. The additional parameter and assumption would complicate using that locale. The case study shows that this can be improved in two ways:

1. Make the quotient group locale a clone of the group locale and define the quotient group $\mathcal{Q}$ within that.
2. As their means for composing structures is limited, avoid Isabelle/HOL's records.

A full assessment whether improving HOL-Algebra in this way is feasible (and worth the effort) is beyond the scope of this discussion. I expect that the first change is fairly straightforward, but changes to the record package may be required for compatibility with definitions in locales. The second change will have a significant impact on work that depends on the library. This can possibly be mitigated by applying the functor pattern so that revised, record-free locales are mapped to existing, record-based ones, and dependent material could be converted gradually.

Since record types are not needed, locales can support abstract algebra in Isabelle's set theory, Isabelle/ZF, equally well. Locales could, in principle, be provided for a wide range of logics and provers. Whether provers based on dependent types, which are a powerful means of representing algebraic structures themselves, could benefit from locales is less clear.

6 Conclusion

Gunter's early work on algebra in simple type theory [12] is foundational and applied in variations in many formalisations. Locales are based on her approach too, and it is lifted to a module system with support for concise local notation, name space management, flexible means of creating and maintaining module hierarchies, and integration with the proof language. In my earlier, detailed account [6], locales are defined in terms of their operational semantics. There, their capabilities are explored by comparison to other structuring mechanisms including type classes, ML-style modules [15] and the module system of Coq [22]. The present case study complements this work by studying the particulars of formalising a mathematical text selected to be challenging from a structural, but not necessarily mathematical, perspective.

[10] This paragraph reproduces my response to a post in the Isabelle Users mailing list: https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2019-September/msg00074.html.

The formalised corpus sets out with equivalence classes and the factoring of maps and extends to the fundamental theorems of monoid, group and ring homomorphisms. Cosets are identified to be the orbits of translations, transformations on the underlying group, and their properties derived through this route. Quotient classes are shown to be the cosets of normal subgroups. The results derived on homomorphisms pick up from there, and their fundamental theorems build on the factoring of maps in a natural way.

An important result of the study is that locales alone are sufficient for representing algebraic structures in a concise and modular manner. Record types are not required. When choosing to group the parameters of an algebraic structure into a single record object it needs to be kept in mind that this approach tends to yield longer statements due to operations having additional record arguments that will be inferred only in simple situations. Also, Isabelle's record types only allow linear structure hierarchies. By avoiding records in this case study, a natural declaration of the ring structure was straightforward.

Locales achieve concise statement notation by hiding parts of the internal representation of objects of discourse in the logic calculus, and by applying context-specific concrete syntax. Locales also achieve concise proofs by providing powerful means of reuse. The size ratio of formal to "pen-and-paper" development, de Bruijn's loss factor, did not increase throughout the work and it was possible to lift the monoid and group results to rings in the formalisation as effectively as in the formalised text.

Locales declared in the case study are not confined to algebraic structures proper such as monoids, groups and rings. More complex and sometimes auxiliary situations—for example, that of a subgroup contained in the kernel of a homomorphism—are captured with locales as well. The module system makes transferring results from such intermediate constructs to contexts of use easy.

Several lessons can be learnt for formalisations in Isabelle. *Reasoning patterns* of locale use in the case study were identified and described. While these patterns occurred in the context of abstract algebra, they appear rather more intrinsic to how locales work than to the application domain, and so these patterns will also be useful in other domains. Dummy values, a means for avoiding undefinedness in definitions, need to be used with care, especially in libraries, since they can preclude desirable generalisation. In general, the case study provides a better specimen of using locales than Isabelle's library of abstract algebra, HOL-Algebra [8], since it is based on more modern techniques. Guidance on how to proceed with the latter was given, but a full assessment is beyond the scope of this work.

References

1. Aransay, J., Ballarin, C., Rubio, J.: A mechanized proof of the basic perturbation lemma. J. Autom. Reason. **40**(4), 271–292 (2008)
2. Bailey, A.: The Machine-Checked Literate Formalisation of Algebra in Type Theory. Ph.D. Thesis, University of Manchester (1998)
3. Ballarin, C.: Locales and locale expressions in Isabelle/Isar. In: Berardi, S., Coppo, M., Damiani, F. (eds.) Types for Proofs and Programs, TYPES 2003, Torino, Italy, LNCS 3085, pp. 34–50. Springer, Berlin (2004)
4. Ballarin, C.: Tutorial to locales and locale interpretation. In: Lambán, L., Romero, A., Rubio, J. (eds.) Contribuciones Científicas en honor de Mirian Andrés Gómez. Servicio de Publicaciones de la Universidad de La Rioja, Logroño, Spain, Also part of the Isabelle user documentation (2010)
5. Ballarin, C.: Reading an algebra textbook. In: Lange C. et al. (eds.) CEUR Workshop Proceedings 1010. Intelligent Computer Mathematics, Bath (2013)
6. Ballarin, C.: Locales: a module system for mathematical theories. J. Autom. Reason. **52**(2), 123–153 (2014)

7. Ballarin, C.: A case study in basic algebra. Archive of Formal Proofs https://isa-afp.org/entries/Jacobson_Basic_Algebra.html (2019)
8. Ballarin, C. et al.: The Isabelle/HOL algebra library. Part of the Isabelle distribution, https://isabelle.in.tum.de/library/HOL/HOL-Algebra/
9. Chan, H.-L., Norrish, M.: Mechanisation of AKS algorithm: Part 1–the main theorem. In: Urban, C., Zhang, X. (eds.) Interactive Theorem Proving, ITP 2015, Nanjing, China, LNCS 9236, pp. 117–136. Springer, Berlin (2015)
10. Farmer, W.M.: Theory interpretation in simple type theory. In: Heering, J., Mainke, K., Möller, B., Nipkow, T. (eds.) Higher-Order Algebra, Logic, and Term Rewriting, HOA '93, Amsterdam, The Netherlands, LNCS 816, pp. 96–123. Springer, Berlin (1994)
11. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992)
12. Gunter, E.L.: Doing Algebra in Simple Type Theory. Technical Report MS-CIS-89-38, University of Pennsylvania (1989)
13. Haftmann, F., Wenzel, M.: Constructive type classes in Isabelle. In: Altenkirch, T., McBride, C. (eds.) Types for proofs and programs, TYPES 2006, Nottingham, UK, LNCS 4502, pp. 160–174. Springer, Berlin (2007)
14. Haftmann, F., Wenzel, M., (2009) Local theory specifications in Isabelle, Isar. In: Berardi S., Damiani F., de'Liguoro U. (eds) Types for Proofs and Programs, TYPES, Torino, Italy, LNCS 5497, pp. 153–168. Springer, Berlin (2008)
15. Harper, R., Pierce, B.C.: Design considerations for ML-style module systems. In: Pierce, B.C. (ed.) Advanced Topics in Types and Programming Languages. MIT Press, Cambridge (2005)
16. Jacobson, N.: Basic Algebra, vol. I, 2nd edn. Freeman, Dallas (1985)
17. Kammüller, F.: Modular Reasoning in Isabelle. Ph.D. Thesis, University of Cambridge, Computer Laboratory, Also Technical Report No. 470 (1999)
18. Kammüller, F., Wenzel, M., Paulson, L.C.: Locales: a sectioning concept for Isabelle. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) Theorem Proving in Higher Order Logics: TPHOLs'99, Nice, France, LNCS 1690, pp. 149–165. Springer, Berlin (1999)
19. Mahboubi, A., Tassi, E.: Canonical structures for the working Coq user. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving, ITP 2013, Rennes, France, LNCS 7998, pp. 19–34. Springer, Berlin (2013)
20. Naraschewski, W., Wenzel, M.: Object-oriented verification based on record subtyping in higher-order logic. In: Theorem Proving in Higher Order Logics, LNCS 1479. Springer, pp 349–366 (1998)
21. Paulson, L.C.: Defining functions on equivalence classes. ACM Transactions on Computational Logic **7**(4), 658–675 (2006)
22. Soubiran, E.: Modular Development of Theories and Name-Space Management for the Coq Proof Assistant. Ph.D. Thesis, École Polytechnique (2012)
23. van Benthem Jutting, L.S.: Checking Landau's "Grundlagen" in the Automath System. Ph.D Thesis, Technische Hogeschool Eindhoven (1977)
24. von Raumer, J.: Secondary Sylow Theorems. Archive of Formal Proofs, https://isa-afp.org/entries/Secondary_Sylow.html, (2014)
25. Wenzel, M.: Isabelle/Isar–a generic framework for human-readable proof documents. Stud. Log. Gramm. Rhetor. **10**(23), 277–298 (2007). (Festschrift in Honour of Andrzej Trybulec)
26. Wiedijk, F.: The de Bruijn factor. https://www.cs.ru.nl/~freek/factor/factor.pdf (2000)