Introduction to the Extension of the Framework Types-To-Sets for Isabelle/HOL

Mihails Milehins

March 17, 2025

Abstract

In [19, 21], Ondřej Kunčar and Andrei Popescu propose an extension of the logic *Isabelle/HOL* and an associated algorithm for the relativization of *type-based theorems* to more flexible *set-based theorems*, collectively referred to as *Types-To-Sets*. One of the aims of their work was to open an opportunity for the development of a software tool for applied relativization in the implementation of the logic Isabelle/HOL in the proof assistant *Isabelle* [27]. In this document, we provide a prototype of a software framework for the interactive automated relativization of definitions and theorems in Isabelle/HOL, developed as an extension of the proof language *Isabelle/Isar* [31, 32]. The software framework incorporates the implementation of the proposed extension of the logic and associated tools provided in [19] and improved further in [14] by Fabian Immler and Bohua Zhan, and builds upon some of the ideas for further work expressed in [14] and [21].

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries and the persons responsible for the development of Types-To-Sets [19, 21] and its official extensions [14, 12] for providing explanations of the existing functionality of the framework and some of the ideas that laid the foundation of this work. Special thanks go to Fabian Immler for the conceptual design of the commands **tts_context**, **tts_lemmas** and **tts_lemma**, to Andrei Popescu for trying the software and providing feedback, and to Kevin Kappelmann for explaining certain aspects of [16]. Furthermore, the author would like to acknowledge the positive impact of [29] and [33] on his ability to code in Isabelle/ML [25, 33]. The author would also like to acknowledge the positive role that the numerous Q&A posted on the Stack Exchange network (especially Stack Overflow and TeX Stack Exchange) played in the development of this work. Finally, the author would like to express gratitude to all members of his family and friends for their continuous support.

Contents

ETTS: Reference Manual

1.1 Introduction

1.1.1 Background

The *standard library* that is associated with the object logic Isabelle/HOL and provided as a part of the standard distribution of Isabelle [1] contains a significant number of formalized results from a variety of fields of mathematics (e.g., order theory, algebra, topology). Nevertheless, using the argot that was promoted in the original publication of Types-To-Sets [19], the formalization is performed using a type-based approach. Thus, for example, the carrier sets associated with the algebraic structures and the underlying sets of the topological spaces, effectively, consist of all terms of an arbitrary type. This restriction can create an inconvenience when working with mathematical objects induced on a subset of the carrier set/underlying set associated with the original object (e.g., see [14]).

To address this limitation, several additional libraries were developed upon the foundations of the standard library (e.g., *HOL-Algebra* [5] and *HOL-Analysis* [2]). In terms of the argot associated with Types-To-Sets, these libraries provide the set-based counterparts of many type-based theorems in the standard library, along with a plethora of additional results. Nonetheless, the proofs of the majority of the theorems that are available in the standard library are restated explicitly in the libraries that contain the set-based results. This unnecessary duplication of efforts is one of the primary problems that the framework Types-To-Sets is meant to address.

The framework Types-To-Sets offers a unified approach to structuring mathematical knowledge formalized in Isabelle/HOL: potentially, every type-based theorem can be converted to a set-based theorem in a semi-automated manner and the relationship between such type-based and set-based theorems can be articulated clearly and explicitly [19]. In this document, we describe a particular implementation of the framework Types-To-Sets in Isabelle/HOL that takes the form of a further extension of the language Isabelle/Isar with several new commands and attributes (e.g., see [34]).

1.1.2 Prerequisites and conventions

A reader of this document is assumed to be familiar with the proof assistant Isabelle, the proof language Isabelle/Isar, the meta-logic Isabelle/Pure and the object logic Isabelle/HOL, as described in, [27, 34], [31, 32, 34], [28, 34] and [20], respectively. Familiarity with the content of the original articles about Types-To-Sets [19, 21] and the first large-scale application of Types-To-Sets (as described in [14]) is not essential but can be useful.

The notational conventions that are used in this document are approximately equivalent to the conventions that were suggested in [19], [20] and [21]. However, a disparity comes from our use of explicit notation for the *schematic variables*. In Isabelle/HOL, free variables that occur in the theorems at the top-level in the theory context are generalized implicitly, which may be expressed by replacing fixed variables by schematic variables [35]. In this article, the schematic variables will be prefixed with the question mark "?", like so: $?a$. Nonetheless, explicit quantification over the type variables at the top-level is also allowed: $\forall \alpha.\phi\,[\alpha]$. Lastly, the square brackets may be

used either for the denotation of substitution or to indicate that certain types/terms are a part of a term: $t\,[?\alpha]$.

1.1.3 Previous work

Relativization Algorithm

Let ${}_{\alpha}(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$ denote

$$\begin{array}{l}(\forall x_\beta.\mathsf{Rep}\ x \in U) \wedge \\ (\forall x_\beta.\mathsf{Abs}\ (\mathsf{Rep}\ x) = x) \wedge \\ (\forall y_\alpha.y \in U \longrightarrow \mathsf{Rep}\ (\mathsf{Abs}\ y) = y)\end{array},$$

let $\rightsquigarrow$ denote the constant/type *dependency relation* (see subsection 2.3 in [19]), let $\rightsquigarrow^{\downarrow}$ be a *substitutive closure* of the constant/type dependency relation, let R^{+} denote the transitive closure of the binary relation R, let Δ_c denote the set of all types for which c is *overloaded* and let $\sigma \notin S$ mean that σ is not an instance of any type in S (see [19] and [20]).

The framework Types-To-Sets extends Isabelle/HOL with two axioms: *Local Typedef Rule* (LT) and the *Unoverloading Rule* (UO). The consistency of Isabelle/HOL augmented with the LT and the UO is proved in Theorem 11 in [20].

The LT is given by

$$\frac{\Gamma \vdash U \neq \varnothing \quad \Gamma \vdash \left(\exists \mathsf{Abs}\ \mathsf{Rep}._\sigma(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \longrightarrow \phi\right)}{\Gamma \vdash \phi}\ \beta \notin U, \phi, \Gamma$$

Thus, if β is fresh for the non-empty set $U_{\sigma\ \mathsf{set}}$, the formula ϕ and the context Γ, then ϕ can be proved in Γ by assuming the existence of a type β isomorphic to U.

The UO is given by

$$\frac{\phi}{\forall x_\sigma.\phi\,[x_\sigma/c_\sigma]}\ \forall u \text{ in } \phi.\neg(u \rightsquigarrow^{\downarrow +} c_\sigma);\ \sigma \notin \Delta_c$$

Thus, a *constant-instance* c_σ may be replaced by a universally quantified variable x_σ in ϕ, if all types and constant-instances in ϕ do not semantically depend on c_σ through a chain of constant and type definitions and there is no matching definition for c_σ.

The statement of the *original relativization algorithm* (ORA) can be found in subsection 5.4 in [19]. Here, we present a variant of the algorithm that includes some of the amendments that were introduced in [14], which will be referred to as the *relativization algorithm* (RA). The differences between the ORA and the RA are implementation-specific and have no effect on the output of the algorithm, if applied to a conventional input. Let $\bar{a}$ denote a finite sequence $a_1, \ldots, a_n$ for some positive integer n; let Υ be a *type class* [26, 30, 10] that depends on the overloaded constants $\bar{*}$ and let $U \downarrow \bar{f}$ be used to state that U is closed under the operations $\bar{f}$; then the RA is given by

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\vdash \phi\,[?\alpha_\Upsilon]}{\vdash \phi_{\mathsf{with}}\,[?\alpha_\Upsilon, \bar{*}]}\ (1)}{\vdash \Upsilon_{\mathsf{with}}\ ?\bar{f}\,[?\alpha] \longrightarrow \phi_{\mathsf{with}}\left[?\alpha, ?\bar{f}\right]}\ (2)}{U \neq \varnothing, {}_{\alpha}(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \vdash \Upsilon_{\mathsf{with}}\ ?\bar{f}\,[?\alpha] \longrightarrow \phi_{\mathsf{with}}\left[?\alpha, ?\bar{f}\right]}\ (3)}{U \neq \varnothing, {}_{\alpha}(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \vdash \Upsilon_{\mathsf{with}}\ ?\bar{f}\,[\beta] \longrightarrow \phi_{\mathsf{with}}\left[\beta, ?\bar{f}\right]}\ (4)}{U \neq \varnothing, {}_{\alpha}(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \vdash U \downarrow ?\bar{f}\,[\alpha] \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ ?\bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, ?\bar{f}\right]}\ (5)}{U_{\alpha\ \mathsf{set}} \neq \varnothing \vdash U \downarrow ?\bar{f}\,[\alpha] \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ ?\bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, ?\bar{f}\right]}\ (6)}{\vdash ?U_{?\alpha\ \mathsf{set}} \neq \varnothing \longrightarrow ?U \downarrow ?\bar{f}\,[?\alpha] \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ ?U\ ?\bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[?\alpha, ?U, ?\bar{f}\right]}\ (7)$$

The input to the RA is assumed to be a theorem $\vdash \phi[?\alpha_\Upsilon]$. Step 1 will be referred to as the first step of the dictionary construction (subsection 5.2 in [19]); step 2 as unoverloading of the type $?\alpha_\Upsilon$: it includes class internalization (subsection 5.1 in [19]) and the application of the UO (it corresponds to the application of the attribute *unoverload-type* [14]); step 3 provides the assumptions that are the prerequisites for the application of the LT; step 4 is reserved for the concrete type instantiation; step 5 is the application of *Transfer* (section 6 in [19]); step 6 refers to the application of the LT; step 7 is the export of the theorem from the local context [33].

Implementation of Types-To-Sets

In [19], the authors extended the implementation of Isabelle/HOL with the LT and UO. Also, they introduced the attributes *internalize-sort*, *unoverload* and *cancel-type-definition* that allowed for the execution of steps 1, 3 and 7 (respectively) of the ORA. Other steps could be performed using the technology that already existed. In [14], the implementation was augmented with the attribute *unoverload-type*, which largely subsumed the functionality of the attributes *internalize-sort* and *unoverload*.

The examples of the application of the ORA to theorems in Isabelle/HOL that were developed in [19] already contained an implicit suggestion that the constants and theorems needed for the first step of the dictionary construction in step 2 of the ORA and the *transfer rules* [18] needed for step 6 of the ORA can and should be obtained prior to the application of the algorithm. Thus, using the notation from subsection 1.1.3, for each constant-instance c_σ that occurs in the type-based theorem $\vdash \phi[?\alpha_\Upsilon]$ prior to the application of the ORA with respect to ${}_\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$, the users were expected to provide an *unoverloaded* constant c_{with} such that $c_\sigma = c_{\mathsf{with}}\ \bar{*}$, and a *relativized* constant $c^{\mathsf{on}}_{\mathsf{with}}$ such that $R\big[T_{\alpha\to\beta\to\mathbb{B}}\big]\ (c^{\mathsf{on}}_{\mathsf{with}}\ U_\alpha\ \mathsf{set})\ c_{\mathsf{with}}$ ($\mathbb{B}$ denotes the built-in Isabelle/HOL type *bool* [18]) is a conditional transfer rule (e.g., see [11]), with T being a binary relation that is at least right-total and bi-unique, assuming the default order on predicates in Isabelle/HOL (see [18]).

The unoverloading [17] and relativization of constants for the application of the RA was performed manually in [19, 21, 14]. Nonetheless, unoverloading could be performed using the *classical overloading elimination algorithm* proposed in [17], but it is likely that an implementation of this algorithm was not publicly available at the time of writing of this document. In [12], an alternative algorithm was implemented and made available via the command **unoverload-definition**, although it suffers from several limitations in comparison to the algorithm in [17]. The transfer rules for the constants that are conditionally parametric can be synthesized automatically using the command **parametric-constant** [9] from the standard distribution of Isabelle; the framework *autoref* [22] allows for the synthesis of transfer rules $R\ t\ t'$, including both the *parametricity relation* [18] R and the term t, based on t', under favorable conditions; lastly, in [22] and [14], the authors suggest an outline of another feasible algorithm for the synthesis of the transfer rules based on the functionality of the framework Transfer [11], but do not provide an implementation.

Finally, the assumption ${}_\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$ can be stated using the constant *type-definition* from the standard library of Isabelle/HOL as *type-definition Rep Abs U*; the instantiation of types required in step 4 of the RA can be performed using the standard attributes of Isabelle; step 6 can be performed using the attribute *cancel-type-definition* developed in [19]; step 7 is expected to be performed manually by the user.

1.1.4 Purpose and scope

The extension of the framework Types-To-Sets that is described in this manual adds a further layer of automation to the existing implementation of the framework Types-To-Sets. The primary functionality of the extension is available via the following Isar commands: **tts-context**,

tts-lemmas and **tts-lemma** (and the synonymous commands **tts-corollary**, **tts-proposition** and **tts-theorem**[1]). The commands **tts-lemmas** and **tts-lemma**, when invoked inside an appropriately defined **tts-context** provide the functionality that is approximately equivalent to the application of all steps of the RA and several additional steps of pre-processing of the input and post-processing of the result (collectively referred to as the *extended relativization algorithm* or ERA).

As part of our work on the ETTS, we have also designed the auxiliary framework *Conditional Transfer Rule* (CTR). The framework CTR provides the commands **ud** and **ctr** for the automation of unoverloading of definitions and synthesis of conditional transfer rules from definitions, respectively. Further information about this framework can be found in its reference manual [24]. In this context, we also mention that both the CTR and the ETTS were tested using the framework SpecCheck [16].[2]

The extension was designed under a policy of non-intervention with the existing implementation of the framework Types-To-Sets. Therefore, it does not reduce the scope of the applicability of the framework. However, the functionality that is provided by the commands associated with the extension is a proper subset of the functionality provided by the existing implementation. Nevertheless, the author of the extension believes that there exist very few practical applications of the relativization algorithm that can be solved using the original interface but cannot be solved using the commands that are introduced within the scope of the extension.

[1] In what follows, any reference to the command **tts-lemma** should be viewed as a reference to the entire family of the commands with the identical functionality.

[2] Some of the elements of the content of the tests are based on the elements of the content of [6].

1.2 ETTS and ERA

1.2.1 Background

In this section, we describe our implementation of the prototype software framework ETTS that offers the integration of Types-To-Sets with the Isabelle/Isar infrastructure and full automation of the application of the ERA under favorable conditions. The design of the framework rests largely on our interpretation of several ideas expressed by the authors of [21] and [14].

It has already been mentioned that the primary functionality of the ETTS is available via the Isabelle/Isar [31, 32] commands **tts-context**, **tts-lemmas** and **tts-lemma**. There also exist secondary commands aimed at resolving certain specific problems that one may encounter during relativization: **tts-register-sbts** and **tts-find-sbts**. More specifically, these commands provide means for using transfer rules stated in a local context during the step of the ERA that is similar to step 5 of the RA. The functionality of these commands is explained in more detail in subsection 1.2.3 below.

It is important to note that the description of the ETTS presented in this subsection is only a simplified model of its programmatic implementation in *Isabelle/ML* [25, 33].

1.2.2 Preliminaries

The ERA is an extension of the RA that provides means for the automation of a design pattern similar to the one that was proposed in [14], as well as several additional steps for pre-processing of the input and post-processing of the result of the relativization. In a certain restricted sense the ERA can be seen as a localized form of the RA, as it provides additional infrastructure aimed specifically at making the relativization of theorems stated in the context of Isabelle's *locales* [15, 3, 4] more convenient.

In what follows, assume the existence of an underlying well-formed theory D that contains all definitional axioms that appear in the standard library of Isabelle/HOL. If $\Gamma \vdash {}_{\alpha}(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$ and $\beta, U_{\alpha\ \mathsf{set}}, \mathsf{Rep}_{\beta\to\alpha}, \mathsf{Abs}_{\alpha\to\beta} \in \Gamma$, then the 4-tuple $(U_{\alpha\ \mathsf{set}}, \beta, \mathsf{Rep}_{\beta\to\alpha}, \mathsf{Abs}_{\alpha\to\beta})$, will be referred to as a *relativization isomorphism* (RI) *with respect to* Γ (or RI, if Γ can be inferred). Given the RI $(U_{\alpha\ \mathsf{set}}, \beta, \mathsf{Rep}_{\beta\to\alpha}, \mathsf{Abs}_{\alpha\to\beta})$, the term $U_{\alpha\ \mathsf{set}}$ will be referred to as the *set associated with the RI*, β will be referred to as the *type variable associated with the RI*, $\mathsf{Rep}_{\beta\to\alpha}$ will be referred to as the *representation associated with the RI* and $\mathsf{Abs}_{\alpha\to\beta}$ will be referred to as the *abstraction associated with the RI*. Moreover, any typed term variable $T_{\alpha\to\beta\to\mathbb{B}}$ such that $\Gamma \vdash T = (\lambda x\ y.\ \mathsf{Rep}\ y = x)$ will be referred to as the *transfer relation associated with the RI*. $\Gamma \vdash \mathit{Domainp}\ T = (\lambda x.\ x \in U)$ that holds for this transfer relation will be referred to as the *transfer domain rule associated with the RI*, $\Gamma \vdash \mathit{bi\_unique}\ T$ and $\Gamma \vdash \mathit{right\_total}\ T$ will be referred to as the *side conditions associated with the RI*. For brevity, the abbreviation $\mathsf{dbr}[T_{\alpha\to\beta\to\mathbb{B}}, U_{\alpha\ \mathsf{set}}]$ will be used to mean that $\mathit{Domainp}\ T = (\lambda x.\ x \in U)$, $\mathit{bi\_unique}\ T$ and $\mathit{right\_total}\ T$ for any α, β, $T_{\alpha\to\beta\to\mathbb{B}}$ and $U_{\alpha\ \mathsf{set}}$.

1.2.3 Set-based terms and their registration

Perhaps, one of the most challenging aspects of the automation of the relativization process is related to the application of Transfer during step 5 of the RA: a suitable transfer rule for a given constant-instance may exist only under non-conventional side conditions: an important example that showcases this issue is the built-in constant ε (see [21] and [14] for further information). Unfortunately, the ETTS does not offer a fundamental solution to this problem: the responsibility for providing suitable transfer rules for the application of the ERA remains at the discretion of the user. Nonetheless, the ETTS does provide additional infrastructure that may improve the user experience when dealing with the transfer rules that can only be conveniently stated in an explicitly relativized local context (usually a relativized locale): a common problem

that was already explored in [14].

The authors of [14] choose to perform the relativization of theorems that stem from their specifications in a locale context from within another dedicated relativized locale context. The relativized operations that are represented either by the locale parameters of the relativized locale or remain overloaded constants associated with a given class constraint are lifted to the type variables associated with the RIs that are used for the application of a variant of the relativization algorithm. This variant includes a step during which the variables introduced during unoverloading are substituted (albeit implicitly) for the terms that represent the lifted locale parameters and constants. The additional infrastructure and the additional step are needed, primarily, for the relativization of the constants whose transfer rules can only be stated conveniently in the context of the relativized locale.

A similar approach is used in the ETTS. However, instead of explicitly declaring the lifted constants in advance of the application of the RA, the user is expected to perform the registration of the so-called *set-based term* (sbterm) for each term of interest that is a relativization of a given concept.

The inputs to the algorithm that is associated with the registration of the sbterms are a context Γ, a term $t : \bar{\alpha}\ K$ (K, applied using a postfix notation, contains all information about the type constructors of the type $\bar{\alpha}\ K$) and a sequence of n distinct typed variables $\bar{U}$ with distinct types of the form $\alpha\ \mathsf{set}$, such that $\bar{\alpha}$ is also of length n, all free variables and free type variables that occur in $t : \bar{\alpha}\ K$ also appear free in Γ and $\bar{U}_i : \bar{\alpha}_i\ \mathsf{set}$ for all i, $1 \leq i \leq n$.

Firstly, a term $\exists b.\ R\big[\bar{A}\big]_{\bar{\alpha}\ K \to \bar{\beta}\ K \to \mathbb{B}}\ t\ b$ is formed, such that $R\big[\bar{A}\big]$ is a parametricity relation associated with some type $\bar{\gamma}\ K$ for $\bar{\gamma}$ of length n, such that the sets of the elements of $\bar{\alpha}$, $\bar{\beta}$ and $\bar{\gamma}$ are pairwise disjoint, $\bar{A}$ and $\bar{\beta}$ are both of length n, the elements of $\bar{A}$, $\bar{\beta}$ and $\bar{\gamma}$ are fresh for Γ and $\bar{A}_i : \bar{\alpha}_i \to \bar{\beta}_i \to \mathbb{B}$ for all i such that $1 \leq i \leq n$. Secondly, the context Γ' is built incrementally starting from Γ by adding the formulae $\mathsf{dbr}[\bar{A}_i, \bar{U}_i]$ for each i such that $1 \leq i \leq n$. The term presented above serves as a goal that is meant to be discharged by the user in Γ', resulting in the deduction

$$\Gamma \vdash \mathsf{dbr}[?\bar{A}_i, \bar{U}_i] \longrightarrow \exists b.\ R\big[?\bar{A}\big]_{\bar{\alpha}\ K \to ?\bar{\beta}\ K \to \mathbb{B}}\ t\ b$$

(the index i is distributed over n) after the export to the context Γ. Once the proof is completed, the result is registered in the so-called *sbt-database* allowing a lookup of such results by the sbterm t (the terms and results are allowed to morph when the lookup is performed from within a context different from Γ [7]).

1.2.4 Parameterization of the ERA

Assuming the existence of some context Γ, the ERA is parameterized by the *RI specification*, *the sbterm specification*, the *rewrite rules for the set-based theorem*, the *known premises for the set-based theorem*, the *specification of the elimination of premises in the set-based theorem* and the *attributes for the set-based theorem*. A sequence of the entities in the list above will be referred to as the *ERA-parameterization for* Γ.

The RI Specification is a finite non-empty sequence of pairs of variables $(?\gamma, U_{\alpha\ \mathsf{set}})$, such that $U_{\alpha\ \mathsf{set}} \in \Gamma$. The individual elements of the RI specification will be referred to as the *RI specification elements*. The first element of the RI specification element will be referred to as the *schematic type variable associated with the RI specification element*, the second element will be referred to as the *set associated with the RI specification element*.

The sbterm specification is a finite sequence of pairs $(t : ?\bar{\alpha}\ K,\ u : \bar{\beta}\ K)$, where t is either a constant-instance or a schematic typed term variable and u is an sbterm with respect to Γ. The notation for the elements of the sbterm specification follows a convention similar to the one introduced for the RI specification elements.

The rewrite rules for the set-based theorem can be any set of valid rules for the Isabelle simplifier [34]; the known premises for the set-based theorem can be any finite sequence of deductions in Γ; the specification of the elimination of premises in the set-based theorem is a pair $(\bar{t}, m)$, where $\bar{t}$ is a sequence of formulae and m is a proof method; the attributes for the set-based theorem is a sequence of attributes of Isabelle (e.g., see [34]).

1.2.5 Definition of the ERA

The relativization is performed inside a local context Γ with an associated ERA-parameterization (such a context-parameterization pair will be called a *tts context*). The ERA provides explicit support for handling the transfer rules local to the context through the infrastructure for the registration of sbterms, as explained in subsection 1.2.3. Apart from this, the main part of the ERA largely follows the outline of the RA. However, the ERA also provides several tools for post-processing of the raw result of the relativization. The ERA also has an initialization stage, but this stage is largely hidden from the end-user. Thus, the ERA can be divided in three distinct parts: *initialization of the relativization context*, *kernel of the ERA* (KERA) and *post-processing*.

Assume that the context Γ contains the variable $U_{\alpha\ \mathsf{set}}$ and the finite sequences of variables $\bar{g}$ and $\bar{f}$ indexed by I and J, respectively, such that $\bar{g}_i : \alpha\ \bar{K}_i$ and $\bar{f}_j : \alpha\ \bar{L}_j$ for all $i \in I$ and $j \in J$ for some sequences of functions $\bar{K}$ and $\bar{L}$ also indexed by I and J, respectively, representing the type constructors. Also, assume that the input to the relativization algorithm is the type-based theorem $\vdash \phi\big[?\alpha_\Upsilon, ?\bar{h}\,[?\alpha_\Upsilon]\big]$ such that $?\bar{h}$ is indexed by I and Υ depends on the overloaded constants $\bar{\star}$ indexed by J. Finally, assume that the ERA is parameterized by the RI specification $(?\alpha_\Upsilon, U_{\alpha\ \mathsf{set}})$ and the sbterm specification elements $(?\bar{h}_i, \bar{g}_i)$ and $(\bar{\star}_j, \bar{f}_j)$ for all $i \in I$ and $j \in J$.

Initialization of the relativization context. Prior to the application of the relativization algorithm, the formula $\exists \mathsf{Rep}\ \mathsf{Abs}.\ {}_\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$ is added to the context Γ, with the type variable β being fresh for Γ, resulting in a new context Γ' such that $\Gamma \subseteq \Gamma'$ and $\exists \mathsf{Rep}\ \mathsf{Abs}.\ {}_\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \in \Gamma'$. Then, the properties of the Hilbert choice ε are used for the definition of Rep and Abs such that $\Gamma' \vdash {}_\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$ (e.g., see [18]). In this case, $(U_{\alpha\ \mathsf{set}}, \beta, \mathsf{Rep}_{\beta\to\alpha}, \mathsf{Abs}_{\alpha\to\beta})$ is an RI with respect to Γ'. Furthermore, a fresh $T_{\alpha\to\beta\to\mathbb{B}}$ (for Γ) is defined as a transfer relation associated with the RI. Finally, the transfer domain rule associated with the RI and the side conditions associated with the RI are proved for T with respect to Γ'. For each $\bar{g}_i$ such that $i \in I$, the sbt-database contains a deduction $\Gamma \vdash \mathsf{dbr}[?A, U] \longrightarrow \exists a.\ R\,[?A]_{\alpha\ \bar{K}_i\to?\delta\ \bar{K}_i\to\mathbb{B}}\ \bar{g}_i\ a$. Thence, for each $i \in I$, $?\delta$ is instantiated as β and $?A_{\alpha\to?\delta\to\mathbb{B}}$ is instantiated as $T_{\alpha\to\beta\to\mathbb{B}}$. The resulting theorems are used for the definition of a fresh (for Γ) sequence of variables $\bar{a}$ such that $\Gamma' \vdash R\big[T_{\alpha\to\beta\to\mathbb{B}}\big]_{\alpha\ \bar{K}_i\to\beta\ \bar{K}_i\to\mathbb{B}}\ \bar{g}_i\ \bar{a}_i$. Similar deductions are also established for the sequence $\bar{f}$, with the sequence of the variables appearing on the right-hand side of the transfer rules denoted by $\bar{b}$. These deductions are meant to be used by the transfer infrastructure during the step of the ERA that is equivalent to step 5 of the RA, as shown below. Thus, at the end of the initialization of the relativization context, the theorem is transformed into a deduction of the form $\Gamma' \vdash \phi\big[?\alpha_\Upsilon, ?\bar{h}\,[?\alpha_\Upsilon]\big]$, where the context Γ' (called the *relativization context*) is such that $\Gamma \subseteq \Gamma'$ and it has an associated relativization isomorphism $(U_{\alpha\ \mathsf{set}}, \beta, \mathsf{Rep}_{\beta\to\alpha}, \mathsf{Abs}_{\alpha\to\beta})$ for some fresh β, the associated fresh transfer relation T and fresh variables $\bar{a}$ and $\bar{b}$ indexed by I and J (with freshness being assessed with respect to Γ). Also, the following transfer rules are provided for all $i \in I$ and $j \in J$: $\Gamma' \vdash \bar{R}_i\,[T]\ \bar{g}_i\ \bar{a}_i$ and $\Gamma' \vdash \bar{R'}_j\,[T]\ \bar{f}_j\ \bar{b}_j$ ($\bar{R}$ and $\bar{R'}$ are sequences of parametricity relations indexed by I and J, respectively).

Kernel of ERA. The KERA is similar to the RA:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma' \vdash \phi\left[?\alpha_\Upsilon, ?\bar{h}\left[?\alpha_\Upsilon\right]\right]}{\Gamma' \vdash \phi_{\mathsf{with}}\left[?\alpha_\Upsilon, ?\bar{h}\left[?\alpha_\Upsilon\right], \bar{*}\right]}\ (1)}{\Gamma' \vdash \Upsilon_{\mathsf{with}}\ ?\bar{f}\left[?\alpha\right] \longrightarrow \phi_{\mathsf{with}}\left[?\alpha, ?\bar{h}\left[?\alpha\right], ?\bar{f}\right]}\ (2)}{\Gamma' \vdash \Upsilon_{\mathsf{with}}\ ?\bar{f}\left[\beta\right] \longrightarrow \phi_{\mathsf{with}}\left[\beta, ?\bar{h}\left[\beta\right], ?\bar{f}\right]}\ (3)}{\Gamma' \vdash \Upsilon_{\mathsf{with}}\ \bar{b} \longrightarrow \phi_{\mathsf{with}}\left[\beta, \bar{a}, \bar{b}\right]}\ (4)}{\Gamma' \vdash U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ \bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, \bar{g}, \bar{f}\right]}\ (5)}{\Gamma \vdash \exists \mathsf{Rep\ Abs}._\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}} \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ \bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, \bar{g}, \bar{f}\right]}\ (6)}{\Gamma \vdash U \neq \varnothing \longrightarrow U \downarrow \bar{g}, \bar{f} \longrightarrow \Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ \bar{f} \longrightarrow \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, \bar{g}, \bar{f}\right]}\ (7)$$

Thus, step 1 will be referred to as the first step of the dictionary construction (similar to step 1 of the RA); step 2 will be referred to as unoverloading of the type $?\alpha_\Upsilon$: it includes class internalization and the application of the UO (similar to step 2 of the RA); in step 3, $?\alpha$ is instantiated as β using the RI specification (similar to step 4 in the RA); in step 4, the sbterm specification is used for the instantiation of $?\bar{h}$ as $\bar{a}$ and $?\bar{f}$ as $\bar{b}$; step 5 refers to the application of transfer, including the transfer rules associated with the sbterms (similar to step 5 in the RA); in step 6, the result is exported from Γ' to Γ, providing the additional premise $\exists \mathsf{Rep\ Abs}._\alpha(\beta \approx U)^{\mathsf{Abs}}_{\mathsf{Rep}}$; step 7 is the application of the attribute *cancel-type-definition* (similar to step 6 in the RA).

The RI specification and the sbterm specification provide the information that is necessary to perform the type and term substitutions in steps 3 and 4 of the KERA. If the specifications are viewed as finite maps, their domains morph along the transformations that the theorem undergoes until step 4.

Post-processing. The deduction that is obtained in the final step of the KERA can often be simplified further. The following post-processing steps were created to allow for the presentation of the set-based theorem in a format that is both desirable and convenient for the usual applications:

1. *Simplification.* The rewriting is performed using the rewrite rules for the set-based theorem: the implementation relies on the functionality of the Isabelle's simplifier.
2. *Substitution of known premises.* The known premises for the set-based theorem are matched with the premises of the set-based theorem, allowing for their elimination.
3. *Elimination of premises.* Each premise is matched against each term in the specification of the elimination of premises in the set-based theorem; the associated method is applied in an attempt to eliminate the matching premises (this can be useful for the elimination of the premises of the form $U \neq \varnothing$).
4. *Application of the attributes for the set-based theorem.* The attributes for the set-based theorem are applied as the final step during post-processing.

Generally, the desired form of the result after a successful application of post-processing is similar to $\Gamma \vdash \phi^{\mathsf{on}}_{\mathsf{with}}\left[\alpha, U, \bar{g}, \bar{f}\right]$ with the premises $U \neq \varnothing$, $U \downarrow \bar{g}, \bar{f}$ and $\Upsilon^{\mathsf{on}}_{\mathsf{with}}\ U\ \bar{f}$ eliminated completely (these premises can often be inferred from the context Γ).

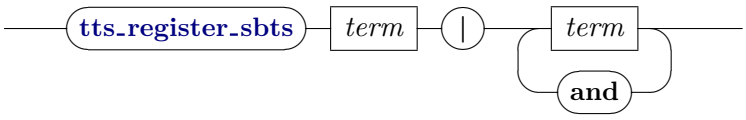
1.3 Syntax

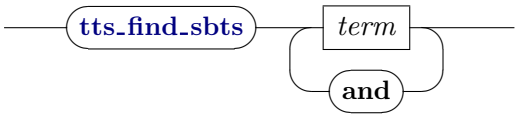
1.3.1 Background

This section presents the syntactic categories that are associated with the commands **tts-context**, **tts-lemmas**, **tts-lemma**, and several other closely related auxiliary commands. It is important to note that the presentation of the syntax is approximate.

1.3.2 Registration of the set-based terms

tts-register-sbts : *local-theory* → *proof*(*prove*)
tts-find-sbts : *context* →





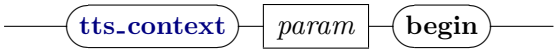
tts-register-sbts $t \mid U_1$ **and** ... **and** U_n allows for the registration of the set-based terms in the sbt-database. Generally, U_i $(1{\leq}i{\leq}n)$ must be distinct fixed variables with distinct types of the form $'a$ *set*, with the set of the type variables that occur in the types of U_i equivalent to the set of the type variables that occur in the type of t.

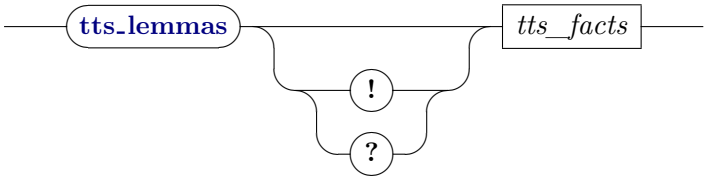
tts-find-sbts t_1 **and** ... **and** t_n prints the templates for the transfer rules for the set-based terms $t_1 \ldots t_n$ for some positive integer n. If no arguments are provided, then the templates for all sbterms in the sbt-database are printed.

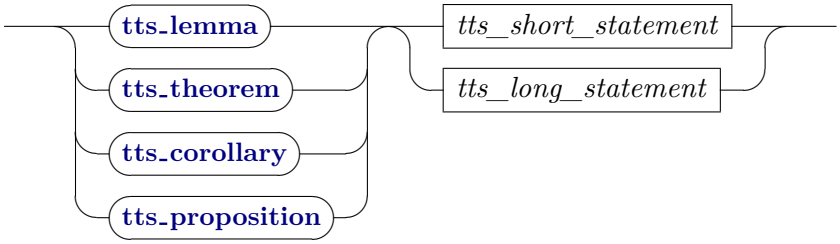
1.3.3 Relativization of theorems

tts-context : *theory* → *local-theory*
tts-lemmas : *local-theory* → *local-theory*
tts-lemma : *local-theory* → *proof*(*prove*)
tts-theorem : *local-theory* → *proof*(*prove*)
tts-corollary : *local-theory* → *proof*(*prove*)
tts-proposition : *local-theory* → *proof*(*prove*)

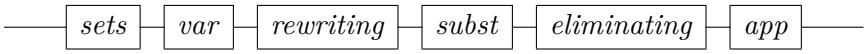
The relativization of theorems should always be performed inside an appropriately parameterized tts context. The tts context can be set up using the command **tts-context**. The framework introduces two types of interfaces for the application of the extended relativization algorithm: **tts-lemmas** and the family of the commands with the identical functionality: **tts-lemma**, **tts-theorem**, **tts-corollary**, **tts-proposition**. Nonetheless, the primary purpose of the command **tts-lemmas** is the experimentation and the automated generation of the relativized results stated using the command **tts-lemma**.



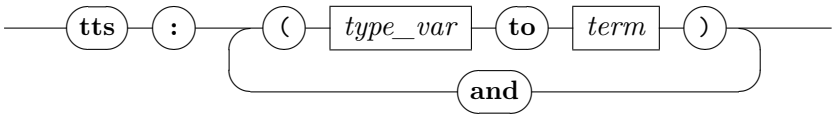




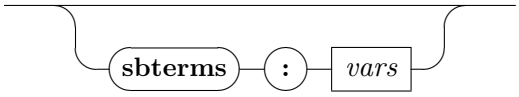
param



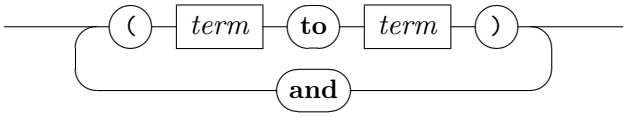
sets



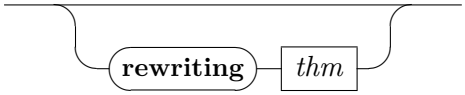
var



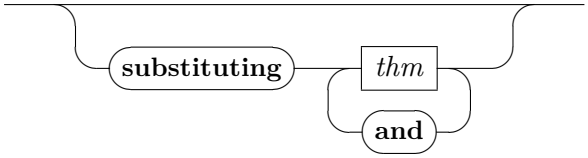
vars



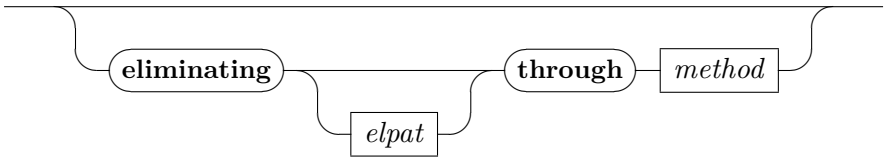
rewriting



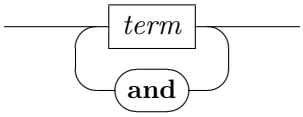
subst



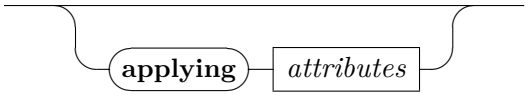
eliminating



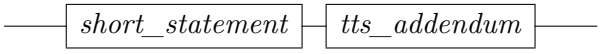
elpat



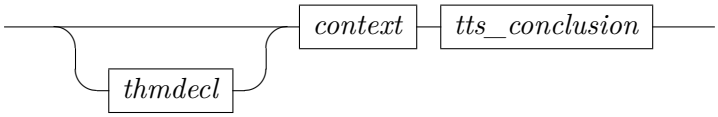
app



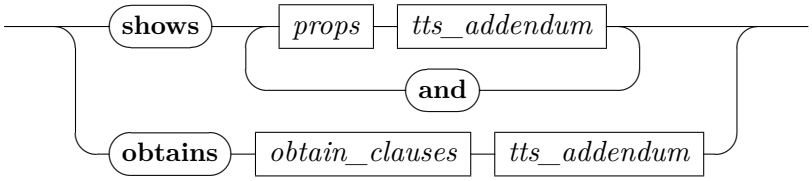
tts_short_statement



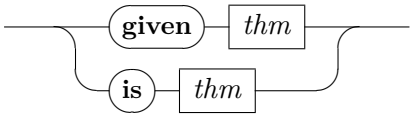
tts_long_statement



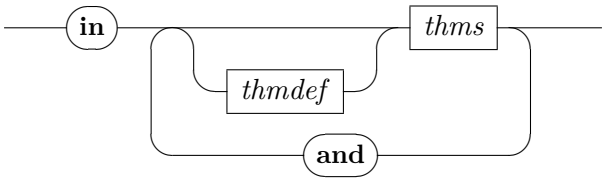
tts_conclusion



tts_addendum



tts_facts



tts-context *param* **begin** provides means for the specification of a new (unnamed) tts context.

tts : (*?*a_1 **to** U_1) **and** ... **and** (*?*a_n **to** U_n) provides means for the declaration of the RI specification. For each i ($1 \leq i \leq n$, n — positive integer), *?*a_i must be a schematic type variable that occurs in each theorem provided as an input to the commands **tts-lemmas** and **tts-lemma** invoked inside the tts context and U_i can be any term of the type $'a$ *set*, where $'a$ is a fixed type variable.

sbterms : (*tbcv*$_1$ **to** *sbt*$_1$) **and** ... **and** (*tbcv*$_n$ **to** *sbt*$_n$) can be used for the declaration of the sbterm specification. For each individual entry i, such that $1 \leq i \leq n$ with n being a non-negative integer, *tbcv*$_i$ has to be either an overloaded operation that occurs in every theorem that is provided as an input to the extended relativization algorithm or a schematic variable that occurs in every theorem that is provided as an input to the command, and *sbt*$_i$ has to be a term registered in the sbt-database.

rewriting *thm* provides means for the declaration of the rewrite rules for the set-based theorem.

substituting thm_1 **and** ... **and** thm_n (n — non-negative integer) provides means for the declaration of the known premises for the set-based theorem.

eliminating $term_1$ **and** ... **and** $term_n$ **through** *method* (n — non-negative integer) provides means for the declaration of the specification of the elimination of premises in the set-based theorem.

applying [$attr_1$, ..., $attr_n$] (n — non-negative integer) provides means for the declaration of the attributes for the set-based theorem.

tts-lemmas applies the ERA to a list of facts and saves the resulting set-based facts in the context. The command **tts-lemmas** should always be invoked from within a tts context. If the statement of the command is followed immediately by the optional keyword !, then it operates in the verbose mode, printing the output of the application of the individual steps of the ERA. If the statement of the command is followed immediately by the optional keyword *?*, then the command operates in the active mode, outputting the set-based facts in the form of the "active areas" that can be embedded in the Isabelle theory file inside the tts context from which the command **tts-lemmas** was invoked. There is a further minor difference between the active mode and the other two modes of operation that is elaborated upon within the description of the keyword **in** below.

in sbf_1 = tbf_1 **and** ... **and** sbf_n = tbf_n is used for the specification of the type-based theorems and the output of the command. For each individual entry i, such that $1 \leq i \leq n$ with n being a positive integer, tbf_i is used for the specification of the input of the extended relativization algorithm and sbf_i is used for the specification of the name binding for the output of the extended relativization algorithm. The specification of the output is optional: if sbf_i is omitted, then a default specification of the output is inferred automatically. tbf_i must be a schematic fact available in the context, whereas sbf_i can be any fresh name binding. Optionally, it is possible to provide attributes for each individual input and output, e.g., sbf_i[*sbf-attrb*] = tbf_i[*tbf-attrb*]. In this case, the list of the attributes *tbf-attrb* is applied to tbf_i during the first part (initialization of the relativization context) of the ERA. If the command operates in the active mode, then the attributes *sbf-attrb* are included in the active area output, but not added to the list of the set-based attributes. For other modes of operation, the attributes *sbf-attrb* are added to the list of the set-based attributes and applied during the third part (post-processing) of the ERA.

tts-lemma *a*: φ *tts-addendum*, enters proof mode with the main goal formed by an application of a tactic that depends on the settings specified in *tts-addendum* to φ. Eventually, this results in some fact $\vdash\varphi$ to be put back into the target context. The command should always be invoked from within a tts context.

A *tts-long-statement* is similar to the standard *long-statement* in that it allows to build up an initial proof context for the subsequent claim incrementally. Similarly, *tts-short-statement* can be viewed as a natural extension of the standard *short-statement*.

tts-addendum is used for the specification of the pre-processing strategy of the goal φ. φ **is** *thm* applies the extended relativization algorithm to *thm*. If the term that is associated with the resulting set-based theorem is α-equivalent to the term associated with the goal φ, then a specialized tactic solves the main goal, leaving only a trivial goal in its place (the trivial goal can be solved using the terminal proof step **.**). φ **given** *thm* also applies the extended relativization algorithm to *thm*, but the resulting set-based theorem is merely added as a premise to the goal φ.

1.4 ETTS by example

1.4.1 Background

In this section, some of the capabilities of the extension of the framework Types-To-Sets are demonstrated by example. The examples that are presented in this section are expected to be sufficient to begin an independent exploration of the extension, but do not cover the entire spectrum of its functionality.

1.4.2 First steps

Problem statement

Consider the task of the relativization of the type-based theorem *topological-space-class.closed-Un* from the standard library of Isabelle/HOL:

$$[\![closed\ S;\ closed\ T]\!] \Longrightarrow closed\ (S \cup T),$$

where $S::'a::topological\text{-}space\ set$ and $T::'a::topological\text{-}space\ set$.

Unoverloading

The constant *closed* that occurs in the theorem is an overloaded constant defined as $closed\ S = open\ (-\ S)$ for any $S::'a::topological\text{-}space\ set$. The constant may be unoverloaded with the help of the command **ud** that is provided as part of the framework CTR:

ud ‹*topological-space.closed*›
ud *closed′* ‹*closed*›

This invocation declares the constant *closed.with* that is defined as

$$closed.with \equiv \lambda open\ S.\ open\ (-\ S)$$

and provides the theorems *closed.with* given by

$$topological\text{-}space.closed \equiv closed.with$$

and *closed′.with* given by

$$closed \equiv closed.with\ open$$

These theorems are automatically added to the dynamic fact *ud-with*.

Conditional transfer rules

Before the relativization can be performed, the transfer rules need to be available for each constant that occurs in the type-based theorem immediately after step 4 of the KERA. All binary relations that are used in the transfer rules must be at least right total and bi-unique (assuming the default order on predicates in Isabelle/HOL). For the theorem *topological-space-class.closed-Un*, there are two such constants: *class.topological-space* and *closed.with*. The transfer rules can be obtained with the help of the command **ctr** from the framework CTR. The process may involve the synthesis of further relativized constants, as described in the reference manual for the framework CTR.

ctr
relativization

synthesis *ctr-simps*
assumes [*transfer-domain-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*)
and [*transfer-rule*]: *right-total A bi-unique A*
trp (*?'a A*)
in *topological-space-ow*: *class.topological-space-def*
and *closed-ow*: *closed.with-def*

Relativization

As mentioned previously, the relativization of theorems can only be performed from within a suitable tts context. In setting up the tts context, the users always need to provide the RI specification elements that are compatible with the theorems that are meant to be relativized in the tts context. The set of the schematic type variables that occur in the theorem *topological-space-class.closed-Un* is {*?'a*}. Thus, there needs to be exactly one RI specification element of the form (*?'a*, *U*::*'a set*):

tts-context
tts: (*?'a* **to** ‹*U*::*'a set*›)
begin

The relativization can be performed by invoking the command **tts-lemmas** in the following manner:

tts-lemmas*?* **in** *closed-Un'* = *topological-space-class.closed-Un*

In this case, the command was invoked in the active mode, providing an active area that can be used to insert the following theorem directly into the theory file:

tts-lemma *closed-Un'*:
assumes *U* ≠ {}
and ∀ *x*∈*S*. *x* ∈ *U*
and ∀ *x*∈*T*. *x* ∈ *U*
and *topological-space-ow U opena*
and *closed-ow U opena S*
and *closed-ow U opena T*
shows *closed-ow U opena* (*S* ∪ *T*)
is *topological-space-class.closed-Un*⟨*proof*⟩

The invocation of the command **tts-lemmas** in the active mode can be removed with no effect on the theorems that were generated using the command.

end

While our goal was achieved, that is, the theorem *closed-Un'* is, indeed, a relativization of the theorem *topological-space-class.closed-Un*, something does not appear right. Is the assumption *U* ≠ {} necessary? Is it possible to simplify ∀ *x*∈*S*. *x* ∈ *U*? Is it necessary to use such a contrived name for the denotation of the open set predicate? Of course, all of these issues can be resolved by restating the theorem in the form that we would like to see and using *closed-Un'* in the proof of this theorem, e.g.

lemma *closed-Un''*:
assumes *S* ⊆ *U*
and *T* ⊆ *U*
and *topological-space-ow U τ*
and *closed-ow U τ S*
and *closed-ow U τ T*
shows *closed-ow U τ* (*S* ∪ *T*)
⟨*proof*⟩

However, having to restate the theorem presents a grave inconvenience. This can be avoided by using a different format of the *tts-addendum*:

tts-context
  tts: (*?'a* **to** ‹*U*::*'a set*›)
begin

tts-lemma *closed-Un'''*:
  assumes $S \subseteq U$
    and $T \subseteq U$
    and *topological-space-ow U τ*
    and *closed-ow U τ S*
    and *closed-ow U τ T*
  shows *closed-ow U τ* $(S \cup T)$
    given *topological-space-class.closed-Un*
⟨*proof*⟩

end

Nevertheless, could there still be some space for improvement? It turns out that instead of having to state the theorem in the desired form manually, often enough, it suffices to provide additional parameters for post-processing of the raw set-based theorem, as demonstrated in the code below:

tts-context
  tts: (*?'a* **to** ‹*U*::*'a set*›)
  rewriting *ctr-simps*
  eliminating ‹*?U*≠{}› **through** (*auto simp*: *topological-space-ow-def*)
  applying[*of - - - τ*]
begin

tts-lemma *closed-Un''''*:
  assumes $S \subseteq U$
    and $T \subseteq U$
    and *topological-space-ow U τ*
    and *closed-ow U τ S*
    and *closed-ow U τ T*
  shows *closed-ow U τ* $(S \cup T)$
    is *topological-space-class.closed-Un*⟨*proof*⟩

end

Finding the most suitable set of parameters for post-processing of the result of the relativization is an iterative process and requires practice before fluency can be achieved.

ETTS Case Studies: Introduction

2.1 Background

2.1.1 Purpose

The remainder of this document presents several examples of the application of the extension of the framework Types-To-Sets and provides the potential users of the extension with a plethora of design patterns to choose from for their own applied relativization needs.

2.1.2 Related work

Since the publication of the framework Types-To-Sets in [19], there has been a growing interest in its use in applied formalization. Some of the examples of the application of the framework include [8], [23] and [14]. However, this list is not exhaustive. Arguably, the most significant application example was developed in [14], where Fabian Immler and Bohua Zhan performed the relativization of over 200 theorems from the standard mathematics library of Isabelle/HOL.

Nonetheless, it is likely that the work presented in this document is the first significant application of the ETTS: unsurprisingly, the content of this document was developed in parallel with the extension of the framework itself. Also, perhaps, it is the largest application of the framework Types-To-Sets at the time of writing: only one of the three libraries (SML Relativization) presented in the context of this work contains the relativization of over 800 theorems from the standard library of Isabelle/HOL.

2.2 Examples: overview

2.2.1 Background

The examples that are presented in this document were developed for the demonstration of the impact of various aspects of the relativization process on the outcome of the relativization. Three libraries of relativized results were developed in the context of this work:

- *SML Relativization*: a relativization of elements of the standard mathematics library of Isabelle/HOL
- *TTS Vector Spaces*: a renovation of the set-based library that was developed in [14] using the ETTS instead of the existing interface for Types-To-Sets
- *TTS Foundations*: a relativization of a miniature type-based library with every constant being parametric under the side conditions compatible with Types-To-Sets

2.2.2 SML Relativization

The standard library that is associated with the object logic Isabelle/HOL and provided as a part of the standard distribution of Isabelle [1] contains a significant number of formalized results from a variety of fields of mathematics. However, the formalization is performed using a type-based approach: for example, the carrier sets associated with the algebraic structures

and the underlying sets of the topological spaces consist of all terms of an arbitrary type. The ETTS was applied to the relativization of a certain number of results from the standard library.

The results that are formalized in the library SML Relativization are taken from an array of topics that include order theory, group theory, ring theory and topology. However, only the results whose relativization could be nearly fully automated using the frameworks UD, CTR and ETTS with almost no additional proof effort are included.

2.2.3 TTS Vector Spaces

The TTS Vector Spaces is a remake of the library of relativized results that was developed in [14] using the ETTS. The theorems that are provided in the library TTS Vector Spaces are nearly identical to the results that are provided in [14].

A detailed description of the original library has already been given in [14] and will not be restated. The definitional frameworks that are used in [14] and the TTS Vector Spaces are similar. While the unoverloading of most of the constants could be performed by using the command **ud**, the command **ctr** could not be used to establish that the unoverloaded constants are parametric under a suitable set of side conditions. Therefore, like in [14], the proofs of the transfer rules were performed manually. However, the advantages of using the ETTS become apparent during the relativization of theorems: the complex infrastructure that was needed for compiling out dependencies on overloaded constants, the manual invocation of the attributes related to the individual steps of the relativization algorithm, the repeated explicit references to the theorem as it undergoes the transformations associated with the individual steps of the relativization algorithm, the explicitly stated names of the set-based theorems were no longer needed. Furthermore, the theorems synthesized by the ETTS in TTS Vector Spaces appear in the formal proof documents in a format that is similar to the canonical format of the Isabelle/Isar declarations associated with the standard commands such as **lemma**.

2.2.4 TTS Foundations

The most challenging aspect of the relativization process, perhaps, is related to the availability of the transfer rules for the constants in the type-based theorems. Nonetheless, even if the transfer rules are available, having to use the relativized constants in the set-based theorems that are different from the original constants that are used in the type-based theorems can be seen as unnatural and inconvenient. Unfortunately, the library SML Relativization suffers from both of the aforementioned problems. The library that was developed in [14] (hence, also the library TTS Vector Spaces) suffers, primarily, from the former problem, but, arguably, due to the methodology that was chosen for the relativization, the library has a more restricted scope of applicability.

The library TTS Foundations provides an example of a miniature type-based library such that all constants associated with the operations on mathematical structures (effectively, this excludes the constants associated with the locale predicates) in the library are parametric under the side conditions compatible with Types-To-Sets. The relativization is performed with respect to all possible type variables; in this case, the type classes are not used in the type-based library. Currently, the library includes the results from the areas of order theory and semigroups. However, it is hoped that it can be seen that the library can be extended to include most of the content that is available in the main library of Isabelle/HOL.

The library TTS Foundations demonstrates that the development of a set-based library can be nearly fully automated using the existing infrastructure associated with the UD, CTR and ETTS, and requires almost no explicit proofs on behalf of the users of these frameworks.

Chapter 3

SML Relativization

3.1 Extension of the theory *Set*

lemma *set-comp-pair*: $\{f\ t\ r\ |t\ r.\ P\ t\ r\} = \{x.\ \exists t\ r.\ P\ t\ r \wedge x = (f\ t\ r)\}$
⟨*proof*⟩

lemma *image-iff'*: $(\forall x{\in}A.\ f\ x \in B) = (f\ `\ A \subseteq B)$ ⟨*proof*⟩

3.2 Extension of the theory *Lifting-Set*

context
includes *lifting-syntax*
begin

lemma *set-pred-eq-transfer*[*transfer-rule*]:
assumes *right-total A*
shows
((*rel-set A* ===> (=)) ===> (*rel-set A* ===> (=)) ===> (=))
(λ*X Y*. ∀ *s*⊆*Collect* (*Domainp A*). *X s* = *Y s*)
((=)::['*b set* ⇒ *bool*, '*b set* ⇒ *bool*] ⇒ *bool*)
⟨*proof*⟩ **lemma** *vimage-fst-transfer-h*:

pred-prod (*Domainp A*) (*Domainp B*) *x* =
(*x* ∈ *Collect* (*Domainp A*) × *Collect* (*Domainp B*))

⟨*proof*⟩

lemma *vimage-fst-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A right-total B*
shows
((*rel-prod A B* ===> *A*) ===> *rel-set A* ===> *rel-set* (*rel-prod A B*))
(λ*f S*. (*f* −‘ *S*) ∩ ((*Collect* (*Domainp A*)) × (*Collect* (*Domainp B*))))
vimage
⟨*proof*⟩

lemma *vimage-snd-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*
shows
((*rel-prod A B* ===> *B*) ===> *rel-set B* ===> *rel-set* (*rel-prod A B*))
(λ*f S*. (*f* −‘ *S*) ∩ ((*Collect* (*Domainp A*)) × (*Collect* (*Domainp B*))))
vimage
⟨*proof*⟩

lemma *vimage-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique B right-total A*
shows
((*A* ===> *B*) ===> (*rel-set B*) ===> *rel-set A*)
(λ*f s*. (*vimage f s*) ∩ (*Collect* (*Domainp A*))) (−‘)
⟨*proof*⟩

lemma *pairwise-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A*
shows ((*A* ===> *A* ===> (=)) ===> *rel-set A* ===> (=)) *pairwise pairwise*
⟨*proof*⟩

lemma *disjnt-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A*
shows (*rel-set A* ===> *rel-set A* ===> (=)) *disjnt disjnt*
⟨*proof*⟩

lemma *bij-betw-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A bi-unique B*
shows ((*A* ===> *B*) ===> *rel-set A* ===> *rel-set B* ===> (=)) *bij-betw bij-betw*
⟨*proof*⟩

end

3.3 Extension of the theory *Product-Type-Ext*

context
includes *lifting-syntax*
begin

lemma *Sigma-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A*
shows
(*rel-set A* ===> (*A* ===> *rel-set B*) ===> *rel-set* (*rel-prod A B*))
Sigma Sigma
⟨*proof*⟩

end

3.4 Extension of the theory *Transfer*

lemma *bi-unique-intersect-r*:
assumes *bi-unique T*
and *rel-set T a a′*
and *rel-set T b b′*
and *rel-set T (a ∩ b) xr*
shows *a′ ∩ b′ = xr*
⟨*proof*⟩

lemma *bi-unique-intersect-l*:
assumes *bi-unique T*
and *rel-set T a a′*
and *rel-set T b b′*
and *rel-set T xl (a′ ∩ b′)*
shows *a ∩ b = xl*
⟨*proof*⟩

lemma *bi-unique-intersect*:
assumes *bi-unique T* **and** *rel-set T a a′* **and** *rel-set T b b′*
shows *rel-set T (a ∩ b) (a′ ∩ b′)*
⟨*proof*⟩

lemma *bi-unique-union-r*:
assumes *bi-unique T*
and *rel-set T a a′*
and *rel-set T b b′*
and *rel-set T (a ∪ b) xr*
shows *a′ ∪ b′ = xr*
⟨*proof*⟩

lemma *bi-unique-union-l*:
assumes *bi-unique T*
and *rel-set T a a′*
and *rel-set T b b′*
and *rel-set T xl (a′ ∪ b′)*
shows *a ∪ b = xl*
⟨*proof*⟩

lemma *bi-unique-union*:
assumes *bi-unique T* **and** *rel-set T a a′* **and** *rel-set T b b′*
shows *rel-set T (a ∪ b) (a′ ∪ b′)*
⟨*proof*⟩

lemma *bi-unique-Union-r*:
fixes *T* :: [′*a*, ′*b*] ⇒ *bool* **and** *K*
defines *K′*: *K′* ≡ {(*x*, *y*). *rel-set T x y*} `` *K*
assumes *bi-unique T*
and ⋃*K* ⊆ *Collect* (*Domainp T*)
and *rel-set T* (⋃*K*) *xr*
shows ⋃*K′* = *xr*
⟨*proof*⟩

lemma *bi-unique-Union-l*:
fixes *T* :: [′*a*, ′*b*] ⇒ *bool* **and** *K′*
defines *K*: *K* ≡ {(*x*, *y*). *rel-set* (λ *y x*. *T x y*) *x y*} `` *K′*
assumes *bi-unique T*

and $\bigcup K' \subseteq$ *Collect* (*Rangep T*)
and *rel-set T xl* ($\bigcup K'$)
shows $\bigcup K$ = *xl*
⟨*proof*⟩

context
includes *lifting-syntax*
begin

The lemma *Domainp-applyI* was adopted from the lemma with the identical name in the theory *Types-To-Sets/Group-on-With.thy*.

lemma *Domainp-applyI*:
includes *lifting-syntax*
shows (*A* ===> *B*) *f g* ⟹ *A x y* ⟹ *Domainp B* (*f x*)
⟨*proof*⟩

lemma *Domainp-fun*:
assumes *left-unique A*
shows
Domainp (*rel-fun A B*) =
(λ*f*. *f* ‘ (*Collect* (*Domainp A*)) ⊆ (*Collect* (*Domainp B*)))
⟨*proof*⟩

lemma *Bex-fun-transfer*[*transfer-rule*]:
assumes *bi-unique A right-total B*
shows
(((*A* ===> *B*) ===> (=)) ===> (=))
(*Bex* (*Collect* (λ*f*. *f* ‘ (*Collect* (*Domainp A*)) ⊆ (*Collect* (*Domainp B*))))))
Ex
⟨*proof*⟩

end

3.5 Relativization of the results about relations

3.5.1 Definitions and common properties

context
notes [[*inductive-internals*]]
begin

inductive-set *trancl-on* :: [$'a$ *set*, ($'a \times 'a$) *set*] ⇒ ($'a \times 'a$) *set*
(‹*on* -/ (-$^+$)› [1000, 1000] 999)
for U :: $'a$ *set* **and** r :: ($'a \times 'a$) *set*
where
r-into-trancl[*intro*, *Pure.intro*]:
⟦ $a \in U$; $b \in U$; $(a, b) \in r$ ⟧ ⟹ $(a, b) \in on\ U\ r^+$
| *trancl-into-trancl*[*Pure.intro*]:

⟦ $a \in U$; $b \in U$; $c \in U$; $(a, b) \in on\ U\ r^+$; $(b, c) \in r$ ⟧ ⟹
$(a, c) \in on\ U\ r^+$

abbreviation *tranclp-on* (‹*on* -/ (-$^{++}$)› [1000, 1000] 1000) **where**
tranclp-on ≡ *trancl-onp*

declare *trancl-on-def*[*nitpick-unfold del*]

lemmas *tranclp-on-def* = *trancl-onp-def*

end

definition *transp-on* :: [$'a$ *set*, [$'a$, $'a$] ⇒ *bool*] ⇒ *bool*
where *transp-on* U = ($\lambda r.\ (\forall x{\in}U.\ \forall y{\in}U.\ \forall z{\in}U.\ r\ x\ y \longrightarrow r\ y\ z \longrightarrow r\ x\ z)$)

definition *acyclic-on* :: [$'a$ *set*, ($'a \times 'a$) *set*] ⇒ *bool*
where *acyclic-on* U = ($\lambda r.\ (\forall x{\in}U.\ (x, x) \notin on\ U\ r^+)$)

lemma *trancl-on-eq-tranclp-on*:
on P ($\lambda x\ y.\ (x, y) \in r)^{++}$ $x\ y$ = ($(x, y) \in on\ (Collect\ P)\ r^+$)
⟨*proof*⟩

lemma *trancl-on-imp-U*: $(x, y) \in on\ U\ r^+ \implies (x, y) \in U \times U$
⟨*proof*⟩

lemmas *tranclp-on-imp-P* = *trancl-on-imp-U*[*to-pred*, *simplified*]

lemma *trancl-on-imp-trancl*: $(x, y) \in on\ U\ r^+ \implies (x, y) \in r^+$
⟨*proof*⟩

lemmas *tranclp-on-imp-tranclp* = *trancl-on-imp-trancl*[*to-pred*]

lemma *tranclp-eq-tranclp-on*: r^{++} = *on* ($\lambda x.\ True$) r^{++}
⟨*proof*⟩

lemma *trancl-eq-trancl-on*: r^+ = *on UNIV* r^+
⟨*proof*⟩

lemma *transp-on-empty*[*simp*]: *transp-on* {} r ⟨*proof*⟩

lemma *transp-eq-transp-on*: *transp* = *transp-on UNIV*

⟨*proof*⟩

lemma *acyclic-on-empty*[*simp*]: *acyclic-on* {} *r* ⟨*proof*⟩

lemma *acyclic-eq-acyclic-on*: *acyclic* = *acyclic-on UNIV*
⟨*proof*⟩

3.5.2 Transfer rules I: *lfp* transfer

The following context contains code from [13].

context
includes *lifting-syntax*
begin

lemma *Inf-transfer*[*transfer-rule*]:
(*rel-set* (*A* ===> (=)) ===> *A* ===> (=)) *Inf Inf*
⟨*proof*⟩

lemma *less-eq-pred-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A*
shows
((*A* ===> (=)) ===> (*A* ===> (=)) ===> (=))
(λ*f g*. ∀ *x*∈*Collect*(*Domainp A*). *f x* ≤ *g x*) (≤)
⟨*proof*⟩

lemma *lfp-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
defines *R* ≡ (((*A* ===> (=)) ===> (*A* ===> (=))) ===> (*A* ===> (=)))
shows *R* (λ*f*. *lfp* (λ*u x*. *if Domainp A x then f u x else bot*)) *lfp*
⟨*proof*⟩

lemma *Inf*2*-transfer*[*transfer-rule*]:
(*rel-set* (*T* ===> *T* ===> (=)) ===> *T* ===> *T* ===> (=)) *Inf Inf*
⟨*proof*⟩

lemma *less-eq*2*-pred-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total T*
shows
((*T* ===> *T* ===> (=)) ===> (*T* ===> *T* ===> (=)) ===> (=))
(λ*f g*. ∀ *x*∈*Collect*(*Domainp T*). ∀ *y*∈*Collect*(*Domainp T*). *f x y* ≤ *g x y*) (≤)
⟨*proof*⟩

lemma *lfp*2*-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
defines
R ≡
(((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=))) ===> (*A* ===> *A* ===> (=)))
shows
R
(
λ*f*. *lfp*
(
λ*u x y*.
if Domainp A x
then if Domainp A y then (*f u*) *x y else bot*
else bot
)

)
lfp
⟨*proof*⟩

end

3.5.3 Transfer rules II: application-specific rules

context
includes *lifting-syntax*
begin

lemma *transp-rt-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (=)) (*transp-on* (*Collect* (*Domainp A*))) *transp*
⟨*proof*⟩

lemma *tranclp-rt-bu-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)))
(*tranclp-on* (*Domainp A*)) *tranclp*
⟨*proof*⟩

lemma *trancl-rt-bu-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(*rel-set* (*rel-prod A A*) ===> *rel-set* (*rel-prod A A*))
(*trancl-on* (*Collect* (*Domainp A*))) *trancl*
⟨*proof*⟩

lemma *acyclic-rt-bu-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*rel-set* (*rel-prod A A*)) ===> (=))
(*acyclic-on* (*Collect* (*Domainp A*))) *acyclic*
⟨*proof*⟩

end

3.6 Relativization of the results about orders

3.6.1 Class *ord*

Definitions and common properties

locale *ord-ow* =
  fixes U :: $'ao$ *set*
    and *le* :: [$'ao$, $'ao$] ⇒ *bool* (**infix** ‹$\leq_{ow}$› 50)
    and *ls* :: [$'ao$, $'ao$] ⇒ *bool* (**infix** ‹$<_{ow}$› 50)
begin

notation *le* (‹$'(\leq_{ow}')$›)
  and *le* (**infix** ‹$\leq_{ow}$› 50)
  and *ls* (‹$'(<_{ow}')$›)
  and *ls* (**infix** ‹$<_{ow}$› 50)

abbreviation (*input*) *ge* (**infix** ‹$\geq_{ow}$› 50) **where** $x \geq_{ow} y \equiv y \leq_{ow} x$
abbreviation (*input*) *gt* (**infix** ‹$>_{ow}$› 50) **where** $x >_{ow} y \equiv y <_{ow} x$

notation *ge* (‹$'(\geq_{ow}')$›)
  and *ge* (**infix** ‹$\geq_{ow}$› 50)
  and *gt* (‹$'(>_{ow}')$›)
  and *gt* (**infix** ‹$>_{ow}$› 50)

tts-register-sbts ‹$(\leq_{ow})$› | U
⟨*proof*⟩

tts-register-sbts ‹$(<_{ow})$› | U
⟨*proof*⟩

end

locale *ord-pair-ow* = ord_1: *ord-ow* U_1 le_1 ls_1 + ord_2: *ord-ow* U_2 le_2 ls_2
  for U_1 :: $'ao$ *set* **and** le_1 ls_1 **and** U_2 :: $'bo$ *set* **and** le_2 ls_2
begin

notation le_1 (‹$'(\leq_{ow.1}')$›)
  and le_1 (**infix** ‹$\leq_{ow.1}$› 50)
  and ls_1 (‹$'(<_{ow.1}')$›)
  and ls_1 (**infix** ‹$<_{ow.1}$› 50)
  and le_2 (‹$'(\leq_{ow.2}')$›)
  and le_2 (**infix** ‹$\leq_{ow.2}$› 50)
  and ls_2 (‹$'(<_{ow.2}')$›)
  and ls_2 (**infix** ‹$<_{ow.2}$› 50)

notation $ord_1.ge$ (‹$'(\geq_{ow.1}')$›)
  and $ord_1.ge$ (**infix** ‹$\geq_{ow.1}$› 50)
  and $ord_1.gt$ (‹$'(>_{ow.1}')$›)
  and $ord_1.gt$ (**infix** ‹$>_{ow.1}$› 50)
  and $ord_2.ge$ (‹$'(\geq_{ow.2}')$›)
  and $ord_2.ge$ (**infix** ‹$\geq_{ow.2}$› 50)
  and $ord_2.gt$ (‹$'(>_{ow.2}')$›)
  and $ord_2.gt$ (**infix** ‹$>_{ow.2}$› 50)

end

ud ‹*ord.lessThan*› (‹(*with* - : ({..<-}))› [1000] 10)
ud *lessThan'* ‹*lessThan*›

ud ‹*ord.atMost*› (‹(*with* - : ({..-}))› [1000] 10)
ud *atMost′* ‹*atMost*›
ud ‹*ord.greaterThan*› (‹(*with* - : ({-<..}))› [1000] 10)
ud *greaterThan′* ‹*greaterThan*›
ud ‹*ord.atLeast*› (‹(*with* - : ({-..}))› [1000] 10)
ud *atLeast′* ‹*atLeast*›
ud ‹*ord.greaterThanLessThan*› (‹(*with* - : ({-<..<-}))› [1000, 999, 1000] 10)
ud *greaterThanLessThan′* ‹*greaterThanLessThan*›
ud ‹*ord.atLeastLessThan*› (‹(*with* - - : ({-..<-}))› [1000, 999, 1000, 1000] 10)
ud *atLeastLessThan′* ‹*atLeastLessThan*›
ud ‹*ord.greaterThanAtMost*› (‹(*with* - - : ({-<..-}))› [1000, 999, 1000, 999] 10)
ud *greaterThanAtMost′* ‹*greaterThanAtMost*›
ud ‹*ord.atLeastAtMost*› (‹(*with* - : ({-..-}))› [1000, 1000, 1000] 10)
ud *atLeastAtMost′* ‹*atLeastAtMost*›
ud ‹*ord.min*› (‹(*with* - : «*min*» - -)› [1000, 1000, 999] 10)
ud *min′* ‹*min*›
ud ‹*ord.max*› (‹(*with* - : «*max*» - -)› [1000, 1000, 999] 10)
ud *max′* ‹*max*›

ctr relativization
  synthesis *ctr-simps*
  assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*)
    and [*transfer-rule*]: *right-total A*
  trp (*?′a A*)
  in *lessThan-ow*: *lessThan.with-def*
    (‹(*on* - *with* - : ({..<-}))› [1000, 1000, 1000] 10)
    and *atMost-ow*: *atMost.with-def*
      (‹(*on* - *with* - : ({..-}))› [1000, 1000, 1000] 10)
    and *greaterThan-ow*: *greaterThan.with-def*
      (‹(*on* - *with* -: ({-<..}))› [1000, 1000, 1000] 10)
    and *atLeast-ow*: *atLeast.with-def*
      (‹(*on* - *with* - : ({-..}))› [1000, 1000, 1000] 10)

ctr relativization
  synthesis *ctr-simps*
  assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*)
    and [*transfer-rule*]: *bi-unique A right-total A*
  trp (*?′a A*)
  in *greaterThanLessThan-ow*: *greaterThanLessThan.with-def*
      (‹(*on* - *with* - : ({-<..<-}))› [1000, 1000, 1000, 1000] 10)
    and *atLeastLessThan-ow*: *atLeastLessThan.with-def*
      (‹(*on* - *with* - - : ({-..<-}))› [1000, 1000, 999, 1000, 1000] 10)
    and *greaterThanAtMost-ow*: *greaterThanAtMost.with-def*
      (‹(*on* - *with* - - : ({-<..-}))› [1000, 1000, 999, 1000, 1000] 10)
    and *atLeastAtMost-ow*: *atLeastAtMost.with-def*
      (‹(*on* - *with* - : ({-..-}))› [1000, 1000, 1000, 1000] 10)

ctr parametricity
  in *min-ow*: *min.with-def*
    and *max-ow*: *max.with-def*

context *ord-ow*
begin

abbreviation *lessThan* :: *′ao* ⇒ *′ao set* (‹(1{..<$_{ow}$-})›)
  where {..<$_{ow}$ *u*} ≡ *on U with* (<$_{ow}$) : {..<*u*}
abbreviation *atMost* :: *′ao* ⇒ *′ao set* (‹(1{..$_{ow}$-})›)
  where {..$_{ow}$ *u*} ≡ *on U with* (≤$_{ow}$) : {..*u*}

abbreviation *greaterThan* :: *'ao* ⇒ *'ao set* (‹(1{-≺$_{ow}$..})›)
  where {*l*≺$_{ow}$..} ≡ *on U with* (≺$_{ow}$) : {*l*<..}
abbreviation *atLeast* :: *'ao* ⇒ *'ao set* (‹(1{-..$_{ow}$})›)
  where *atLeast l* ≡ *on U with* (≤$_{ow}$) : {*l*..}
abbreviation *greaterThanLessThan* :: *'ao* ⇒ *'ao* ⇒ *'ao set* (‹(1{-≺$_{ow}$..≺$_{ow}$-})›)
  where {*l*≺$_{ow}$..≺$_{ow}$*u*} ≡ *on U with* (≺$_{ow}$) : {*l*<..<*u*}
abbreviation *atLeastLessThan* :: *'ao* ⇒ *'ao* ⇒ *'ao set* (‹(1{-..≺$_{ow}$-})›)
  where {*l*..≺$_{ow}$ *u*} ≡ *on U with* (≤$_{ow}$) (≺$_{ow}$) : {*l*<..*u*}
abbreviation *greaterThanAtMost* :: *'ao* ⇒ *'ao* ⇒ *'ao set* (‹(1{-≺$_{ow}$..-})›)
  where {*l*≺$_{ow}$..*u*} ≡ *on U with* (≤$_{ow}$) (≺$_{ow}$) : {*l*<..*u*}
abbreviation *atLeastAtMost* :: *'ao* ⇒ *'ao* ⇒ *'ao set* (‹(1{-..$_{ow}$-})›)
  where {*l*..$_{ow}$*u*} ≡ *on U with* (≤$_{ow}$) : {*l*..*u*}
abbreviation *min* :: *'ao* ⇒ *'ao* ⇒ *'ao* **where** *min* ≡ *min.with* (≤$_{ow}$)
abbreviation *max* :: *'ao* ⇒ *'ao* ⇒ *'ao* **where** *max* ≡ *max.with* (≤$_{ow}$)

end

context *ord-pair-ow*
begin

notation *ord*$_1$.*lessThan* (‹(1{..≺$_{ow.1}$-})›)
notation *ord*$_1$.*atMost* (‹(1{..$_{ow.1}$-})›)
notation *ord*$_1$.*greaterThan* (‹(1{-≺$_{ow.1}$..})›)
notation *ord*$_1$.*atLeast* (‹(1{-..$_{ow.1}$})›)
notation *ord*$_1$.*greaterThanLessThan* (‹(1{-≺$_{ow.1}$..≺$_{ow.1}$-})›)
notation *ord*$_1$.*atLeastLessThan* (‹(1{-..≺$_{ow.1}$-})›)
notation *ord*$_1$.*greaterThanAtMost* (‹(1{-≺$_{ow.1}$..-})›)
notation *ord*$_1$.*atLeastAtMost* (‹(1{-..$_{ow.1}$-})›)

notation *ord*$_2$.*lessThan* (‹(1{..≺$_{ow.2}$-})›)
notation *ord*$_2$.*atMost* (‹(1{..$_{ow.2}$-})›)
notation *ord*$_2$.*greaterThan* (‹(1{-≺$_{ow.2}$..})›)
notation *ord*$_2$.*atLeast* (‹(1{-..$_{ow.2}$})›)
notation *ord*$_2$.*greaterThanLessThan* (‹(1{-≺$_{ow.2}$..≺$_{ow.2}$-})›)
notation *ord*$_2$.*atLeastLessThan* (‹(1{-..≺$_{ow.2}$-})›)
notation *ord*$_2$.*greaterThanAtMost* (‹(1{-≺$_{ow.2}$..-})›)
notation *ord*$_2$.*atLeastAtMost* (‹(1{-..$_{ow.2}$-})›)

end

3.6.2 Preorders

Definitions and common properties

locale *partial-preordering-ow* =
  fixes *U* :: *'ao set*
    and *le* :: *'ao* ⇒ *'ao* ⇒ *bool* (**infix** ‹≤$_{ow}$› 50)
  assumes *refl*: *a* ∈ *U* ⟹ *a* ≤$_{ow}$ *a*
    and *trans*: ⟦ *a* ∈ *U*; *b* ∈ *U*; *c* ∈ *U*; *a* ≤$_{ow}$ *b*; *b* ≤$_{ow}$ *c* ⟧ ⟹ *a* ≤$_{ow}$ *c*
begin

notation *le* (**infix** ‹≤$_{ow}$› 50)

end

locale *preordering-ow* = *partial-preordering-ow U le*
  for *U* :: *'ao set*
    and *le* :: *'ao* ⇒ *'ao* ⇒ *bool* (**infix** ‹≤$_{ow}$› 50) +

fixes $ls :: \langle 'ao \Rightarrow 'ao \Rightarrow bool\rangle$ (**infix** $\langle <_{ow}\rangle$ 50)
assumes *strict-iff-not*:
$[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a <_{ow} b \longleftrightarrow a \leq_{ow} b \wedge \neg\ b \leq_{ow} a$

locale *preorder-ow* = *ord-ow U le ls*
for $U :: 'ao\ set$ **and** *le ls* +
assumes *less-le-not-le*:
$[\![\ x \in U;\ y \in U\ ]\!] \Longrightarrow x <_{ow} y \longleftrightarrow x \leq_{ow} y \wedge \neg\ (y \leq_{ow} x)$
and *order-refl*[*iff*]: $x \in U \Longrightarrow x \leq_{ow} x$
and *order-trans*: $[\![\ x \in U;\ y \in U;\ z \in U;\ x \leq_{ow} y;\ y \leq_{ow} z\ ]\!] \Longrightarrow x \leq_{ow} z$
begin

sublocale
order: *preordering-ow U* $\langle(\leq_{ow})\rangle$ $\langle(<_{ow})\rangle$ +
dual-order: *preordering-ow U* $\langle(\geq_{ow})\rangle$ $\langle(>_{ow})\rangle$
⟨*proof*⟩

end

locale *ord-preorder-ow* =
ord_1: *ord-ow* U_1 le_1 ls_1 + ord_2: *preorder-ow* U_2 le_2 ls_2
for $U_1 :: 'ao\ set$ **and** le_1 ls_1 **and** $U_2 :: 'bo\ set$ **and** le_2 ls_2
begin

sublocale *ord-pair-ow* ⟨*proof*⟩

end

locale *preorder-pair-ow* =
ord_1: *preorder-ow* U_1 le_1 ls_1 + ord_2: *preorder-ow* U_2 le_2 ls_2
for $U_1 :: 'ao\ set$ **and** le_1 **and** ls_1 **and** $U_2 :: 'bo\ set$ **and** le_2 **and** ls_2
begin

sublocale *ord-preorder-ow* ⟨*proof*⟩

end

ud ‹*preordering-bdd.bdd*›
ud ‹*preorder.bdd-above*›
ud *bdd-above'* ‹*bdd-above*›
ud ‹*preorder.bdd-below*›
ud *bdd-below'* ‹*bdd-below*›

ctr relativization
synthesis *ctr-simps*
assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp A* = $(\lambda x.\ x \in U)$
and [*transfer-rule*]: *right-total A*
trp (*?'a A*)
in *bdd-ow*: *bdd.with-def*
(‹(*on - with -* : «*bdd*» -)› [1000, 1000, 1000] 10)
and *bdd-above-ow*: *bdd-above.with-def*
(‹(*on - with -* : «*bdd'-above*» -)› [1000, 1000, 1000] 10)
and *bdd-below-ow*: *bdd-below.with-def*
(‹(*on - with -* : «*bdd'-below*» -)› [1000, 1000, 1000] 10)

declare *bdd.with*[*ud-with del*]

context *preorder-ow*

begin

abbreviation *bdd-above* :: *'ao set* ⇒ *bool*
  where *bdd-above* ≡ *bdd-above-ow U* ($\leq_{ow}$)
abbreviation *bdd-below* :: *'ao set* ⇒ *bool*
  where *bdd-below* ≡ *bdd-below-ow U* ($\leq_{ow}$)

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *partial-preordering-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *right-total A*
  shows
    ((*A* ===> *A* ===> (=)) ===> (=))
      (*partial-preordering-ow* (*Collect* (*Domainp A*))) *partial-preordering*
  ⟨*proof*⟩

lemma *preordering-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *right-total A*
  shows
    ((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
      (*preordering-ow* (*Collect* (*Domainp A*))) *preordering*
  ⟨*proof*⟩

lemma *preorder-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *right-total A*
  shows
    ((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
      (*preorder-ow* (*Collect* (*Domainp A*))) *class.preorder*
  ⟨*proof*⟩

end

Relativization

context *preordering-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  rewriting *ctr-simps*
  substituting *preordering-ow-axioms*
  eliminating through *auto*
begin

tts-lemma *strict-implies-order*:
  assumes *a* ∈ *U* **and** *b* ∈ *U* **and** *a* $<_{ow}$ *b*
  shows *a* $\leq_{ow}$ *b*
    is *preordering.strict-implies-order*⟨*proof*⟩

tts-lemma *irrefl*:
  assumes *a* ∈ *U*
  shows ¬*a* $<_{ow}$ *a*

is *preordering.irrefl*⟨*proof*⟩

tts-lemma *asym*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$ **and** $b <_{ow} a$
shows *False*
is *preordering.asym*⟨*proof*⟩

tts-lemma *strict-trans*1:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a \leq_{ow} b$ **and** $b <_{ow} c$
shows $a <_{ow} c$
is *preordering.strict-trans*1⟨*proof*⟩

tts-lemma *strict-trans*2:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a <_{ow} b$ **and** $b \leq_{ow} c$
shows $a <_{ow} c$
is *preordering.strict-trans*2⟨*proof*⟩

tts-lemma *strict-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a <_{ow} b$ **and** $b <_{ow} c$
shows $a <_{ow} c$
is *preordering.strict-trans*⟨*proof*⟩

end

end

context *preorder-ow*
begin

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *preorder-ow-axioms*
eliminating through *auto*
begin

tts-lemma *less-irrefl*:
assumes $x \in U$
shows $\neg\ x <_{ow} x$
is *preorder-class.less-irrefl*⟨*proof*⟩

tts-lemma *bdd-below-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a<_{ow}..b\}$
is *preorder-class.bdd-below-Ioc*⟨*proof*⟩

tts-lemma *bdd-above-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a<_{ow}..b\}$
is *preorder-class.bdd-above-Ioc*⟨*proof*⟩

tts-lemma *bdd-above-Iic*:
assumes $b \in U$
shows *bdd-above* $\{..<_{ow}b\}$
is *preorder-class.bdd-above-Iic*⟨*proof*⟩

tts-lemma *bdd-above-Iio*:
assumes $b \in U$

shows *bdd-above* $\{..<_{ow} b\}$
is *preorder-class.bdd-above-Iio*⟨*proof*⟩

tts-lemma *bdd-below-Ici*:
assumes $a \in U$
shows *bdd-below* $\{a..._{ow}\}$
is *preorder-class.bdd-below-Ici*⟨*proof*⟩

tts-lemma *bdd-below-Ioi*:
assumes $a \in U$
shows *bdd-below* $\{a<_{ow}..\}$
is *preorder-class.bdd-below-Ioi*⟨*proof*⟩

tts-lemma *bdd-above-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a..._{ow} b\}$
is *preorder-class.bdd-above-Icc*⟨*proof*⟩

tts-lemma *bdd-above-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a<_{ow}..<_{ow} b\}$
is *preorder-class.bdd-above-Ioo*⟨*proof*⟩

tts-lemma *bdd-below-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a..._{ow} b\}$
is *preorder-class.bdd-below-Icc*⟨*proof*⟩

tts-lemma *bdd-below-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a<_{ow}..<_{ow} b\}$
is *preorder-class.bdd-below-Ioo*⟨*proof*⟩

tts-lemma *bdd-above-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* (*on U with* $(\leq_{ow})$ $(<_{ow})$: $\{a..<b\}$)
is *preorder-class.bdd-above-Ico*⟨*proof*⟩

tts-lemma *bdd-below-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* (*on U with* $(\leq_{ow})$ $(<_{ow})$: $\{a..<b\}$)
is *preorder-class.bdd-below-Ico*⟨*proof*⟩

tts-lemma *Ioi-le-Ico*:
assumes $a \in U$
shows $\{a<_{ow}..\} \subseteq \{a..._{ow}\}$
is *preorder-class.Ioi-le-Ico*⟨*proof*⟩

tts-lemma *eq-refl*:
assumes $y \in U$ **and** $x = y$
shows $x \leq_{ow} y$
is *preorder-class.eq-refl*⟨*proof*⟩

tts-lemma *less-imp-le*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $x \leq_{ow} y$
is *preorder-class.less-imp-le*⟨*proof*⟩

tts-lemma *less-not-sym*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\neg\ y <_{ow} x$
is *preorder-class.less-not-sym*⟨*proof*⟩

tts-lemma *less-imp-not-less*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $(\neg\ y <_{ow} x) = \mathit{True}$
is *preorder-class.less-imp-not-less*⟨*proof*⟩

tts-lemma *less-asym'*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$ **and** $b <_{ow} a$
shows P
is *preorder-class.less-asym'*⟨*proof*⟩

tts-lemma *less-imp-triv*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $(y <_{ow} x \longrightarrow P) = \mathit{True}$
is *preorder-class.less-imp-triv*⟨*proof*⟩

tts-lemma *less-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_{ow} y$ **and** $y <_{ow} z$
shows $x <_{ow} z$
is *preorder-class.less-trans*⟨*proof*⟩

tts-lemma *less-le-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_{ow} y$ **and** $y \leq_{ow} z$
shows $x <_{ow} z$
is *preorder-class.less-le-trans*⟨*proof*⟩

tts-lemma *le-less-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x \leq_{ow} y$ **and** $y <_{ow} z$
shows $x <_{ow} z$
is *preorder-class.le-less-trans*⟨*proof*⟩

tts-lemma *bdd-aboveI*:
assumes $A \subseteq U$ **and** $M \in U$ **and** $\bigwedge x.\ [\![x \in U;\ x \in A]\!] \Longrightarrow x \leq_{ow} M$
shows *bdd-above A*
is *preorder-class.bdd-aboveI*⟨*proof*⟩

tts-lemma *bdd-belowI*:
assumes $A \subseteq U$ **and** $m \in U$ **and** $\bigwedge x.\ [\![x \in U;\ x \in A]\!] \Longrightarrow m \leq_{ow} x$
shows *bdd-below A*
is *preorder-class.bdd-belowI*⟨*proof*⟩

tts-lemma *less-asym*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$ **and** $\neg\ P \Longrightarrow y <_{ow} x$
shows P
is *preorder-class.less-asym*⟨*proof*⟩

tts-lemma *bdd-above-Int*1:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-above A*
shows *bdd-above* $(A \cap B)$
is *preorder-class.bdd-above-Int*1⟨*proof*⟩

tts-lemma *bdd-above-Int*2:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-above B*
shows *bdd-above* $(A \cap B)$

is *preorder-class.bdd-above-Int2*⟨*proof*⟩

tts-lemma *bdd-below-Int1*:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-below A*
shows *bdd-below* $(A \cap B)$
is *preorder-class.bdd-below-Int1*⟨*proof*⟩

tts-lemma *bdd-below-Int2*:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-below B*
shows *bdd-below* $(A \cap B)$
is *preorder-class.bdd-below-Int2*⟨*proof*⟩

tts-lemma *bdd-above-mono*:
assumes $B \subseteq U$ **and** *bdd-above B* **and** $A \subseteq B$
shows *bdd-above A*
is *preorder-class.bdd-above-mono*⟨*proof*⟩

tts-lemma *bdd-below-mono*:
assumes $B \subseteq U$ **and** *bdd-below B* **and** $A \subseteq B$
shows *bdd-below A*
is *preorder-class.bdd-below-mono*⟨*proof*⟩

tts-lemma *atLeastAtMost-subseteq-atLeastLessThan-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a..{}_{ow}b\} \subseteq (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{c..<d\})) =$
$(a \leq_{ow} b \longrightarrow b <_{ow} d \wedge c \leq_{ow} a)$
is *preorder-class.atLeastAtMost-subseteq-atLeastLessThan-iff*⟨*proof*⟩

tts-lemma *atMost-subset-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(\{..{}_{ow}x\} \subseteq \{..{}_{ow}y\}) = (x \leq_{ow} y)$
is *Set-Interval.atMost-subset-iff*⟨*proof*⟩

tts-lemma *single-Diff-lessThan*:
assumes $k \in U$
shows $\{k\} - \{..<_{ow}k\} = \{k\}$
is *Set-Interval.single-Diff-lessThan*⟨*proof*⟩

tts-lemma *atLeast-subset-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(\{x..{}_{ow}\} \subseteq \{y..{}_{ow}\}) = (y \leq_{ow} x)$
is *Set-Interval.atLeast-subset-iff*⟨*proof*⟩

tts-lemma *atLeastatMost-psubset-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows
$(\{a..{}_{ow}b\} \subset \{c..{}_{ow}d\}) =$
$(c \leq_{ow} d \wedge (\neg\ a \leq_{ow} b \vee c \leq_{ow} a \wedge b \leq_{ow} d \wedge (c <_{ow} a \vee b <_{ow} d)))$
is *preorder-class.atLeastatMost-psubset-iff*⟨*proof*⟩

tts-lemma *not-empty-eq-Iic-eq-empty*:
assumes $h \in U$
shows $\{\} \neq \{..{}_{ow}h\}$
is *preorder-class.not-empty-eq-Iic-eq-empty*⟨*proof*⟩

tts-lemma *atLeastatMost-subset-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a..{}_{ow}b\} \subseteq \{c..{}_{ow}d\}) = (\neg\ a \leq_{ow} b \vee b \leq_{ow} d \wedge c \leq_{ow} a)$

is *preorder-class.atLeastatMost-subset-iff⟨proof⟩*

tts-lemma *Icc-subset-Ici-iff*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$
shows $(\{l..{}_{ow}h\} \subseteq \{l'..{}_{ow}\}) = (\neg\ l \leq_{ow} h \vee l' \leq_{ow} l)$
is *preorder-class.Icc-subset-Ici-iff⟨proof⟩*

tts-lemma *Icc-subset-Iic-iff*:
assumes $l \in U$ **and** $h \in U$ **and** $h' \in U$
shows $(\{l..{}_{ow}h\} \subseteq \{..{}_{ow}h'\}) = (\neg\ l \leq_{ow} h \vee h \leq_{ow} h')$
is *preorder-class.Icc-subset-Iic-iff⟨proof⟩*

tts-lemma *not-empty-eq-Ici-eq-empty*:
assumes $l \in U$
shows $\{\} \neq \{l..{}_{ow}\}$
is *preorder-class.not-empty-eq-Ici-eq-empty⟨proof⟩*

tts-lemma *not-Ici-eq-empty*:
assumes $l \in U$
shows $\{l..{}_{ow}\} \neq \{\}$
is *preorder-class.not-Ici-eq-empty⟨proof⟩*

tts-lemma *not-Iic-eq-empty*:
assumes $h \in U$
shows $\{..{}_{ow}h\} \neq \{\}$
is *preorder-class.not-Iic-eq-empty⟨proof⟩*

tts-lemma *atLeastatMost-empty-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(\{\} = \{a..{}_{ow}b\}) = (\neg\ a \leq_{ow} b)$
is *preorder-class.atLeastatMost-empty-iff2⟨proof⟩*

tts-lemma *atLeastLessThan-empty-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(\{\} = (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\})) = (\neg\ a <_{ow} b)$
is *preorder-class.atLeastLessThan-empty-iff2⟨proof⟩*

tts-lemma *greaterThanAtMost-empty-iff2*:
assumes $k \in U$ **and** $l \in U$
shows $(\{\} = \{k<{}_{ow}..l\}) = (\neg\ k <_{ow} l)$
is *preorder-class.greaterThanAtMost-empty-iff2⟨proof⟩*

tts-lemma *atLeastatMost-empty-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(\{a..{}_{ow}b\} = \{\}) = (\neg\ a \leq_{ow} b)$
is *preorder-class.atLeastatMost-empty-iff⟨proof⟩*

tts-lemma *atLeastLessThan-empty-iff*:
assumes $a \in U$ **and** $b \in U$
shows $((on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) = \{\}) = (\neg\ a <_{ow} b)$
is *preorder-class.atLeastLessThan-empty-iff⟨proof⟩*

tts-lemma *greaterThanAtMost-empty-iff*:
assumes $k \in U$ **and** $l \in U$
shows $(\{k<{}_{ow}..l\} = \{\}) = (\neg\ k <_{ow} l)$
is *preorder-class.greaterThanAtMost-empty-iff⟨proof⟩*

end

tts-context
tts: (*?'a* **to** *U*)
substituting *preorder-ow-axioms*
begin

tts-lemma *bdd-above-empty*:
assumes $U \neq \{\}$
shows *bdd-above* $\{\}$
is *preorder-class.bdd-above-empty*⟨*proof*⟩

tts-lemma *bdd-below-empty*:
assumes $U \neq \{\}$
shows *bdd-below* $\{\}$
is *preorder-class.bdd-below-empty*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹U_2::*'a set*›)
rewriting *ctr-simps*
substituting *preorder-ow-axioms*
eliminating through (*auto intro*: *bdd-above-empty bdd-below-empty*)
begin

tts-lemma *bdd-belowI2*:
assumes $A \subseteq U_2$
and $m \in U$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $\bigwedge x.\ x \in A \Longrightarrow m \leq_{ow} f\ x$
shows *bdd-below* $(f\ `\ A)$
given *preorder-class.bdd-belowI2*
⟨*proof*⟩

tts-lemma *bdd-aboveI2*:
assumes $A \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $M \in U$
and $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq_{ow} M$
shows *bdd-above* $(f\ `\ A)$
given *preorder-class.bdd-aboveI2*
⟨*proof*⟩

end

end

3.6.3 Partial orders

Definitions and common properties

locale *ordering-ow = partial-preordering-ow U le*
for U :: *'ao set* **and** *le* :: *'ao* ⇒ *'ao* ⇒ *bool* (**infix** ‹$\leq_{ow}$› 50) +
fixes *ls* :: *'ao* ⇒ *'ao* ⇒ *bool* (**infix** ‹$<_{ow}$› 50)
assumes *strict-iff-order*: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a <_{ow} b \longleftrightarrow a \leq_{ow} b \wedge a \neq b$
and *antisym*: $[\![\ a \in U;\ b \in U;\ a \leq_{ow} b;\ b \leq_{ow} a\ ]\!] \Longrightarrow a = b$
begin

notation *le* (**infix** ‹$\leq_{ow}$› 50)
and *ls* (**infix** ‹$<_{ow}$› 50)

sublocale *preordering-ow U* ‹$(\leq_{ow})$› ‹$(<_{ow})$›
⟨*proof*⟩

end

locale *order-ow* = *preorder-ow U le ls* **for** *U* :: *'ao set* **and** *le ls* +
assumes *antisym*: ⟦ $x \in U$; $y \in U$; $x \leq_{ow} y$; $y \leq_{ow} x$ ⟧ ⟹ $x = y$
begin

sublocale
order: *ordering-ow U* ‹$(\leq_{ow})$› ‹$(<_{ow})$› +
dual-order: *ordering-ow U* ‹$(\geq_{ow})$› ‹$(>_{ow})$›
⟨*proof*⟩

no-notation *le* (**infix** ‹$\leq_{ow}$› 50)
and *ls* (**infix** ‹$<_{ow}$› 50)

end

locale *ord-order-ow* = ord_1: *ord-ow* U_1 le_1 ls_1 + ord_2: *order-ow* U_2 le_2 ls_2
for U_1 :: *'ao set* **and** le_1 ls_1 **and** U_2 :: *'bo set* **and** le_2 ls_2

sublocale *ord-order-ow* ⊆ *ord-preorder-ow* ⟨*proof*⟩

locale *preorder-order-ow* =
ord_1: *preorder-ow* U_1 le_1 ls_1 + ord_2: *order-ow* U_2 le_2 ls_2
for U_1 :: *'ao set* **and** le_1 ls_1 **and** U_2 :: *'bo set* **and** le_2 ls_2

sublocale *preorder-order-ow* ⊆ *preorder-pair-ow* ⟨*proof*⟩

locale *order-pair-ow* = ord_1: *order-ow* U_1 le_1 ls_1 + ord_2: *order-ow* U_2 le_2 ls_2
for U_1 :: *'ao set* **and** le_1 ls_1 **and** U_2 :: *'bo set* **and** le_2 ls_2

sublocale *order-pair-ow* ⊆ *preorder-order-ow* ⟨*proof*⟩

ud ‹*monoseq*› (‹(*with* - : «*monoseq*» -)› [1000, 1000] 10)

ctr relativization
synthesis *ctr-simps*
assumes [*transfer-domain-rule*, *transfer-rule*]:
Domainp (*B*::*'c*⇒*'d*⇒*bool*) = (λ*x*. $x \in U_2$)
and [*transfer-rule*]: *right-total B*
trp (*?'b* ‹*A*::*'a*⇒*'b*⇒*bool*›) **and** (*?'a B*)
in *monoseq-ow*: *monoseq.with-def*

Transfer rules

context
includes *lifting-syntax*
begin

lemma *ordering-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))

(*ordering-ow* (*Collect* (*Domainp A*))) *ordering*
⟨*proof*⟩

lemma *order-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
(*order-ow* (*Collect* (*Domainp A*))) *class.order*
⟨*proof*⟩

end

Relativization

context *ordering-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ordering-ow-axioms*
eliminating through *simp*
begin

tts-lemma *strict-implies-not-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows $a \neq b$
is *ordering.strict-implies-not-eq*⟨*proof*⟩

tts-lemma *order-iff-strict*:
assumes $a \in U$ **and** $b \in U$
shows $(a \leq_{ow} b) = (a <_{ow} b \vee a = b)$
is *ordering.order-iff-strict*⟨*proof*⟩

tts-lemma *not-eq-order-implies-strict*:
assumes $a \in U$ **and** $b \in U$ **and** $a \neq b$ **and** $a \leq_{ow} b$
shows $a <_{ow} b$
is *ordering.not-eq-order-implies-strict*⟨*proof*⟩

tts-lemma *eq-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(a = b) = (a \leq_{ow} b \wedge b \leq_{ow} a)$
is *ordering.eq-iff*⟨*proof*⟩

end

end

context *order-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *atLeastAtMost-singleton*:
assumes $a \in U$
shows $\{a..{}_{ow}a\} = \{a\}$
is *order-class.atLeastAtMost-singleton*⟨*proof*⟩

tts-lemma *less-imp-neq*:
assumes $y \in U$ **and** $x <_{ow} y$
shows $x \neq y$
is *order-class.less-imp-neq*⟨*proof*⟩

tts-lemma *atLeastatMost-empty*:
assumes $b \in U$ **and** $a \in U$ **and** $b <_{ow} a$
shows $\{a..{}_{ow}b\} = \{\}$
is *order-class.atLeastatMost-empty*⟨*proof*⟩

tts-lemma *less-imp-not-eq*:
assumes $y \in U$ **and** $x <_{ow} y$
shows $(x = y) = False$
is *order-class.less-imp-not-eq*⟨*proof*⟩

tts-lemma *less-imp-not-eq*2:
assumes $y \in U$ **and** $x <_{ow} y$
shows $(y = x) = False$
is *order-class.less-imp-not-eq*2⟨*proof*⟩

tts-lemma *atLeastLessThan-empty*:
assumes $b \in U$ **and** $a \in U$ **and** $b \leq_{ow} a$
shows $(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) = \{\}$
is *order-class.atLeastLessThan-empty*⟨*proof*⟩

tts-lemma *greaterThanAtMost-empty*:
assumes $l \in U$ **and** $k \in U$ **and** $l \leq_{ow} k$
shows $\{k<_{ow}..l\} = \{\}$
is *order-class.greaterThanAtMost-empty*⟨*proof*⟩

tts-lemma *antisym-conv*1:
assumes $x \in U$ **and** $y \in U$ **and** $\neg\ x <_{ow} y$
shows $(x \leq_{ow} y) = (x = y)$
is *order-class.antisym-conv*1⟨*proof*⟩

tts-lemma *antisym-conv*2:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $(\neg\ x <_{ow} y) = (x = y)$
is *order-class.antisym-conv*2⟨*proof*⟩

tts-lemma *greaterThanLessThan-empty*:
assumes $l \in U$ **and** $k \in U$ **and** $l \leq_{ow} k$
shows $\{k<_{ow}..<_{ow}l\} = \{\}$
is *order-class.greaterThanLessThan-empty*⟨*proof*⟩

tts-lemma *atLeastLessThan-eq-atLeastAtMost-diff*:
assumes $a \in U$ **and** $b \in U$
shows $(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) = \{a..{}_{ow}b\} - \{b\}$
is *order-class.atLeastLessThan-eq-atLeastAtMost-diff*⟨*proof*⟩

tts-lemma *greaterThanAtMost-eq-atLeastAtMost-diff*:
assumes $a \in U$ **and** $b \in U$
shows $\{a<_{ow}..b\} = \{a..{}_{ow}b\} - \{a\}$

is *order-class.greaterThanAtMost-eq-atLeastAtMost-diff⟨proof⟩*

tts-lemma *less-separate*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows
$\exists x' \in U.\ \exists y' \in U.\ x \in \{..<_{ow} x'\} \wedge y \in \{y' <_{ow} ..\} \wedge \{..<_{ow} x'\} \cap \{y' <_{ow} ..\} = \{\}$
is *order-class.less-separate⟨proof⟩*

tts-lemma *order-iff-strict*:
assumes $a \in U$ **and** $b \in U$
shows $(a \leq_{ow} b) = (a <_{ow} b \vee a = b)$
is *order-class.order.order-iff-strict⟨proof⟩*

tts-lemma *le-less*:
assumes $x \in U$ **and** $y \in U$
shows $(x \leq_{ow} y) = (x <_{ow} y \vee x = y)$
is *order-class.le-less⟨proof⟩*

tts-lemma *strict-iff-order*:
assumes $a \in U$ **and** $b \in U$
shows $(a <_{ow} b) = (a \leq_{ow} b \wedge a \neq b)$
is *order-class.order.strict-iff-order⟨proof⟩*

tts-lemma *less-le*:
assumes $x \in U$ **and** $y \in U$
shows $(x <_{ow} y) = (x \leq_{ow} y \wedge x \neq y)$
is *order-class.less-le⟨proof⟩*

tts-lemma *atLeastAtMost-singleton'*:
assumes $b \in U$ **and** $a = b$
shows $\{a..{}_{ow} b\} = \{a\}$
is *order-class.atLeastAtMost-singleton'⟨proof⟩*

tts-lemma *le-imp-less-or-eq*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $x <_{ow} y \vee x = y$
is *order-class.le-imp-less-or-eq⟨proof⟩*

tts-lemma *antisym-conv*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$
shows $(x \leq_{ow} y) = (x = y)$
is *order-class.antisym-conv⟨proof⟩*

tts-lemma *le-neq-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $a \leq_{ow} b$ **and** $a \neq b$
shows $a <_{ow} b$
is *order-class.le-neq-trans⟨proof⟩*

tts-lemma *neq-le-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $a \neq b$ **and** $a \leq_{ow} b$
shows $a <_{ow} b$
is *order-class.neq-le-trans⟨proof⟩*

tts-lemma *Iio-Int-singleton*:
assumes $k \in U$ **and** $x \in U$
shows $\{..<_{ow} k\} \cap \{x\} = (\textit{if } x <_{ow} k \textit{ then } \{x\} \textit{ else } \{\})$
is *order-class.Iio-Int-singleton⟨proof⟩*

tts-lemma *atLeastAtMost-singleton-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(\{a..{}_{ow}b\} = \{c\}) = (a = b \wedge b = c)$
is *order-class.atLeastAtMost-singleton-iff*⟨*proof*⟩

tts-lemma *Icc-eq-Icc*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $(\{l..{}_{ow}h\} = \{l'..{}_{ow}h'\}) = (h = h' \wedge l = l' \vee \neg\ l' \leq_{ow} h' \wedge \neg\ l \leq_{ow} h)$
is *order-class.Icc-eq-Icc*⟨*proof*⟩

tts-lemma *lift-Suc-mono-less-iff*:
assumes *range* $f \subseteq U$ **and** $\bigwedge n.\ f\ n <_{ow} f\ (Suc\ n)$
shows $(f\ n <_{ow} f\ m) = (n < m)$
is *order-class.lift-Suc-mono-less-iff*⟨*proof*⟩

tts-lemma *lift-Suc-mono-less*:
assumes *range* $f \subseteq U$ **and** $\bigwedge n.\ f\ n <_{ow} f\ (Suc\ n)$ **and** $n < n'$
shows $f\ n <_{ow} f\ n'$
is *order-class.lift-Suc-mono-less*⟨*proof*⟩

tts-lemma *lift-Suc-mono-le*:
assumes *range* $f \subseteq U$ **and** $\bigwedge n.\ f\ n \leq_{ow} f\ (Suc\ n)$ **and** $n \leq n'$
shows $f\ n \leq_{ow} f\ n'$
is *order-class.lift-Suc-mono-le*⟨*proof*⟩

tts-lemma *lift-Suc-antimono-le*:
assumes *range* $f \subseteq U$ **and** $\bigwedge n.\ f\ (Suc\ n) \leq_{ow} f\ n$ **and** $n \leq n'$
shows $f\ n' \leq_{ow} f\ n$
is *order-class.lift-Suc-antimono-le*⟨*proof*⟩

tts-lemma *ivl-disj-int-two*:
assumes $l \in U$ **and** $m \in U$ **and** $u \in U$
shows
$\{l<_{ow}..<_{ow}m\} \cap (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{m..<u\}) = \{\}$
$\{l<_{ow}..m\} \cap \{m<_{ow}..<_{ow}u\} = \{\}$
$(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{l..<m\}) \cap (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{m..<u\}) = \{\}$
$\{l..{}_{ow}m\} \cap \{m<_{ow}..<_{ow}u\} = \{\}$
$\{l<_{ow}..<_{ow}m\} \cap \{m..{}_{ow}u\} = \{\}$
$\{l<_{ow}..m\} \cap \{m<_{ow}..u\} = \{\}$
$(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{l..<m\}) \cap \{m..{}_{ow}u\} = \{\}$
$\{l..{}_{ow}m\} \cap \{m<_{ow}..u\} = \{\}$
is *Set-Interval.ivl-disj-int-two*⟨*proof*⟩

tts-lemma *ivl-disj-int-one*:
assumes $l \in U$ **and** $u \in U$
shows
$\{..{}_{ow}l\} \cap \{l<_{ow}..<_{ow}u\} = \{\}$
$\{..<_{ow}l\} \cap (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{l..<u\}) = \{\}$
$\{..{}_{ow}l\} \cap \{l<_{ow}..u\} = \{\}$
$\{..<_{ow}l\} \cap \{l..{}_{ow}u\} = \{\}$
$\{l<_{ow}..u\} \cap \{u<_{ow}..\} = \{\}$
$\{l<_{ow}..<_{ow}u\} \cap \{u..{}_{ow}\} = \{\}$
$\{l..{}_{ow}u\} \cap \{u<_{ow}..\} = \{\}$
$(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{l..<u\}) \cap \{u..{}_{ow}\} = \{\}$
is *Set-Interval.ivl-disj-int-one*⟨*proof*⟩

tts-lemma *min-absorb2*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$

shows *local.min x y = y*
is *Orderings.min-absorb2⟨proof⟩*

tts-lemma *max-absorb1*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_{ow} x$
shows *local.max x y = x*
is *Orderings.max-absorb1⟨proof⟩*

tts-lemma *atMost-Int-atLeast*:
assumes $n \in U$
shows $\{..{}_{ow} n\} \cap \{n..{}_{ow}\} = \{n\}$
is *Set-Interval.atMost-Int-atLeast⟨proof⟩*

tts-lemma *monoseq-Suc*:
assumes *range X ⊆ U*
shows
(with (≤ow) : «monoseq» X) =
((∀ x. X x ≤ow X (Suc x)) ∨ (∀ x. X (Suc x) ≤ow X x))
is *Topological-Spaces.monoseq-Suc⟨proof⟩*

tts-lemma *mono-SucI2*:
assumes *range X ⊆ U* **and** *∀ x. X (Suc x) ≤ow X x*
shows *with (≤ow) : «monoseq» X*
is *Topological-Spaces.mono-SucI2⟨proof⟩*

tts-lemma *mono-SucI1*:
assumes *range X ⊆ U* **and** *∀ x. X x ≤ow X (Suc x)*
shows *with (≤ow) : «monoseq» X*
is *Topological-Spaces.mono-SucI1⟨proof⟩*

tts-lemma *monoI2*:
assumes *range X ⊆ U* **and** *∀ x y. x ≤ y ⟶ X y ≤ow X x*
shows *with (≤ow) : «monoseq» X*
is *Topological-Spaces.monoI2⟨proof⟩*

tts-lemma *monoI1*:
assumes *range X ⊆ U* **and** *∀ x y. x ≤ y ⟶ X x ≤ow X y*
shows *with (≤ow) : «monoseq» X*
is *Topological-Spaces.monoI1⟨proof⟩*

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *ex-min-if-finite*:
assumes *S ⊆ U*
and *finite S*
and *S ≠ {}*
shows *∃ x∈S. ¬ (∃ y∈S. y <ow x)*
is *Lattices-Big.ex-min-if-finite⟨proof⟩*

end

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹(≤)::['a::*order*, 'a::*order*] ⇒ *bool*› **to** ‹($\leq_{ow}$)›)
and (‹(<)::['a::*order*, 'a::*order*] ⇒ *bool*› **to** ‹($<_{ow}$)›)
substituting *order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *xt1*:
shows
$[\![a = b;\ c <_{ow} b]\!] \Longrightarrow c <_{ow} a$
$[\![b <_{ow} a;\ b = c]\!] \Longrightarrow c <_{ow} a$
$[\![a = b;\ c \leq_{ow} b]\!] \Longrightarrow c \leq_{ow} a$
$[\![b \leq_{ow} a;\ b = c]\!] \Longrightarrow c \leq_{ow} a$
$[\![y \in U;\ x \in U;\ y \leq_{ow} x;\ x \leq_{ow} y]\!] \Longrightarrow x = y$
$[\![y \in U;\ x \in U;\ z \in U;\ y \leq_{ow} x;\ z \leq_{ow} y]\!] \Longrightarrow z \leq_{ow} x$
$[\![y \in U;\ x \in U;\ z \in U;\ y <_{ow} x;\ z \leq_{ow} y]\!] \Longrightarrow z <_{ow} x$
$[\![y \in U;\ x \in U;\ z \in U;\ y \leq_{ow} x;\ z <_{ow} y]\!] \Longrightarrow z <_{ow} x$
$[\![b \in U;\ a \in U;\ b <_{ow} a;\ a <_{ow} b]\!] \Longrightarrow P$
$[\![y \in U;\ x \in U;\ z \in U;\ y <_{ow} x;\ z <_{ow} y]\!] \Longrightarrow z <_{ow} x$
$[\![b \in U;\ a \in U;\ b \leq_{ow} a;\ a \neq b]\!] \Longrightarrow b <_{ow} a$
$[\![a \in U;\ b \in U;\ a \neq b;\ b \leq_{ow} a]\!] \Longrightarrow b <_{ow} a$
$[\![$
$b \in U;$
$c \in U;$
$a = f\ b;$
$c <_{ow} b;$
$\bigwedge x\ y.\ [\![x \in U;\ y \in U;\ y <_{ow} x]\!] \Longrightarrow f\ y <_{ow} f\ x$
$]\!] \Longrightarrow f\ c <_{ow} a$
$[\![$
$b \in U;$
$a \in U;$
$b <_{ow} a;$
$f\ b = c;$
$\bigwedge x\ y.\ [\![x \in U;\ y \in U;\ y <_{ow} x]\!] \Longrightarrow f\ y <_{ow} f\ x$
$]\!] \Longrightarrow c <_{ow} f\ a$
$[\![$
$b \in U;$
$c \in U;$
$a = f\ b;$
$c \leq_{ow} b;$
$\bigwedge x\ y.\ [\![x \in U;\ y \in U;\ y \leq_{ow} x]\!] \Longrightarrow f\ y \leq_{ow} f\ x$
$]\!] \Longrightarrow f\ c \leq_{ow} a$
$[\![$
$b \in U;$
$a \in U;$
$b \leq_{ow} a;$
$f\ b = c;$
$\bigwedge x\ y.\ [\![x \in U;\ y \in U;\ y \leq_{ow} x]\!] \Longrightarrow f\ y \leq_{ow} f\ x$
$]\!] \Longrightarrow c \leq_{ow} f\ a$
is *Orderings.xt1*⟨*proof*⟩

end

end

context *ord-order-ow*
begin

tts-context
tts: (*?'a* **to** U_2) **and** (*?'b* **to** U_1)
sbterms: (‹(≤)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.2}$)›)
and (‹(<)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.2}$)›)
and (‹(≤)::[*?'b*::*ord*, *?'b*::*ord*] ⇒ *bool*› **to** ‹(≤$_{ow.1}$)›)
and (‹(<)::[*?'b*::*ord*, *?'b*::*ord*] ⇒ *bool*› **to** ‹(<$_{ow.1}$)›)
rewriting *ctr-simps*
substituting *ord$_2$.order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *xt2*:
assumes ∀ x∈U_1. $f\ x \in U_2$
and $b \in U_1$
and $a \in U_2$
and $c \in U_1$
and $f\ b \leq_{ow.2} a$
and $c \leq_{ow.1} b$
and ⋀$x\ y$. [[$x \in U_1$; $y \in U_1$; $y \leq_{ow.1} x$]] ⟹ $f\ y \leq_{ow.2} f\ x$
shows $f\ c \leq_{ow.2} a$
is *Orderings.xt2*⟨*proof*⟩

tts-lemma *xt6*:
assumes ∀ x∈U_1. $f\ x \in U_2$
and $b \in U_1$
and $a \in U_2$
and $c \in U_1$
and $f\ b \leq_{ow.2} a$
and $c <_{ow.1} b$
and ⋀$x\ y$. [[$x \in U_1$; $y \in U_1$; $y <_{ow.1} x$]] ⟹ $f\ y <_{ow.2} f\ x$
shows $f\ c <_{ow.2} a$
is *Orderings.xt6*⟨*proof*⟩

end

end

context *order-pair-ow*
begin

tts-context
tts: (*?'a* **to** U_1) **and** (*?'b* **to** U_2)
sbterms: (‹(≤)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.1}$)›)
and (‹(<)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.1}$)›)
and (‹(≤)::[*?'b*::*order*, *?'b*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.2}$)›)
and (‹(<)::[*?'b*::*order*, *?'b*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.2}$)›)
rewriting *ctr-simps*
substituting *ord$_1$.order-ow-axioms* **and** *ord$_2$.order-ow-axioms*
eliminating through *clarsimp*
begin

tts-lemma *xt3*:
assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and ∀ x∈U_1. $f\ x \in U_2$
and $b \leq_{ow.1} a$

and $c \leq_{ow.2} f\ b$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ y \leq_{ow.1} x]\!] \Longrightarrow f\ y \leq_{ow.2} f\ x$
shows $c \leq_{ow.2} f\ a$
is *Orderings.xt3*⟨*proof*⟩

tts-lemma *xt4*:
assumes $\forall\ x{\in}U_2.\ f\ x \in U_1$
and $b \in U_2$
and $a \in U_1$
and $c \in U_2$
and $f\ b <_{ow.1} a$
and $c \leq_{ow.2} b$
and $\bigwedge x\ y.\ [\![x \in U_2;\ y \in U_2;\ y \leq_{ow.2} x]\!] \Longrightarrow f\ y \leq_{ow.1} f\ x$
shows $f\ c <_{ow.1} a$
is *Orderings.xt4*⟨*proof*⟩

tts-lemma *xt5*:
assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall\ x{\in}U_1.\ f\ x \in U_2$
and $b <_{ow.1} a$
and $c \leq_{ow.2} f\ b$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ y <_{ow.1} x]\!] \Longrightarrow f\ y <_{ow.2} f\ x$
shows $c <_{ow.2} f\ a$
is *Orderings.xt5*⟨*proof*⟩

tts-lemma *xt7*:
assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall\ x{\in}U_1.\ f\ x \in U_2$
and $b \leq_{ow.1} a$
and $c <_{ow.2} f\ b$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ y \leq_{ow.1} x]\!] \Longrightarrow f\ y \leq_{ow.2} f\ x$
shows $c <_{ow.2} f\ a$
is *Orderings.xt7*⟨*proof*⟩

tts-lemma *xt8*:
assumes $\forall\ x{\in}U_2.\ f\ x \in U_1$
and $b \in U_2$
and $a \in U_1$
and $c \in U_2$
and $f\ b <_{ow.1} a$
and $c <_{ow.2} b$
and $\bigwedge x\ y.\ [\![x \in U_2;\ y \in U_2;\ y <_{ow.2} x]\!] \Longrightarrow f\ y <_{ow.1} f\ x$
shows $f\ c <_{ow.1} a$
is *Orderings.xt8*⟨*proof*⟩

tts-lemma *xt9*:
assumes $b \in U_1$
and $a \in U_1$
and $c \in U_2$
and $\forall\ x{\in}U_1.\ f\ x \in U_2$
and $b <_{ow.1} a$
and $c <_{ow.2} f\ b$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ y <_{ow.1} x]\!] \Longrightarrow f\ y <_{ow.2} f\ x$
shows $c <_{ow.2} f\ a$

is *Orderings.xt*9⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U_1) **and** (*?'b* **to** U_2)
sbterms: (‹(≤)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.1}$)›)
and (‹(<)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.1}$)›)
and (‹(≤)::[*?'b*::*order*, *?'b*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.2}$)›)
and (‹(<)::[*?'b*::*order*, *?'b*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.2}$)›)
rewriting *ctr-simps*
substituting *ord*$_1$.*order-ow-axioms* **and** *ord*$_2$.*order-ow-axioms*
eliminating through *simp*
begin

tts-lemma *order-less-subst*1:
assumes $a \in U_1$
and $\forall x{\in}U_2.\ f\ x \in U_1$
and $b \in U_2$
and $c \in U_2$
and $a <_{ow.1} f\ b$
and $b <_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_2;\ y \in U_2;\ x <_{ow.2} y]\!] \Longrightarrow f\ x <_{ow.1} f\ y$
shows $a <_{ow.1} f\ c$
is *Orderings.order-less-subst*1⟨*proof*⟩

tts-lemma *order-subst*1:
assumes $a \in U_1$
and $\forall x{\in}U_2.\ f\ x \in U_1$
and $b \in U_2$
and $c \in U_2$
and $a \leq_{ow.1} f\ b$
and $b \leq_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_2;\ y \in U_2;\ x \leq_{ow.2} y]\!] \Longrightarrow f\ x \leq_{ow.1} f\ y$
shows $a \leq_{ow.1} f\ c$
is *Orderings.order-subst*1⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U_1) **and** (*?'c* **to** U_2)
sbterms: (‹(≤)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.1}$)›)
and (‹(<)::[*?'a*::*order*, *?'a*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.1}$)›)
and (‹(≤)::[*?'c*::*order*, *?'c*::*order*] ⇒ *bool*› **to** ‹(≤$_{ow.2}$)›)
and (‹(<)::[*?'c*::*order*, *?'c*::*order*] ⇒ *bool*› **to** ‹(<$_{ow.2}$)›)
rewriting *ctr-simps*
substituting *ord*$_1$.*order-ow-axioms* **and** *ord*$_2$.*order-ow-axioms*
eliminating through *simp*
begin

tts-lemma *order-less-le-subst*2:
assumes $a \in U_1$
and $b \in U_1$
and $\forall x{\in}U_1.\ f\ x \in U_2$
and $c \in U_2$
and $a <_{ow.1} b$
and $f\ b \leq_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ x <_{ow.1} y]\!] \Longrightarrow f\ x <_{ow.2} f\ y$

shows $f\ a <_{ow.2} c$
is *Orderings.order-less-le-subst2*⟨*proof*⟩

tts-lemma *order-le-less-subst2*:
assumes $a \in U_1$
and $b \in U_1$
and $\forall x{\in}U_1.\ f\ x \in U_2$
and $c \in U_2$
and $a \leq_{ow.1} b$
and $f\ b <_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ x \leq_{ow.1} y]\!] \Longrightarrow f\ x \leq_{ow.2} f\ y$
shows $f\ a <_{ow.2} c$
is *Orderings.order-le-less-subst2*⟨*proof*⟩

tts-lemma *order-less-subst2*:
assumes $a \in U_1$
and $b \in U_1$
and $\forall x{\in}U_1.\ f\ x \in U_2$
and $c \in U_2$
and $a <_{ow.1} b$
and $f\ b <_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ x <_{ow.1} y]\!] \Longrightarrow f\ x <_{ow.2} f\ y$
shows $f\ a <_{ow.2} c$
is *Orderings.order-less-subst2*⟨*proof*⟩

tts-lemma *order-subst2*:
assumes $a \in U_1$
and $b \in U_1$
and $\forall x{\in}U_1.\ f\ x \in U_2$
and $c \in U_2$
and $a \leq_{ow.1} b$
and $f\ b \leq_{ow.2} c$
and $\bigwedge x\ y.\ [\![x \in U_1;\ y \in U_1;\ x \leq_{ow.1} y]\!] \Longrightarrow f\ x \leq_{ow.2} f\ y$
shows $f\ a \leq_{ow.2} c$
is *Orderings.order-subst2*⟨*proof*⟩

end

end

3.6.4 Dense orders

Definitions and common properties

locale *dense-order-ow = order-ow U le ls*
for $U :: 'ao\ set$ **and** *le ls* +
assumes *dense*: $[\![\ x \in U;\ y \in U;\ x <_{ow} y\ ]\!] \Longrightarrow (\exists z{\in}U.\ x <_{ow} z \wedge z <_{ow} y)$

Transfer rules

context
includes *lifting-syntax*
begin

lemma *dense-order-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
(*dense-order-ow* (*Collect* (*Domainp A*))) *class.dense-order*

⟨*proof*⟩

end

3.6.5 Partial orders with the greatest element and partial orders with the least elements

Definitions and common properties

locale *ordering-top-ow* = *ordering-ow U le ls*
  for *U* :: *'ao set* **and** *le ls* +
  fixes *top* :: *'ao* (‹T_{ow}›)
  assumes *top-closed*[*simp*]: $\mathsf{T}_{ow} \in U$
  assumes *extremum*[*simp*]: $a \in U \Longrightarrow a \leq_{ow} \mathsf{T}_{ow}$
begin

notation *top* (‹T_{ow}›)

end

locale *bot-ow* =
  fixes *U* :: *'ao set* **and** *bot* (‹$\perp_{ow}$›)
  assumes *bot-closed*[*simp*]: $\perp_{ow} \in U$
begin

notation *bot* (‹$\perp_{ow}$›)

end

locale *bot-pair-ow* = *ord*$_1$: *bot-ow U*$_1$ *bot*$_1$ + *ord*$_2$: *bot-ow U*$_2$ *bot*$_2$
  for *U*$_1$ :: *'ao set* **and** *bot*$_1$ **and** *U*$_2$ :: *'bo set* **and** *bot*$_2$
begin

notation *bot*$_1$ (‹$\perp_{ow.1}$›)
notation *bot*$_2$ (‹$\perp_{ow.2}$›)

end

locale *order-bot-ow* = *order-ow U le ls* + *bot-ow U bot*
  for *U* :: *'ao set* **and** *bot le ls* +
  assumes *bot-least*: $a \in U \Longrightarrow \perp_{ow} \leq_{ow} a$
begin

sublocale *bot*: *ordering-top-ow U* ‹$(\geq_{ow})$› ‹$(>_{ow})$› ‹$\perp_{ow}$›
  ⟨*proof*⟩

no-notation *le* (**infix** ‹$\leq_{ow}$› 50)
  and *ls* (**infix** ‹$<_{ow}$› 50)
  and *top* (‹T_{ow}›)

end

locale *order-bot-pair-ow* =
  ord$_1$: *order-bot-ow U*$_1$ *bot*$_1$ *le*$_1$ *ls*$_1$ + *ord*$_2$: *order-bot-ow U*$_2$ *bot*$_2$ *le*$_2$ *ls*$_2$
  for *U*$_1$ :: *'ao set* **and** *bot*$_1$ *le*$_1$ *ls*$_1$ **and** *U*$_2$ :: *'bo set* **and** *bot*$_2$ *le*$_2$ *ls*$_2$
begin

sublocale *bot-pair-ow* ⟨*proof*⟩

sublocale *order-pair-ow* ⟨*proof*⟩

end

locale *top-ow* =
  fixes U :: *'ao set* **and** *top* (‹$\top_{ow}$›)
  assumes *top-closed*[*simp*]: $\top_{ow} \in U$
begin

notation *top* (‹$\top_{ow}$›)

end

locale *top-pair-ow* = ord_1: *top-ow* U_1 top_1 + ord_2: *top-ow* U_2 top_2
  for U_1 :: *'ao set* **and** top_1 **and** U_2 :: *'bo set* **and** top_2
begin

notation top_1 (‹$\top_{ow.1}$›)
notation top_2 (‹$\top_{ow.2}$›)

end

locale *order-top-ow* = *order-ow U le ls* + *top-ow U top*
  for U :: *'ao set* **and** *le ls top* +
  assumes *top-greatest*: $a \in U \Longrightarrow a \leq_{ow} \top_{ow}$
begin

sublocale *top*: *ordering-top-ow U* ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$\top_{ow}$›
  ⟨*proof*⟩

no-notation *le* (**infix** ‹$\leq_{ow}$› 50)
  and *ls* (**infix** ‹$<_{ow}$› 50)
  and *top* (‹$\top_{ow}$›)

end

locale *order-top-pair-ow* =
  ord_1: *order-top-ow* U_1 le_1 ls_1 top_1 + ord_2: *order-top-ow* U_2 le_2 ls_2 top_2
  for U_1 :: *'ao set* **and** le_1 ls_1 top_1 **and** U_2 :: *'bo set* **and** le_2 ls_2 top_2
begin

sublocale *top-pair-ow* ⟨*proof*⟩
sublocale *order-pair-ow* ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *ordering-top-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> *A* ===> (=))
      (*ordering-top-ow* (*Collect* (*Domainp A*))) *ordering-top*

⟨*proof*⟩

lemma *order-bot-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
(*order-bot-ow* (*Collect* (*Domainp A*))) *class.order-bot*
⟨*proof*⟩

lemma *order-top-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> *A* ===> (=))
(*order-top-ow* (*Collect* (*Domainp A*))) *class.order-top*
⟨*proof*⟩

end

Relativization

context *ordering-top-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ordering-top-ow-axioms*
eliminating through *simp*
applying [*OF top-closed*]
begin

tts-lemma *extremum-uniqueI*:
assumes $\top_{ow} \leq_{ow} \top_{ow}$
shows $\top_{ow} = \top_{ow}$
is *ordering-top.extremum-uniqueI*⟨*proof*⟩

tts-lemma *extremum-unique*:
shows $(\top_{ow} \leq_{ow} \top_{ow}) = (\top_{ow} = \top_{ow})$
is *ordering-top.extremum-unique*⟨*proof*⟩

tts-lemma *extremum-strict*:
shows $\neg\ \top_{ow} <_{ow} \top_{ow}$
is *ordering-top.extremum-strict*⟨*proof*⟩

tts-lemma *not-eq-extremum*:
shows $(\top_{ow} \neq \top_{ow}) = (\top_{ow} <_{ow} \top_{ow})$
is *ordering-top.not-eq-extremum*⟨*proof*⟩

end

end

context *order-bot-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*

substituting *order-bot-ow-axioms*
eliminating through *simp*
begin

tts-lemma *bdd-above-bot*:
assumes $A \subseteq U$
shows *bdd-below A*
is *order-bot-class.bdd-below-bot⟨proof⟩*

tts-lemma *not-less-bot*:
assumes $a \in U$
shows $\neg\ a <_{ow} \bot_{ow}$
is *order-bot-class.not-less-bot⟨proof⟩*

tts-lemma *max-bot*:
assumes $x \in U$
shows $max\ \bot_{ow}\ x = x$
is *order-bot-class.max-bot⟨proof⟩*

tts-lemma *max-bot2*:
assumes $x \in U$
shows $max\ x\ \bot_{ow} = x$
is *order-bot-class.max-bot2⟨proof⟩*

tts-lemma *min-bot*:
assumes $x \in U$
shows $min\ \bot_{ow}\ x = \bot_{ow}$
is *order-bot-class.min-bot⟨proof⟩*

tts-lemma *min-bot2*:
assumes $x \in U$
shows $min\ x\ \bot_{ow} = \bot_{ow}$
is *order-bot-class.min-bot2⟨proof⟩*

tts-lemma *bot-unique*:
assumes $a \in U$
shows $(a \leq_{ow} \bot_{ow}) = (a = \bot_{ow})$
is *order-bot-class.bot-unique⟨proof⟩*

tts-lemma *bot-less*:
assumes $a \in U$
shows $(a \not= \bot_{ow}) = (\bot_{ow} <_{ow} a)$
is *order-bot-class.bot-less⟨proof⟩*

tts-lemma *atLeast-eq-UNIV-iff*:
assumes $x \in U$
shows $(\{x..._{ow}\} = U) = (x = \bot_{ow})$
is *order-bot-class.atLeast-eq-UNIV-iff⟨proof⟩*

tts-lemma *le-bot*:
assumes $a \in U$ **and** $a \leq_{ow} \bot_{ow}$
shows $a = \bot_{ow}$
is *order-bot-class.le-bot⟨proof⟩*

end

end

context *order-top-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *order-top-ow-axioms*
eliminating through *simp*
begin

tts-lemma *bdd-above-top*:
assumes $A \subseteq U$
shows *bdd-above A*
is *order-top-class.bdd-above-top⟨proof⟩*

tts-lemma *not-top-less*:
assumes $a \in U$
shows $\neg\ \top_{ow} <_{ow} a$
is *order-top-class.not-top-less⟨proof⟩*

tts-lemma *max-top*:
assumes $x \in U$
shows $max\ \top_{ow}\ x = \top_{ow}$
is *order-top-class.max-top⟨proof⟩*

tts-lemma *max-top2*:
assumes $x \in U$
shows $max\ x\ \top_{ow} = \top_{ow}$
is *order-top-class.max-top2⟨proof⟩*

tts-lemma *min-top*:
assumes $x \in U$
shows $min\ \top_{ow}\ x = x$
is *order-top-class.min-top⟨proof⟩*

tts-lemma *min-top2*:
assumes $x \in U$
shows $min\ x\ \top_{ow} = x$
is *order-top-class.min-top2⟨proof⟩*

tts-lemma *top-unique*:
assumes $a \in U$
shows $(\top_{ow} \leq_{ow} a) = (a = \top_{ow})$
is *order-top-class.top-unique⟨proof⟩*

tts-lemma *less-top*:
assumes $a \in U$
shows $(a \neq \top_{ow}) = (a <_{ow} \top_{ow})$
is *order-top-class.less-top⟨proof⟩*

tts-lemma *atMost-eq-UNIV-iff*:
assumes $x \in U$
shows $(\{..{}_{ow}x\} = U) = (x = \top_{ow})$
is *order-top-class.atMost-eq-UNIV-iff⟨proof⟩*

tts-lemma *top-le*:
assumes $a \in U$ **and** $\top_{ow} \leq_{ow} a$
shows $a = \top_{ow}$

is *order-top-class.top-le*⟨*proof*⟩

end

end

3.7 Relativization of the results about semigroups

3.7.1 Simple semigroups

Definitions and common properties

locale *semigroup-ow* =
  fixes $U :: 'ag\ set$ **and** $f :: ['ag, 'ag] \Rightarrow 'ag$ (**infixl** ‹$*_{ow}$› 70)
  assumes *f-closed*: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a *_{ow} b \in U$
  assumes *assoc*: $[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow a *_{ow} b *_{ow} c = a *_{ow} (b *_{ow} c)$
begin

notation f (**infixl** ‹$*_{ow}$› 70)

lemma *f-closed'*[*simp*]: $\forall x{\in}U.\ \forall y{\in}U.\ x *_{ow} y \in U$ ⟨*proof*⟩

tts-register-sbts ‹$(*_{ow})$› | U ⟨*proof*⟩

end

lemma *semigroup-ow*: *semigroup* = *semigroup-ow UNIV*
  ⟨*proof*⟩

locale *plus-ow* =
  fixes $U :: 'ag\ set$ **and** *plus* :: $['ag, 'ag] \Rightarrow 'ag$ (**infixl** ‹$+_{ow}$› 65)
  assumes *plus-closed*[*simp*, *intro*]: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a +_{ow} b \in U$
begin

notation *plus* (**infixl** ‹$+_{ow}$› 65)

lemma *plus-closed'*[*simp*]: $\forall x{\in}U.\ \forall y{\in}U.\ x +_{ow} y \in U$ ⟨*proof*⟩

tts-register-sbts ‹$(+_{ow})$› | U ⟨*proof*⟩

end

locale *times-ow* =
  fixes $U :: 'ag\ set$ **and** *times* :: $['ag, 'ag] \Rightarrow 'ag$ (**infixl** ‹$*_{ow}$› 70)
  assumes *times-closed*[*simp*, *intro*]: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a *_{ow} b \in U$
begin

notation *times* (**infixl** ‹$*_{ow}$› 70)

lemma *times-closed'*[*simp*]: $\forall x{\in}U.\ \forall y{\in}U.\ x *_{ow} y \in U$ ⟨*proof*⟩

tts-register-sbts ‹$(*_{ow})$› | U ⟨*proof*⟩

end

locale *semigroup-add-ow* = *plus-ow U plus*
  for $U :: 'ag\ set$ **and** *plus* +
  assumes *plus-assoc*:
    $[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c)$
begin

sublocale *add*: *semigroup-ow U* ‹$(+_{ow})$›
  ⟨*proof*⟩

end

lemma *semigroup-add-ow*: *class.semigroup-add = semigroup-add-ow UNIV*
⟨*proof*⟩

locale *semigroup-mult-ow = times-ow U times*
for *U :: 'ag set* **and** *times +*
assumes *mult-assoc*:
⟦ *a ∈ U*; *b ∈ U*; *c ∈ U* ⟧ ⟹ *a *$_{ow}$ b *$_{ow}$ c = a *$_{ow}$ (b *$_{ow}$ c)*
begin

sublocale *mult*: *semigroup-ow U ‹(*$_{ow}$)›*
⟨*proof*⟩

end

lemma *semigroup-mult-ow*: *class.semigroup-mult = semigroup-mult-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semigroup-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*semigroup-ow* (*Collect* (*Domainp A*))) *semigroup*
⟨*proof*⟩

lemma *semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*semigroup-add-ow* (*Collect* (*Domainp A*))) *class.semigroup-add*
⟨*proof*⟩

lemma *semigroup-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*semigroup-mult-ow* (*Collect* (*Domainp A*))) *class.semigroup-mult*
⟨*proof*⟩

end

3.7.2 Cancellative semigroups

Definitions and common properties

locale *cancel-semigroup-add-ow = semigroup-add-ow U plus*
for *U :: 'ag set* **and** *plus +*
assumes *add-left-imp-eq*:
⟦ *a ∈ U*; *b ∈ U*; *c ∈ U*; *a +*$_{ow}$ *b = a +*$_{ow}$ *c* ⟧ ⟹ *b = c*
assumes *add-right-imp-eq*:
⟦ *b ∈ U*; *a ∈ U*; *c ∈ U*; *b +*$_{ow}$ *a = c +*$_{ow}$ *a* ⟧ ⟹ *b = c*

lemma *cancel-semigroup-add-ow*:

class.cancel-semigroup-add = cancel-semigroup-add-ow UNIV
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *cancel-semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*cancel-semigroup-add-ow* (*Collect* (*Domainp A*)))
class.cancel-semigroup-add
⟨*proof*⟩

end

Relativization

context *cancel-semigroup-add-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *cancel-semigroup-add-ow-axioms*
eliminating through *simp*
begin

tts-lemma *add-right-cancel*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows $(b +_{ow} a = c +_{ow} a) = (b = c)$
is *cancel-semigroup-add-class.add-right-cancel*⟨*proof*⟩

tts-lemma *add-left-cancel*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a +_{ow} b = a +_{ow} c) = (b = c)$
is *cancel-semigroup-add-class.add-left-cancel*⟨*proof*⟩

tts-lemma *bij-betw-add*:
assumes $a \in U$ **and** $A \subseteq U$ **and** $B \subseteq U$
shows *bij-betw* $((+_{ow})\ a)\ A\ B = ((+_{ow})\ a\ `\ A = B)$
is *cancel-semigroup-add-class.bij-betw-add*⟨*proof*⟩

tts-lemma *inj-on-add*:
assumes $a \in U$ **and** $A \subseteq U$
shows *inj-on* $((+_{ow})\ a)\ A$
is *cancel-semigroup-add-class.inj-on-add*⟨*proof*⟩

tts-lemma *inj-on-add'*:
assumes $a \in U$ **and** $A \subseteq U$
shows *inj-on* $(\lambda b.\ b +_{ow} a)\ A$
is *cancel-semigroup-add-class.inj-on-add'*⟨*proof*⟩

end

end

3.7.3 Commutative semigroups

Definitions and common properties

locale *abel-semigroup-ow* =
  semigroup-ow U f **for** *U* :: *'ag set* **and** *f* +
  assumes *commute*: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a *_{ow} b = b *_{ow} a$
begin

lemma *fun-left-comm*:
  assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
  shows $y *_{ow} (x *_{ow} z) = x *_{ow} (y *_{ow} z)$
  ⟨*proof*⟩

end

lemma *abel-semigroup-ow*: *abel-semigroup* = *abel-semigroup-ow UNIV*
  ⟨*proof*⟩

locale *ab-semigroup-add-ow* =
  semigroup-add-ow U plus **for** *U* :: *'ag set* **and** *plus* +
  assumes *add-commute*: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a +_{ow} b = b +_{ow} a$
begin

sublocale *add*: *abel-semigroup-ow U* ‹$(+_{ow})$›
  ⟨*proof*⟩

end

lemma *ab-semigroup-add-ow*: *class.ab-semigroup-add* = *ab-semigroup-add-ow UNIV*
  ⟨*proof*⟩

locale *ab-semigroup-mult-ow* =
  semigroup-mult-ow U times **for** *U* :: *'ag set* **and** *times*+
  assumes *mult-commute*: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a *_{ow} b = b *_{ow} a$
begin

sublocale *mult*: *abel-semigroup-ow U* ‹$(*_{ow})$›
  ⟨*proof*⟩

end

lemma *ab-semigroup-mult-ow*:
  class.ab-semigroup-mult = *ab-semigroup-mult-ow UNIV*
  ⟨*proof*⟩

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *abel-semigroup-transfer*[*transfer-rule*]:
  assumes[*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*A* ===> *A* ===> *A*) ===> (=))
      (*abel-semigroup-ow* (*Collect* (*Domainp A*))) *abel-semigroup*

⟨*proof*⟩

lemma *ab-semigroup-add-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*ab-semigroup-add-ow* (*Collect* (*Domainp A*))) *class.ab-semigroup-add*
⟨*proof*⟩

lemma *ab-semigroup-mult-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*ab-semigroup-mult-ow* (*Collect* (*Domainp A*))) *class.ab-semigroup-mult*
⟨*proof*⟩

end

Relativization

context *abel-semigroup-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *abel-semigroup-ow-axioms*
eliminating through *simp*
begin

tts-lemma *left-commute*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows $b *_{ow} (a *_{ow} c) = a *_{ow} (b *_{ow} c)$
is *abel-semigroup.left-commute*⟨*proof*⟩

end

end

context *ab-semigroup-add-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ab-semigroup-add-ow-axioms*
eliminating through *simp*
begin

tts-lemma *add-ac*:
shows $[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c)$
is *ab-semigroup-add-class.add-ac*(1)
and $[\![a \in U;\ b \in U]\!] \Longrightarrow a +_{ow} b = b +_{ow} a$
is *ab-semigroup-add-class.add-ac*(2)
and $[\![b \in U;\ a \in U;\ c \in U]\!] \Longrightarrow b +_{ow} (a +_{ow} c) = a +_{ow} (b +_{ow} c)$
is *ab-semigroup-add-class.add-ac*(3)⟨*proof*⟩

end

end

context *ab-semigroup-mult-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ab-semigroup-mult-ow-axioms*
eliminating through *simp*
begin

tts-lemma *mult-ac*:
shows $[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow a *_{ow} b *_{ow} c = a *_{ow} (b *_{ow} c)$
is *ab-semigroup-mult-class.mult-ac*(1)
and $[\![a \in U;\ b \in U]\!] \Longrightarrow a *_{ow} b = b *_{ow} a$
is *ab-semigroup-mult-class.mult-ac*(2)
and $[\![b \in U;\ a \in U;\ c \in U]\!] \Longrightarrow b *_{ow} (a *_{ow} c) = a *_{ow} (b *_{ow} c)$
is *ab-semigroup-mult-class.mult-ac*(3)⟨*proof*⟩

end

end

3.7.4 Cancellative commutative semigroups

Definitions and common properties

locale *minus-ow* =
fixes *U* :: *'ag set* **and** *minus* :: [*'ag*, *'ag*] ⇒ *'ag* (**infixl** ‹$-_{ow}$› 65)
assumes *minus-closed*[*simp*,*intro*]: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a -_{ow} b \in U$
begin

notation *minus* (**infixl** ‹$-_{ow}$› 65)

lemma *minus-closed'*[*simp*]: $\forall x{\in}U.\ \forall y{\in}U.\ x -_{ow} y \in U$ ⟨*proof*⟩

tts-register-sbts ‹$(-_{ow})$› | *U* ⟨*proof*⟩

end

locale *cancel-ab-semigroup-add-ow* =
ab-semigroup-add-ow U plus + *minus-ow U minus*
for *U* :: *'ag set* **and** *plus minus* +
assumes *add-diff-cancel-left'*[*simp*]:
$[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow (a +_{ow} b) -_{ow} a = b$
assumes *diff-diff-add*:
$[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow a -_{ow} b -_{ow} c = a -_{ow} (b +_{ow} c)$
begin

sublocale *cancel-semigroup-add-ow U* ‹$(+_{ow})$›
⟨*proof*⟩

end

lemma *cancel-ab-semigroup-add-ow*:
class.cancel-ab-semigroup-add = *cancel-ab-semigroup-add-ow UNIV*

⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *cancel-ab-semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> (=))
(*cancel-ab-semigroup-add-ow* (*Collect* (*Domainp A*)))
class.cancel-ab-semigroup-add
⟨*proof*⟩

end

Relativization

context *cancel-ab-semigroup-add-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *cancel-ab-semigroup-add-ow-axioms*
eliminating through *simp*
begin

tts-lemma *add-diff-cancel-right'*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} b -_{ow} b = a$
is *cancel-ab-semigroup-add-class.add-diff-cancel-right'*⟨*proof*⟩

tts-lemma *add-diff-cancel-right*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $a +_{ow} c -_{ow} (b +_{ow} c) = a -_{ow} b$
is *cancel-ab-semigroup-add-class.add-diff-cancel-right*⟨*proof*⟩

tts-lemma *add-diff-cancel-left*:
assumes $c \in U$ **and** $a \in U$ **and** $b \in U$
shows $c +_{ow} a -_{ow} (c +_{ow} b) = a -_{ow} b$
is *cancel-ab-semigroup-add-class.add-diff-cancel-left*⟨*proof*⟩

tts-lemma *diff-right-commute*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $a -_{ow} c -_{ow} b = a -_{ow} b -_{ow} c$
is *cancel-ab-semigroup-add-class.diff-right-commute*⟨*proof*⟩

tts-lemma *diff-diff-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} b -_{ow} c = a -_{ow} (b +_{ow} c)$
is *diff-diff-eq*⟨*proof*⟩

end

end

3.8 Relativization of the results about monoids

3.8.1 Simple monoids

Definitions and common properties

locale *neutral-ow* =
  fixes U :: *'ag set* **and** z :: *'ag* (‹$\mathbf{1}_{ow}$›)
  assumes *z-closed*[*simp*]: $\mathbf{1}_{ow} \in U$
begin

notation z (‹$\mathbf{1}_{ow}$›)

tts-register-sbts ‹$\mathbf{1}_{ow}$› | U ⟨*proof*⟩

lemma *not-empty*[*simp*]: $U \neq \{\}$ ⟨*proof*⟩

lemma *neutral-map*: $(\lambda y.\ \mathbf{1}_{ow})$ ‘ $A \subseteq U$ ⟨*proof*⟩

end

locale *monoid-ow* = *semigroup-ow U f* + *neutral-ow U z*
  for U :: *'ag set* **and** f z +
  assumes *left-neutral-mow*[*simp*]: $a \in U \Longrightarrow (\mathbf{1}_{ow} *_{ow} a) = a$
    and *right-neutral-mow*[*simp*]: $a \in U \Longrightarrow (a *_{ow} \mathbf{1}_{ow}) = a$

locale *zero-ow* = *zero*: *neutral-ow U zero*
  for U :: *'ag set* **and** *zero* :: *'ag* (‹0_{ow}›)
begin

notation *zero* (‹0_{ow}›)

lemma *zero-closed*: $0_{ow} \in U$ ⟨*proof*⟩

end

lemma *monoid-ow*: *monoid* = *monoid-ow UNIV*
  ⟨*proof*⟩

locale *one-ow* = *one*: *neutral-ow U one*
  for U :: *'ag set* **and** *one* :: *'ag* (‹1_{ow}›)
begin

notation *one* (‹1_{ow}›)

lemma *one-closed*: $1_{ow} \in U$ ⟨*proof*⟩

end

locale *power-ow* = *one-ow U one* + *times-ow U times*
  for U :: *'ag set* **and** *one* :: *'ag* (‹1_{ow}›) **and** *times* (**infixl** ‹$*_{ow}$› 70)

primrec *power-with* :: [*'a*, [*'a*, *'a*] ⇒ *'a*, *'a*, *nat*] ⇒ *'a*
  (‹*'(/with - - : - ^ow -/')*› [1000, 999, 1000, 1000] 10)
  where
    power-0: *power-with one times a* 0 = *one* **for** *one times*
  | *power-Suc*: *power-with one times a* (*Suc n*) =
      times a (*power-with one times a n*) **for** *one times*

lemma *power-with*[*ud-with*]: *power* = *power-with* 1 (∗)
⟨*proof*⟩

context *power-ow*
begin

abbreviation *power* (‹(- ^$_{ow}$ -)› [81, 80] 80) **where**
power ≡ *power-with* 1_{ow} (∗$_{ow}$)

end

locale *monoid-add-ow* =
semigroup-add-ow U plus + *zero-ow U zero* **for** *U* :: 'a*g set* **and** *plus zero* +
assumes *add*-0-*left*: *a* ∈ *U* ⟹ (0_{ow} +$_{ow}$ *a*) = *a*
assumes *add*-0-*right*: *a* ∈ *U* ⟹ (*a* +$_{ow}$ 0_{ow}) = *a*
begin

sublocale *add*: *monoid-ow U* ‹(+$_{ow}$)› ‹0_{ow}›
⟨*proof*⟩

end

lemma *monoid-add-ow*: *class.monoid-add* = *monoid-add-ow UNIV*
⟨*proof*⟩

locale *monoid-mult-ow* = *semigroup-mult-ow U times* + *one-ow U one*
for *U* :: 'a*g set* **and** *one times* +
assumes *mult*-1-*left*: *a* ∈ *U* ⟹ (1_{ow} ∗$_{ow}$ *a*) = *a*
assumes *mult*-1-*right*: *a* ∈ *U* ⟹ (*a* ∗$_{ow}$ 1_{ow}) = *a*
begin

sublocale *mult*: *monoid-ow U* ‹(∗$_{ow}$)› ‹1_{ow}›
⟨*proof*⟩

sublocale *power-ow* ⟨*proof*⟩

end

lemma *monoid-mult-ow*: *class.monoid-mult* = *monoid-mult-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *monoid-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*monoid-ow* (*Collect* (*Domainp A*))) *monoid*
⟨*proof*⟩

lemma *monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))

(*monoid-add-ow* (*Collect* (*Domainp A*))) *class.monoid-add*
⟨*proof*⟩

lemma *monoid-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*A* ===> (*A* ===> *A* ===> *A*) ===> (=))
(*monoid-mult-ow* (*Collect* (*Domainp A*))) *class.monoid-mult*
⟨*proof*⟩

lemma *power-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*A* ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=) ===> *A*) *power-with power-with*
⟨*proof*⟩

end

Relativization

context *power-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹(∗)::[*?'a*::*power*, *?'a*::*power*] ⇒ *?'a*::*power*› **to** ‹(∗$_{ow}$)›)
and (‹1::*?'a*::*power*› **to** ‹1$_{ow}$›)
rewriting *ctr-simps*
substituting *power-ow-axioms* **and** *one.not-empty*
begin

tts-lemma *power-Suc*:
assumes *a* ∈ *U*
shows *a* ^$_{ow}$ *Suc n* = *a* ∗$_{ow}$ *a* ^$_{ow}$ *n*
is *power-class.power.power-Suc*⟨*proof*⟩

tts-lemma *power*-0:
assumes *a* ∈ *U*
shows *a* ^$_{ow}$ 0 = 1$_{ow}$
is *power-class.power.power*-0⟨*proof*⟩

tts-lemma *power-eq-if*:
assumes *p* ∈ *U*
shows *p* ^$_{ow}$ *m* = (*if m* = 0 *then* 1$_{ow}$ *else p* ∗$_{ow}$ *p* ^$_{ow}$ (*m* − 1))
is *power-class.power-eq-if*⟨*proof*⟩

tts-lemma *simps*:
assumes *a* ∈ *U*
shows *a* ^$_{ow}$ 0 = 1$_{ow}$
is *power-class.power.simps*(1)
and *a* ^$_{ow}$ *Suc n* = *a* ∗$_{ow}$ *a* ^$_{ow}$ *n*
is *power-class.power.simps*(2)⟨*proof*⟩

end

end

context *monoid-mult-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *monoid-mult-ow-axioms* **and** *one.not-empty*
applying [*OF one-closed mult.f-closed'*]
begin

tts-lemma *power-commuting-commutes*:
assumes $x \in U$ **and** $y \in U$ **and** $x *_{ow} y = y *_{ow} x$
shows $x \hat{}_{ow} n *_{ow} y = y *_{ow} x \hat{}_{ow} n$
is *monoid-mult-class.power-commuting-commutes*⟨*proof*⟩

tts-lemma *left-right-inverse-power*:
assumes $x \in U$ **and** $y \in U$ **and** $x *_{ow} y = 1_{ow}$
shows $x \hat{}_{ow} n *_{ow} y \hat{}_{ow} n = 1_{ow}$
is *monoid-mult-class.left-right-inverse-power*⟨*proof*⟩

tts-lemma *power-numeral-even*:
assumes $z \in U$
shows $z \hat{}_{ow} numeral\ (num.Bit0\ w) = (let\ w = z \hat{}_{ow} numeral\ w\ in\ w *_{ow} w)$
is *monoid-mult-class.power-numeral-even*⟨*proof*⟩

tts-lemma *power-numeral-odd*:
assumes $z \in U$
shows $z \hat{}_{ow} numeral\ (num.Bit1\ w) = (let\ w = z \hat{}_{ow} numeral\ w\ in\ z *_{ow} w *_{ow} w)$
is *monoid-mult-class.power-numeral-odd*⟨*proof*⟩

tts-lemma *power-minus-mult*:
assumes $a \in U$ **and** $0 < n$
shows $a \hat{}_{ow} (n - 1) *_{ow} a = a \hat{}_{ow} n$
is *monoid-mult-class.power-minus-mult*⟨*proof*⟩

tts-lemma *power-Suc0-right*:
assumes $a \in U$
shows $a \hat{}_{ow} Suc\ 0 = a$
is *monoid-mult-class.power-Suc0-right*⟨*proof*⟩

tts-lemma *power2-eq-square*:
assumes $a \in U$
shows $a \hat{}_{ow} 2 = a *_{ow} a$
is *monoid-mult-class.power2-eq-square*⟨*proof*⟩

tts-lemma *power-one-right*:
assumes $a \in U$
shows $a \hat{}_{ow} 1 = a$
is *monoid-mult-class.power-one-right*⟨*proof*⟩

tts-lemma *power-commutes*:
assumes $a \in U$
shows $a \hat{}_{ow} n *_{ow} a = a *_{ow} a \hat{}_{ow} n$
is *monoid-mult-class.power-commutes*⟨*proof*⟩

tts-lemma *power3-eq-cube*:
assumes $a \in U$
shows $a \hat{}_{ow} 3 = a *_{ow} a *_{ow} a$

is *monoid-mult-class.power3-eq-cube*⟨*proof*⟩

tts-lemma *power-even-eq*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} (2 * n) = (a \mathbin{\widehat{}_{ow}} n) \mathbin{\widehat{}_{ow}} 2$
is *monoid-mult-class.power-even-eq*⟨*proof*⟩

tts-lemma *power-odd-eq*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} Suc\ (2 * n) = a *_{ow} (a \mathbin{\widehat{}_{ow}} n) \mathbin{\widehat{}_{ow}} 2$
is *monoid-mult-class.power-odd-eq*⟨*proof*⟩

tts-lemma *power-mult*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} (m * n) = (a \mathbin{\widehat{}_{ow}} m) \mathbin{\widehat{}_{ow}} n$
is *monoid-mult-class.power-mult*⟨*proof*⟩

tts-lemma *power-Suc2*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} Suc\ n = a \mathbin{\widehat{}_{ow}} n *_{ow} a$
is *monoid-mult-class.power-Suc2*⟨*proof*⟩

tts-lemma *power-one*: $1_{ow} \mathbin{\widehat{}_{ow}} n = 1_{ow}$
is *monoid-mult-class.power-one*⟨*proof*⟩

tts-lemma *power-add*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} (m + n) = a \mathbin{\widehat{}_{ow}} m *_{ow} a \mathbin{\widehat{}_{ow}} n$
is *monoid-mult-class.power-add*⟨*proof*⟩

tts-lemma *power-mult-numeral*:
assumes $a \in U$
shows $(a \mathbin{\widehat{}_{ow}} numeral\ m) \mathbin{\widehat{}_{ow}} numeral\ n = a \mathbin{\widehat{}_{ow}} numeral\ (m * n)$
is *Power.power-mult-numeral*⟨*proof*⟩

tts-lemma *power-add-numeral2*:
assumes $a \in U$ **and** $b \in U$
shows
$a \mathbin{\widehat{}_{ow}} numeral\ m *_{ow} (a \mathbin{\widehat{}_{ow}} numeral\ n *_{ow} b) = a \mathbin{\widehat{}_{ow}} numeral\ (m + n) *_{ow} b$
is *Power.power-add-numeral2*⟨*proof*⟩

tts-lemma *power-add-numeral*:
assumes $a \in U$
shows $a \mathbin{\widehat{}_{ow}} numeral\ m *_{ow} a \mathbin{\widehat{}_{ow}} numeral\ n = a \mathbin{\widehat{}_{ow}} numeral\ (m + n)$
is *Power.power-add-numeral*⟨*proof*⟩

end

end

3.8.2 Commutative monoids

Definitions and common properties

locale *comm-monoid-ow* =
abel-semigroup-ow U f + *neutral-ow U z* **for** $U :: 'ag\ set$ **and** *f z* +
assumes *comm-neutral*: $a \in U \Longrightarrow (a *_{ow} \mathbf{1}_{ow}) = a$
begin

sublocale *monoid-ow U* ‹$(*_{ow})$› ‹$\mathbf{1}_{ow}$›
⟨*proof*⟩

end

lemma *comm-monoid-ow*: *comm-monoid = comm-monoid-ow UNIV*
⟨*proof*⟩

locale *comm-monoid-set-ow = comm-monoid-ow U f z* **for** *U* :: *'ag set* **and** *f z*
begin

tts-register-sbts ‹$(*_{ow})$› | *U* ⟨*proof*⟩

end

lemma *comm-monoid-set-ow*: *comm-monoid-set = comm-monoid-set-ow UNIV*
⟨*proof*⟩

locale *comm-monoid-add-ow =*
ab-semigroup-add-ow U plus + zero-ow U zero
for *U* :: *'ag set* **and** *plus zero +*
assumes *add-0*[*simp*]: $a \in U \Longrightarrow 0_{ow} +_{ow} a = a$
begin

sublocale *add*: *comm-monoid-ow U* ‹$(+_{ow})$› ‹0_{ow}›
⟨*proof*⟩

sublocale *monoid-add-ow U* ‹$(+_{ow})$› ‹0_{ow}› ⟨*proof*⟩

sublocale *sum*: *comm-monoid-set-ow U* ‹$(+_{ow})$› ‹0_{ow}› ⟨*proof*⟩

notation *sum.F* (‹«*sum*»›)

abbreviation *Sum* (‹$\sum_{ow}$/ -› [1000] 1000)
where $\sum_{ow}\ A \equiv$ («*sum*» (λ*x*. *x*) *A*)

notation *Sum* (‹$\sum_{ow}$/ -› [1000] 1000)

end

lemma *comm-monoid-add-ow*: *class.comm-monoid-add = comm-monoid-add-ow UNIV*
⟨*proof*⟩

locale *dvd-ow = times-ow U times*
for *U* :: *'ag set* **and** *times*

ud ‹*dvd.dvd*›
ud *dvd'* ‹*dvd-class.dvd*›

ctr relativization
synthesis *ctr-simps*
assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*)
and [*transfer-rule*]: *bi-unique A right-total A*
trp (*?'a A*)
in *dvd-ow'*: *dvd.with-def*
(‹(*on - with* -: - «*dvd*» -)› [1000, 1000, 1000, 1000] 50)

ctr parametricity
in *dvd-ow''*: *dvd-ow'-def*

context *dvd-ow*
begin

abbreviation *dvd* (**infixr** ‹«*dvd*»› 50) **where** a «*dvd*» $b \equiv$ *dvd-ow'* U $(*_{ow})$ a b
notation *dvd* (**infixr** ‹«*dvd*»› 50)

end

locale *comm-monoid-mult-ow* =
ab-semigroup-mult-ow U times + one-ow U one
for U :: *'ag set* **and** *times one +*
assumes *mult-1*[*simp*]: $a \in U \Longrightarrow 1_{ow} *_{ow} a = a$
begin

sublocale *dvd-ow* ⟨*proof*⟩

sublocale *mult*: *comm-monoid-ow U* ‹$(*_{ow})$› ‹1_{ow}›
⟨*proof*⟩

sublocale *monoid-mult-ow U* ‹1_{ow}› ‹$(*_{ow})$› ⟨*proof*⟩

sublocale *prod*: *comm-monoid-set-ow U* ‹$(*_{ow})$› ‹1_{ow}› ⟨*proof*⟩

notation *prod.F* (‹«*prod*»›)

abbreviation *Prod* (‹$\prod_{ow}$ -› [1000] 1000)
where $\prod_{ow} A \equiv$ («*prod*» $(\lambda x.\ x)$ A)

notation *Prod* (‹$\prod_{ow}$ -› [1000] 1000)

end

lemma *comm-monoid-mult-ow*: *class.comm-monoid-mult = comm-monoid-mult-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *bij-betw-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
((*A* ===> *B*) ===> *rel-set A* ===> *rel-set B* ===> (=)) *bij-betw bij-betw*
⟨*proof*⟩

lemma *comm-monoid-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-monoid-ow* (*Collect* (*Domainp A*))) *comm-monoid*
⟨*proof*⟩

lemma *comm-monoid-set-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-monoid-set-ow* (*Collect* (*Domainp A*))) *comm-monoid-set*
⟨*proof*⟩

lemma *comm-monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-monoid-add-ow* (*Collect* (*Domainp A*))) *class.comm-monoid-add*
⟨*proof*⟩

lemma *comm-monoid-mult-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-monoid-mult-ow* (*Collect* (*Domainp A*))) *class.comm-monoid-mult*
⟨*proof*⟩

lemma *dvd-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> *A* ===> (=))
(*dvd-ow′* (*Collect* (*Domainp A*))) *dvd.with*
⟨*proof*⟩

end

Relativization

context *dvd-ow*
begin

tts-context
tts: (*?′a* **to** *U*)
sbterms: (‹(∗)::[*?′a*::*times*, *?′a*::*times*] ⇒ *?′a*::*times*› **to** ‹(∗$_{ow}$)›)
rewriting *ctr-simps*
substituting *dvd-ow-axioms*
eliminating through *simp*
begin

tts-lemma *dvdI*:
assumes *b* ∈ *U* **and** *k* ∈ *U* **and** *a* = *b* ∗$_{ow}$ *k*
shows *b* «*dvd*» *a*
is *dvd-class.dvdI*⟨*proof*⟩

tts-lemma *dvdE*:
assumes *b* ∈ *U*
and *a* ∈ *U*
and *b* «*dvd*» *a*
and ⋀*k*. [[*k* ∈ *U*; *a* = *b* ∗$_{ow}$ *k*]] ⟹ *P*
shows *P*
is *dvd-class.dvdE*⟨*proof*⟩

end

end

context *comm-monoid-mult-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *comm-monoid-mult-ow-axioms* **and** *one.not-empty*
applying [*OF mult.f-closed' one-closed*]
begin

tts-lemma *strict-subset-divisors-dvd*:
assumes $a \in U$ **and** $b \in U$
shows
$(\{x \in U.\ x\ «dvd»\ a\} \subset \{x \in U.\ x\ «dvd»\ b\}) = (a\ «dvd»\ b \wedge \neg\ b\ «dvd»\ a)$
is *comm-monoid-mult-class.strict-subset-divisors-dvd⟨proof⟩*

tts-lemma *subset-divisors-dvd*:
assumes $a \in U$ **and** $b \in U$
shows $(\{x \in U.\ x\ «dvd»\ a\} \subseteq \{x \in U.\ x\ «dvd»\ b\}) = (a\ «dvd»\ b)$
is *comm-monoid-mult-class.subset-divisors-dvd⟨proof⟩*

tts-lemma *power-mult-distrib*:
assumes $a \in U$ **and** $b \in U$
shows $(a *_{ow} b)\ \hat{}_{ow}\ n = a\ \hat{}_{ow}\ n *_{ow} b\ \hat{}_{ow}\ n$
is *Power.comm-monoid-mult-class.power-mult-distrib⟨proof⟩*

tts-lemma *dvd-triv-right*:
assumes $a \in U$ **and** $b \in U$
shows $a\ «dvd»\ b *_{ow} a$
is *comm-monoid-mult-class.dvd-triv-right⟨proof⟩*

tts-lemma *dvd-mult-right*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a *_{ow} b\ «dvd»\ c$
shows $b\ «dvd»\ c$
is *comm-monoid-mult-class.dvd-mult-right⟨proof⟩*

tts-lemma *mult-dvd-mono*:
assumes $a \in U$
and $b \in U$
and $c \in U$
and $d \in U$
and $a\ «dvd»\ b$
and $c\ «dvd»\ d$
shows $a *_{ow} c\ «dvd»\ b *_{ow} d$
is *comm-monoid-mult-class.mult-dvd-mono⟨proof⟩*

tts-lemma *dvd-triv-left*:
assumes $a \in U$ **and** $b \in U$
shows $a\ «dvd»\ a *_{ow} b$
is *comm-monoid-mult-class.dvd-triv-left⟨proof⟩*

tts-lemma *dvd-mult-left*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a *_{ow} b\ «dvd»\ c$
shows $a\ «dvd»\ c$
is *comm-monoid-mult-class.dvd-mult-left⟨proof⟩*

tts-lemma *dvd-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** a «*dvd*» b **and** b «*dvd*» c
shows a «*dvd*» c
is *comm-monoid-mult-class.dvd-trans*⟨*proof*⟩

tts-lemma *dvd-mult*2:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** a «*dvd*» b
shows a «*dvd*» $b *_{ow} c$
is *comm-monoid-mult-class.dvd-mult*2⟨*proof*⟩

tts-lemma *dvd-refl*:
assumes $a \in U$
shows a «*dvd*» a
is *comm-monoid-mult-class.dvd-refl*⟨*proof*⟩

tts-lemma *dvd-mult*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** a «*dvd*» c
shows a «*dvd*» $b *_{ow} c$
is *comm-monoid-mult-class.dvd-mult*⟨*proof*⟩

tts-lemma *one-dvd*:
assumes $a \in U$
shows 1_{ow} «*dvd*» a
is *comm-monoid-mult-class.one-dvd*⟨*proof*⟩

end

end

3.8.3 Cancellative commutative monoids

Definitions and common properties

locale *cancel-comm-monoid-add-ow* =
cancel-ab-semigroup-add-ow U plus minus +
comm-monoid-add-ow U plus zero
for U :: *'ag set* **and** *plus minus zero*

lemma *cancel-comm-monoid-add-ow*:
class.cancel-comm-monoid-add = *cancel-comm-monoid-add-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *cancel-comm-monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*cancel-comm-monoid-add-ow* (*Collect* (*Domainp A*)))
class.cancel-comm-monoid-add
⟨*proof*⟩

end

Relativization

context *cancel-comm-monoid-add-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *cancel-comm-monoid-add-ow-axioms* **and** *zero.not-empty*
applying [*OF add.f-closed' minus-closed' zero-closed*]
begin

tts-lemma *add-cancel-right-right*:
assumes $a \in U$ **and** $b \in U$
shows $(a = a +_{ow} b) = (b = 0_{ow})$
is *cancel-comm-monoid-add-class.add-cancel-right-right*⟨*proof*⟩

tts-lemma *add-cancel-right-left*:
assumes $a \in U$ **and** $b \in U$
shows $(a = b +_{ow} a) = (b = 0_{ow})$
is *cancel-comm-monoid-add-class.add-cancel-right-left*⟨*proof*⟩

tts-lemma *add-cancel-left-right*:
assumes $a \in U$ **and** $b \in U$
shows $(a +_{ow} b = a) = (b = 0_{ow})$
is *cancel-comm-monoid-add-class.add-cancel-left-right*⟨*proof*⟩

tts-lemma *add-cancel-left-left*:
assumes $b \in U$ **and** $a \in U$
shows $(b +_{ow} a = a) = (b = 0_{ow})$
is *cancel-comm-monoid-add-class.add-cancel-left-left*⟨*proof*⟩

tts-lemma *add-implies-diff*:
assumes $c \in U$ **and** $b \in U$ **and** $a \in U$ **and** $c +_{ow} b = a$
shows $c = a -_{ow} b$
is *cancel-comm-monoid-add-class.add-implies-diff*⟨*proof*⟩

tts-lemma *diff-cancel*:
assumes $a \in U$
shows $a -_{ow} a = 0_{ow}$
is *cancel-comm-monoid-add-class.diff-cancel*⟨*proof*⟩

tts-lemma *diff-zero*:
assumes $a \in U$
shows $a -_{ow} 0_{ow} = a$
is *cancel-comm-monoid-add-class.diff-zero*⟨*proof*⟩

end

end

3.9 Relativization of the results about groups

3.9.1 Simple groups

Definitions and common properties

locale *group-ow* = *semigroup-ow U f* **for** $U :: 'ag\ set$ **and** f +
  fixes z ($\langle \mathbf{1}_{ow} \rangle$)
    and *inverse* :: $'ag \Rightarrow 'ag$
  assumes *z-closed*[*simp*]: $\mathbf{1}_{ow} \in U$
    and *inverse-closed*[*simp*]: $a \in U \Longrightarrow inverse\ a \in U$
    and *group-left-neutral*: $a \in U \Longrightarrow \mathbf{1}_{ow} *_{ow} a = a$
    and *left-inverse*[*simp*]: $a \in U \Longrightarrow inverse\ a *_{ow} a = \mathbf{1}_{ow}$
begin

notation z ($\langle \mathbf{1}_{ow} \rangle$)

lemma *inverse-closed'*: *inverse* ‘ $U \subseteq U$ ⟨*proof*⟩
lemma *inverse-closed''*: $\forall x \in U.\ inverse\ x \in U$ ⟨*proof*⟩

lemma *left-cancel*:
  assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
  shows $a *_{ow} b = a *_{ow} c \longleftrightarrow b = c$
⟨*proof*⟩

sublocale *monoid-ow U* ‹$(*_{ow})$› ‹$\mathbf{1}_{ow}$›
⟨*proof*⟩

lemma *inverse-image*[*simp*]: *inverse* ‘ $U \subseteq U$ ⟨*proof*⟩

end

lemma *group-ow*: *group* = *group-ow UNIV*
  ⟨*proof*⟩

locale *uminus-ow* =
  fixes $U :: 'ag\ set$ **and** *uminus* :: $'ag \Rightarrow 'ag$ ($\langle -_{ow}$ -› [81] 80)
  assumes *uminus-closed*: $a \in U \Longrightarrow -_{ow}\ a \in U$
begin

notation *uminus* ($\langle -_{ow}$ -› [81] 80)

lemma *uminus-closed'*: *uminus* ‘ $U \subseteq U$ ⟨*proof*⟩
lemma *uminus-closed''*: $\forall a \in U.\ -_{ow}\ a \in U$ ⟨*proof*⟩

tts-register-sbts *uminus* | U ⟨*proof*⟩

end

locale *group-add-ow* =
  minus-ow U minus + *uminus-ow U uminus* + *monoid-add-ow U plus zero*
  for $U :: 'ag\ set$ **and** *minus plus zero uminus* +
  assumes *left-inverse*: $a \in U \Longrightarrow (-_{ow}\ a) +_{ow} a = 0_{ow}$
    and *add-inv-conv-diff*: $[\![a \in U;\ b \in U]\!] \Longrightarrow a +_{ow} (-_{ow}\ b) = a -_{ow} b$
begin

sublocale *add*: *group-ow U* ‹$(+_{ow})$› ‹0_{ow}› *uminus*
  ⟨*proof*⟩

lemma *inverse-unique*:
assumes $a \in U$ **and** $b \in U$ **and** $a +_{ow} b = 0_{ow}$
shows $-_{ow}\ a = b$
⟨*proof*⟩

lemma *inverse-neutral*[*simp*]: $-_{ow}\ 0_{ow} = 0_{ow}$
⟨*proof*⟩

lemma *inverse-inverse*:
assumes $a \in U$
shows $-_{ow}\ (-_{ow}\ a) = a$
⟨*proof*⟩

lemma *right-inverse*:
assumes $a \in U$
shows $a +_{ow} (-_{ow}\ a) = 0_{ow}$
⟨*proof*⟩

sublocale *cancel-semigroup-add-ow U* ‹$(+_{ow})$›
⟨*proof*⟩

end

lemma *group-add-ow*: *class.group-add* = *group-add-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *group-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows ((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A*) ===> (=))
(*group-ow* (*Collect* (*Domainp A*))) *group*
⟨*proof*⟩

lemma *group-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A*) ===> (=))
(*group-add-ow* (*Collect* (*Domainp A*))) *class.group-add*
⟨*proof*⟩

end

Relativization

context *group-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *group-ow-axioms* **and** *not-empty*
applying [*OF f-closed' z-closed inverse-closed''*]
begin

tts-lemma *inverse-neutral*: *inverse* $\mathbf{1}_{ow} = \mathbf{1}_{ow}$
is *group.inverse-neutral*⟨*proof*⟩

tts-lemma *inverse-inverse*:
assumes $a \in U$
shows *inverse* (*inverse* a) = a
is *group.inverse-inverse*⟨*proof*⟩

tts-lemma *right-inverse*:
assumes $a \in U$
shows $a *_{ow}$ *inverse* $a = \mathbf{1}_{ow}$
is *group.right-inverse*⟨*proof*⟩

tts-lemma *inverse-distrib-swap*:
assumes $a \in U$ **and** $b \in U$
shows *inverse* ($a *_{ow} b$) = *inverse* $b *_{ow}$ *inverse* a
is *group.inverse-distrib-swap*⟨*proof*⟩

tts-lemma *right-cancel*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows ($b *_{ow} a = c *_{ow} a$) = ($b = c$)
is *group.right-cancel*⟨*proof*⟩

tts-lemma *inverse-unique*:
assumes $a \in U$ **and** $b \in U$ **and** $a *_{ow} b = \mathbf{1}_{ow}$
shows *inverse* $a = b$
is *group.inverse-unique*⟨*proof*⟩
end

end

context *group-add-ow*
begin

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *group-add-ow-axioms* **and** *zero.not-empty*
applying [*OF minus-closed' plus-closed' zero-closed add.inverse-closed''*]
begin

tts-lemma *diff*-0:
assumes $a \in U$
shows $0_{ow} -_{ow} a = -_{ow} a$
is *group-add-class.diff*-0⟨*proof*⟩

tts-lemma *diff*-0-*right*:
assumes $a \in U$
shows $a -_{ow} 0_{ow} = a$
is *group-add-class.diff*-0-*right*⟨*proof*⟩

tts-lemma *diff-self*:
assumes $a \in U$
shows $a -_{ow} a = 0_{ow}$
is *group-add-class.diff-self*⟨*proof*⟩

tts-lemma *group-left-neutral*:

assumes $a \in U$
shows $0_{ow} +_{ow} a = a$
is *group-add-class.add.group-left-neutral*⟨*proof*⟩

tts-lemma *minus-minus*:
assumes $a \in U$
shows $-_{ow} (-_{ow} a) = a$
is *group-add-class.minus-minus*⟨*proof*⟩

tts-lemma *right-minus*:
assumes $a \in U$
shows $a +_{ow} -_{ow} a = 0_{ow}$
is *group-add-class.right-minus*⟨*proof*⟩

tts-lemma *left-minus*:
assumes $a \in U$
shows $-_{ow} a +_{ow} a = 0_{ow}$
is *group-add-class.left-minus*⟨*proof*⟩

tts-lemma *add-diff-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} b -_{ow} b = a$
is *group-add-class.add-diff-cancel*⟨*proof*⟩

tts-lemma *diff-add-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} b +_{ow} b = a$
is *group-add-class.diff-add-cancel*⟨*proof*⟩

tts-lemma *diff-conv-add-uminus*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} b = a +_{ow} -_{ow} b$
is *group-add-class.diff-conv-add-uminus*⟨*proof*⟩

tts-lemma *diff-minus-eq-add*:
assumes $a \in U$ **and** $b \in U$
shows $a -_{ow} -_{ow} b = a +_{ow} b$
is *group-add-class.diff-minus-eq-add*⟨*proof*⟩

tts-lemma *add-uminus-conv-diff*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} -_{ow} b = a -_{ow} b$
is *group-add-class.add-uminus-conv-diff*⟨*proof*⟩

tts-lemma *minus-diff-eq*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} (a -_{ow} b) = b -_{ow} a$
is *group-add-class.minus-diff-eq*⟨*proof*⟩

tts-lemma *add-minus-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $a +_{ow} (-_{ow} a +_{ow} b) = b$
is *group-add-class.add-minus-cancel*⟨*proof*⟩

tts-lemma *minus-add-cancel*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} a +_{ow} (a +_{ow} b) = b$
is *group-add-class.minus-add-cancel*⟨*proof*⟩

tts-lemma *neg-0-equal-iff-equal*:
assumes $a \in U$
shows $(0_{ow} = -_{ow}\ a) = (0_{ow} = a)$
is *group-add-class.neg-0-equal-iff-equal*⟨*proof*⟩

tts-lemma *neg-equal-0-iff-equal*:
assumes $a \in U$
shows $(-_{ow}\ a = 0_{ow}) = (a = 0_{ow})$
is *group-add-class.neg-equal-0-iff-equal*⟨*proof*⟩

tts-lemma *eq-iff-diff-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(a = b) = (a -_{ow} b = 0_{ow})$
is *group-add-class.eq-iff-diff-eq-0*⟨*proof*⟩

tts-lemma *equation-minus-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(a = -_{ow}\ b) = (b = -_{ow}\ a)$
is *group-add-class.equation-minus-iff*⟨*proof*⟩

tts-lemma *minus-equation-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow}\ a = b) = (-_{ow}\ b = a)$
is *group-add-class.minus-equation-iff*⟨*proof*⟩

tts-lemma *neg-equal-iff-equal*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow}\ a = -_{ow}\ b) = (a = b)$
is *group-add-class.neg-equal-iff-equal*⟨*proof*⟩

tts-lemma *right-minus-eq*:
assumes $a \in U$ **and** $b \in U$
shows $(a -_{ow} b = 0_{ow}) = (a = b)$
is *group-add-class.right-minus-eq*⟨*proof*⟩

tts-lemma *minus-add*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow}\ (a +_{ow} b) = -_{ow}\ b +_{ow} -_{ow}\ a$
is *group-add-class.minus-add*⟨*proof*⟩

tts-lemma *eq-neg-iff-add-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(a = -_{ow}\ b) = (a +_{ow} b = 0_{ow})$
is *group-add-class.eq-neg-iff-add-eq-0*⟨*proof*⟩

tts-lemma *neg-eq-iff-add-eq-0*:
assumes $a \in U$ **and** $b \in U$
shows $(-_{ow}\ a = b) = (a +_{ow} b = 0_{ow})$
is *group-add-class.neg-eq-iff-add-eq-0*⟨*proof*⟩

tts-lemma *add-eq-0-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(a +_{ow} b = 0_{ow}) = (a = -_{ow}\ b)$
is *group-add-class.add-eq-0-iff2*⟨*proof*⟩

tts-lemma *add-eq-0-iff*:
assumes $a \in U$ **and** $b \in U$

shows $(a +_{ow} b = 0_{ow}) = (b = -_{ow} a)$
is *group-add-class.add-eq-0-iff*⟨*proof*⟩

tts-lemma *diff-diff-eq2*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} (b -_{ow} c) = a +_{ow} c -_{ow} b$
is *group-add-class.diff-diff-eq2*⟨*proof*⟩

tts-lemma *diff-add-eq-diff-diff-swap*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} (b +_{ow} c) = a -_{ow} c -_{ow} b$
is *group-add-class.diff-add-eq-diff-diff-swap*⟨*proof*⟩

tts-lemma *add-diff-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a +_{ow} (b -_{ow} c) = a +_{ow} b -_{ow} c$
is *group-add-class.add-diff-eq*⟨*proof*⟩

tts-lemma *eq-diff-eq*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $(a = c -_{ow} b) = (a +_{ow} b = c)$
is *group-add-class.eq-diff-eq*⟨*proof*⟩

tts-lemma *diff-eq-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a -_{ow} b = c) = (a = c +_{ow} b)$
is *group-add-class.diff-eq-eq*⟨*proof*⟩

tts-lemma *left-cancel*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a +_{ow} b = a +_{ow} c) = (b = c)$
is *group-add-class.add.left-cancel*⟨*proof*⟩

tts-lemma *right-cancel*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows $(b +_{ow} a = c +_{ow} a) = (b = c)$
is *group-add-class.add.right-cancel*⟨*proof*⟩

tts-lemma *minus-unique*:
assumes $a \in U$ **and** $b \in U$ **and** $a +_{ow} b = 0_{ow}$
shows $-_{ow} a = b$
is *group-add-class.minus-unique*⟨*proof*⟩

tts-lemma *diff-eq-diff-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$ **and** $a -_{ow} b = c -_{ow} d$
shows $(a = b) = (c = d)$
is *group-add-class.diff-eq-diff-eq*⟨*proof*⟩

end

end

3.9.2 Abelian groups

Definitions and common properties

locale *ab-group-add-ow* =
minus-ow U minus + uminus-ow U uminus + comm-monoid-add-ow U plus zero

for $U :: 'ag\ set$ **and** *plus zero minus uminus* +
assumes *ab-left-minus*: $a \in U \Longrightarrow -_{ow}\ a +_{ow} a = 0_{ow}$
assumes *ab-diff-conv-add-uminus*:
$[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow a -_{ow} b = a +_{ow} (-_{ow}\ b)$
begin

sublocale *group-add-ow*
⟨*proof*⟩

sublocale *cancel-comm-monoid-add-ow*
⟨*proof*⟩

end

lemma *ab-group-add-ow*: *class.ab-group-add = ab-group-add-ow UNIV*
⟨*proof*⟩

lemma *ab-group-add-ow-UNIV-axioms*:
ab-group-add-ow (*UNIV*::'*a*::*ab-group-add set*) (+) 0 (−) *uminus*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *ab-group-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A*) ===> (=))
(*ab-group-add-ow* (*Collect* (*Domainp A*))) *class.ab-group-add*
⟨*proof*⟩

end

Relativization

context *ab-group-add-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ab-group-add-ow-axioms* **and** *zero.not-empty*
applying [*OF plus-closed' zero-closed minus-closed' add.inverse-closed''*]
begin

tts-lemma *uminus-add-conv-diff*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow}\ a +_{ow} b = b -_{ow} a$
is *ab-group-add-class.uminus-add-conv-diff*⟨*proof*⟩

tts-lemma *diff-add-eq*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a -_{ow} b +_{ow} c = a +_{ow} c -_{ow} b$
is *ab-group-add-class.diff-add-eq*⟨*proof*⟩

end

end

3.10 Relativization of the results about semirings

3.10.1 Semirings

Definitions and common properties

locale *semiring-ow* =
  ab-semigroup-add-ow U plus + *semigroup-mult-ow U times*
  for *U* :: *'ag set* **and** *plus times* +
  assumes *distrib-right*[*simp*]:
    $[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$
  assumes *distrib-left*[*simp*]:
    $[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow a *_{ow} (b +_{ow} c) = a *_{ow} b +_{ow} a *_{ow} c$

lemma *semiring-ow*: *class.semiring* = *semiring-ow UNIV*
  ⟨*proof*⟩

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *semiring-transfer*[*transfer-rule*]:
  assumes[*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> (=))
      (*semiring-ow* (*Collect* (*Domainp A*))) *class.semiring*
  ⟨*proof*⟩

end

Relativization

context *semiring-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  substituting *semiring-ow-axioms*
  eliminating through *simp*
begin

tts-lemma *combine-common-factor*:
  assumes $a \in U$ **and** $e \in U$ **and** $b \in U$ **and** $c \in U$
  shows $a *_{ow} e +_{ow} (b *_{ow} e +_{ow} c) = (a +_{ow} b) *_{ow} e +_{ow} c$
    is *semiring-class.combine-common-factor*⟨*proof*⟩

end

end

3.10.2 Commutative semirings

Definitions and common properties

locale *comm-semiring-ow* =
  ab-semigroup-add-ow U plus + *ab-semigroup-mult-ow U times*
  for *U* :: *'ag set* **and** *plus times* +
  assumes *distrib*:

$[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$
begin

sublocale *semiring-ow*
⟨*proof*⟩

end

lemma *comm-semiring-ow*: *class.comm-semiring = comm-semiring-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> (=))
(*comm-semiring-ow* (*Collect* (*Domainp A*))) *class.comm-semiring*
(**is** *?PR* (*comm-semiring-ow* (*Collect* (*Domainp A*))) *class.comm-semiring*)
⟨*proof*⟩

end

3.10.3 Semirings with zero

Definitions and further results

locale *mult-zero-ow = times-ow U times + zero-ow U zero*
for *U* :: *'ag set* **and** *times zero +*
assumes *mult-zero-left*[*simp*]: $a \in U \Longrightarrow 0_{ow} *_{ow} a = 0_{ow}$
assumes *mult-zero-right*[*simp*]: $a \in U \Longrightarrow a *_{ow} 0_{ow} = 0_{ow}$

lemma *mult-zero-ow*: *class.mult-zero = mult-zero-ow UNIV*
⟨*proof*⟩

locale *semiring*-0-*ow =*
semiring-ow U plus times +
comm-monoid-add-ow U plus zero +
mult-zero-ow U times zero
for *U* :: *'ag set* **and** *plus zero times*

lemma *semiring*-0-*ow*: *class.semiring*-0 = *semiring*-0-*ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semiring*-0-*transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A* ===> *A*) ===> (=))
(*semiring*-0-*ow* (*Collect* (*Domainp A*))) *class.semiring*-0

(**is** *?PR* (*semiring*-0-*ow* (*Collect* (*Domainp A*))) *class.semiring*-0)
⟨*proof*⟩

end

3.10.4 Commutative semirings with zero

Definitions and common properties

locale *comm-semiring*-0-*ow* =
comm-semiring-ow U plus times +
comm-monoid-add-ow U plus zero +
mult-zero-ow U times zero
for *U* :: *'ag set* **and** *plus zero times*
begin

sublocale *semiring*-0-*ow* ⟨*proof*⟩

end

lemma *comm-semiring*-0-*ow*: *class.comm-semiring*-0 = *comm-semiring*-0-*ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring*-0-*transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A* ===> *A*) ===> (=))
(*comm-semiring*-0-*ow* (*Collect* (*Domainp A*))) *class.comm-semiring*-0
(**is** *?PR* (*comm-semiring*-0-*ow* (*Collect* (*Domainp A*))) *class.comm-semiring*-0)
⟨*proof*⟩

end

3.10.5 Cancellative semirings with zero

Definitions and common properties

locale *semiring*-0-*cancel-ow* =
semiring-ow U plus times + *cancel-comm-monoid-add-ow U plus minus zero*
for *U* :: *'ag set* **and** *plus minus zero times*
begin

sublocale *semiring*-0-*ow*
⟨*proof*⟩

end

lemma *semiring*-0-*cancel-ow*:
class.semiring-0-*cancel* = *semiring*-0-*cancel-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semiring-0-cancel-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(=)
) (*semiring-0-cancel-ow* (*Collect* (*Domainp A*))) *class.semiring-0-cancel*
⟨*proof*⟩

end

3.10.6 Commutative cancellative semirings with zero

Definitions and common properties

locale *comm-semiring-0-cancel-ow* =
comm-semiring-ow U plus times +
cancel-comm-monoid-add-ow U plus minus zero
for *U* :: *'ag set* **and** *plus minus zero times*
begin

sublocale *semiring-0-cancel-ow* ⟨*proof*⟩

sublocale *comm-semiring-0-ow* ⟨*proof*⟩

end

lemma *comm-semiring-0-cancel-ow*:
class.comm-semiring-0-cancel = *comm-semiring-0-cancel-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring-0-cancel-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(=)
)
(*comm-semiring-0-cancel-ow* (*Collect* (*Domainp A*)))
class.comm-semiring-0-cancel
⟨*proof*⟩

end

3.10.7 Class *zero-neq-one*

Definitions and common properties

locale *zero-neq-one-ow* =
  zero-ow U zero + *one-ow U one*
  for U :: *'ag set* **and** *one* (‹1_{ow}›) **and** *zero* (‹0_{ow}›) +
  assumes *zero-neq-one*[*simp*]: $0_{ow} \neq 1_{ow}$

lemma *zero-neq-one-ow*: *class.zero-neq-one* = *zero-neq-one-ow UNIV*
  ⟨*proof*⟩

ud ‹*zero-neq-one.of-bool*› (‹(*with* - - : «*of'-bool*» -)› [1000, 999, 1000] 10)
ud *of-bool'* ‹*of-bool*›

ctr parametricity
  in *of-bool.with-def*

context *zero-neq-one-ow*
begin

abbreviation *of-bool* **where** *of-bool* ≡ *of-bool.with* 1_{ow} 0_{ow}

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *zero-neq-one-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (*A* ===> *A* ===> (=))
      (*zero-neq-one-ow* (*Collect* (*Domainp A*))) *class.zero-neq-one*
    (**is** *?PR* (*zero-neq-one-ow* (*Collect* (*Domainp A*))) *class.zero-neq-one*)
⟨*proof*⟩

end

Relativization

context *zero-neq-one-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  rewriting *ctr-simps*
  substituting *zero-neq-one-ow-axioms*
  eliminating through *simp*
begin

tts-lemma *split-of-bool-asm*:
  shows *P* (*of-bool p*) = (¬ (*p* ∧ ¬ *P* 1_{ow} ∨ ¬ *p* ∧ ¬ *P* 0_{ow}))
    is *zero-neq-one-class.split-of-bool-asm*⟨*proof*⟩

tts-lemma *of-bool-eq-iff*:
shows (*of-bool p* = *local.of-bool q*) = (*p* = *q*)
is *zero-neq-one-class.of-bool-eq-iff*⟨*proof*⟩

tts-lemma *split-of-bool*:
shows *P* (*of-bool p*) = ((*p* ⟶ *P* 1_{ow}) ∧ (¬ *p* ⟶ *P* 0_{ow}))
is *zero-neq-one-class.split-of-bool*⟨*proof*⟩

tts-lemma *one-neq-zero*: $1_{ow} \neq 0_{ow}$
is *zero-neq-one-class.one-neq-zero*⟨*proof*⟩

tts-lemma *of-bool-eq*:
shows *of-bool False* = 0_{ow}
is *zero-neq-one-class.of-bool-eq*(1)
and *of-bool True* = 1_{ow}
is *zero-neq-one-class.of-bool-eq*(2)⟨*proof*⟩

end

end

3.10.8 Semirings with zero and one (rigs)

Definitions and commmon properties

locale *semiring*-1-*ow* =
zero-neq-one-ow U one zero +
semiring-0-*ow U plus zero times* +
monoid-mult-ow U one times
for *U* :: *'ag set* **and** *one times plus zero*

lemma *semiring*-1-*ow*: *class.semiring*-1 = *semiring*-1-*ow UNIV*
⟨*proof*⟩

ud ‹*semiring*-1.*of-nat*› (‹(*with* - - - : «*of'-nat*» -)› [1000, 999, 998, 1000] 10)
ud *of-nat'* ‹*of-nat*›

ud ‹*semiring*-1.*Nats*› (‹(*with* - - - : ℕ)› [1000, 999, 998] 10)
ud *Nats'* ‹*Nats*›

ctr parametricity
in *of-nat-ow*: *of-nat.with-def*
and *Nats-ow*: *Nats.with-def*

context *semiring*-1-*ow*
begin

abbreviation *of-nat* **where** *of-nat* ≡ *of-nat.with* 1_{ow} ($+_{ow}$) 0_{ow}
abbreviation *Nats* (‹«ℕ»›) **where** «ℕ» ≡ *Nats.with* 1_{ow} ($+_{ow}$) 0_{ow}
notation *Nats* (‹«ℕ»›)

end

context *semiring*-1
begin

lemma *Nat-ss-UNIV*: ℕ ⊆ *UNIV* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semiring-1-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*A* ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*semiring-1-ow* (*Collect* (*Domainp A*))) *class.semiring-1*
⟨*proof*⟩

end

Relativization

context *semiring-1-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *semiring-1-ow-axioms* **and** *zero.not-empty*
eliminating through *simp*
begin

tts-lemma *Nat-ss-UNIV*[*simp*]:
shows «ℕ» ⊆ *U*
is *Nat-ss-UNIV*⟨*proof*⟩

end

lemma *Nat-closed*[*simp*, *intro*]: *a* ∈ «ℕ» ⟹ *a* ∈ *U* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *semiring-1-ow-axioms* **and** *zero.not-empty*
eliminating through *auto*
begin

tts-lemma *mult-of-nat-commute*:
assumes *y* ∈ *U*
shows *of-nat x* $*_{ow}$ *y* = *y* $*_{ow}$ *of-nat x*
is *semiring-1-class.mult-of-nat-commute*⟨*proof*⟩

tts-lemma *of-bool-conj*: *of-bool* (*P* ∧ *Q*) = *of-bool P* $*_{ow}$ *of-bool Q*
is *semiring-1-class.of-bool-conj*⟨*proof*⟩

tts-lemma *power-0-left*: 0_{ow} $\hat{}_{ow}$ *n* = (*if n* = 0 *then* 1_{ow} *else* 0_{ow})
is *semiring-1-class.power-0-left*⟨*proof*⟩

tts-lemma *of-nat-power*: *of-nat* ((*with* 1 (*)) : *m* $\hat{}_{ow}$ *n*)) = *of-nat m* $\hat{}_{ow}$ *n*
is *semiring-1-class.of-nat-power*⟨*proof*⟩

tts-lemma *of-nat-of-bool*: *of-nat* (*with* 1 0 : «*of-bool*» *P*) = *of-bool P*
is *semiring-1-class.of-nat-of-bool*⟨*proof*⟩

tts-lemma *of-nat-in-Nats*: *of-nat n* ∈ «ℕ»
is *semiring-1-class.of-nat-in-Nats*⟨*proof*⟩

tts-lemma *zero-power2*: 0_{ow} $\widehat{}_{ow}$ 2 = 0_{ow}
is *semiring-1-class.zero-power2*⟨*proof*⟩

tts-lemma *power-0-Suc*: 0_{ow} $\widehat{}_{ow}$ *Suc n* = 0_{ow}
is *semiring-1-class.power-0-Suc*⟨*proof*⟩

tts-lemma *zero-power*:
assumes 0 < *n*
shows 0_{ow} $\widehat{}_{ow}$ *n* = 0_{ow}
is *semiring-1-class.zero-power*⟨*proof*⟩

tts-lemma *one-power2*: 1_{ow} $\widehat{}_{ow}$ 2 = 1_{ow}
is *semiring-1-class.one-power2*⟨*proof*⟩

tts-lemma *of-nat-simps*:
shows *of-nat* 0 = 0_{ow}
is *semiring-1-class.of-nat-simps*(1)
and *of-nat* (*Suc m*) = 1_{ow} +$_{ow}$ *of-nat m*
is *semiring-1-class.of-nat-simps*(2)⟨*proof*⟩

tts-lemma *of-nat-mult*: *of-nat* (*m* * *n*) = *of-nat m* *$_{ow}$ *of-nat n*
is *semiring-1-class.of-nat-mult*⟨*proof*⟩

tts-lemma *Nats-induct*:
assumes *x* ∈ «ℕ» **and** ⋀*n*. *P* (*of-nat n*)
shows *P x*
is *semiring-1-class.Nats-induct*⟨*proof*⟩

tts-lemma *of-nat-add*: *of-nat* (*m* + *n*) = *of-nat m* +$_{ow}$ *of-nat n*
is *semiring-1-class.of-nat-add*⟨*proof*⟩

tts-lemma *of-nat-Suc*: *of-nat* (*Suc m*) = 1_{ow} +$_{ow}$ *of-nat m*
is *semiring-1-class.of-nat-Suc*⟨*proof*⟩

tts-lemma *Nats-cases*:
assumes *x* ∈ «ℕ»
obtains (*of-nat*) *n* **where** *x* = *of-nat n*
given *semiring-1-class.Nats-cases* ⟨*proof*⟩

tts-lemma *Nats-mult*:
assumes *a* ∈ «ℕ» **and** *b* ∈ «ℕ»
shows *a* *$_{ow}$ *b* ∈ «ℕ»
is *semiring-1-class.Nats-mult*⟨*proof*⟩

tts-lemma *of-nat-1*: *of-nat* 1 = 1_{ow}
is *semiring-1-class.of-nat-1*⟨*proof*⟩

tts-lemma *of-nat-0*: *of-nat* 0 = 0_{ow}
is *semiring-1-class.of-nat-0*⟨*proof*⟩

tts-lemma *Nats-add*:

assumes $a \in$ «$\mathbb{N}$» **and** $b \in$ «$\mathbb{N}$»
shows $a +_{ow} b \in$ «$\mathbb{N}$»
is *semiring-1-class.Nats-add*⟨*proof*⟩

tts-lemma *Nats-1*: $1_{ow} \in$ «$\mathbb{N}$»
is *semiring-1-class.Nats-1*⟨*proof*⟩

tts-lemma *Nats-0*: $0_{ow} \in$ «$\mathbb{N}$»
is *semiring-1-class.Nats-0*⟨*proof*⟩

end

end

3.10.9 Commutative rigs

Definitions and common properties

locale *comm-semiring-1-ow* =
zero-neq-one-ow U one zero +
comm-semiring-0-ow U plus zero times +
comm-monoid-mult-ow U times one
for *U* :: *'ag set* **and** *times one plus zero*
begin

sublocale *semiring-1-ow* ⟨*proof*⟩

end

lemma *comm-semiring-1-ow*: *class.comm-semiring-1* = *comm-semiring-1-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring-1-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-semiring-1-ow* (*Collect* (*Domainp A*))) *class.comm-semiring-1*
⟨*proof*⟩

end

Relativization

context *comm-semiring-1-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *comm-semiring-1-ow-axioms* **and** *zero.not-empty*
applying [*OF times-closed' one-closed plus-closed' zero-closed*]
begin

tts-lemma *semiring-normalization-rules*:
shows
$[\![a \in U;\ m \in U;\ b \in U]\!] \Longrightarrow a *_{ow} m +_{ow} b *_{ow} m = (a +_{ow} b) *_{ow} m$
$[\![a \in U;\ m \in U]\!] \Longrightarrow a *_{ow} m +_{ow} m = (a +_{ow} 1_{ow}) *_{ow} m$
$[\![m \in U;\ a \in U]\!] \Longrightarrow m +_{ow} a *_{ow} m = (a +_{ow} 1_{ow}) *_{ow} m$
$m \in U \Longrightarrow m +_{ow} m = (1_{ow} +_{ow} 1_{ow}) *_{ow} m$
$a \in U \Longrightarrow 0_{ow} +_{ow} a = a$
$a \in U \Longrightarrow a +_{ow} 0_{ow} = a$
$[\![a \in U;\ b \in U]\!] \Longrightarrow a *_{ow} b = b *_{ow} a$
$[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow (a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$
$a \in U \Longrightarrow 0_{ow} *_{ow} a = 0_{ow}$
$a \in U \Longrightarrow a *_{ow} 0_{ow} = 0_{ow}$
$a \in U \Longrightarrow 1_{ow} *_{ow} a = a$
$a \in U \Longrightarrow a *_{ow} 1_{ow} = a$
$[\![lx \in U;\ ly \in U;\ rx \in U;\ ry \in U]\!] \Longrightarrow$
$lx *_{ow} ly *_{ow} (rx *_{ow} ry) = lx *_{ow} rx *_{ow} (ly *_{ow} ry)$
$[\![lx \in U;\ ly \in U;\ rx \in U;\ ry \in U]\!] \Longrightarrow$
$lx *_{ow} ly *_{ow} (rx *_{ow} ry) = lx *_{ow} (ly *_{ow} (rx *_{ow} ry))$
$[\![lx \in U;\ ly \in U;\ rx \in U;\ ry \in U]\!] \Longrightarrow$
$lx *_{ow} ly *_{ow} (rx *_{ow} ry) = rx *_{ow} (lx *_{ow} ly *_{ow} ry)$
$[\![lx \in U;\ ly \in U;\ rx \in U]\!] \Longrightarrow lx *_{ow} ly *_{ow} rx = lx *_{ow} rx *_{ow} ly$
$[\![lx \in U;\ ly \in U;\ rx \in U]\!] \Longrightarrow lx *_{ow} ly *_{ow} rx = lx *_{ow} (ly *_{ow} rx)$
$[\![lx \in U;\ rx \in U;\ ry \in U]\!] \Longrightarrow lx *_{ow} (rx *_{ow} ry) = lx *_{ow} rx *_{ow} ry$
$[\![lx \in U;\ rx \in U;\ ry \in U]\!] \Longrightarrow lx *_{ow} (rx *_{ow} ry) = rx *_{ow} (lx *_{ow} ry)$
$[\![a \in U;\ b \in U;\ c \in U;\ d \in U]\!] \Longrightarrow$
$a +_{ow} b +_{ow} (c +_{ow} d) = a +_{ow} c +_{ow} (b +_{ow} d)$
$[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} (b +_{ow} c)$
$[\![a \in U;\ c \in U;\ d \in U]\!] \Longrightarrow a +_{ow} (c +_{ow} d) = c +_{ow} (a +_{ow} d)$
$[\![a \in U;\ b \in U;\ c \in U]\!] \Longrightarrow a +_{ow} b +_{ow} c = a +_{ow} c +_{ow} b$
$[\![a \in U;\ c \in U]\!] \Longrightarrow a +_{ow} c = c +_{ow} a$
$[\![a \in U;\ c \in U;\ d \in U]\!] \Longrightarrow a +_{ow} (c +_{ow} d) = a +_{ow} c +_{ow} d$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} p *_{ow} x \mathbin{\widehat{}_{ow}} q = x \mathbin{\widehat{}_{ow}} (p + q)$
$x \in U \Longrightarrow x *_{ow} x \mathbin{\widehat{}_{ow}} q = x \mathbin{\widehat{}_{ow}} Suc\ q$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} q *_{ow} x = x \mathbin{\widehat{}_{ow}} Suc\ q$
$x \in U \Longrightarrow x *_{ow} x = x \mathbin{\widehat{}_{ow}} 2$
$[\![x \in U;\ y \in U]\!] \Longrightarrow (x *_{ow} y) \mathbin{\widehat{}_{ow}} q = x \mathbin{\widehat{}_{ow}} q *_{ow} y \mathbin{\widehat{}_{ow}} q$
$x \in U \Longrightarrow (x \mathbin{\widehat{}_{ow}} p) \mathbin{\widehat{}_{ow}} q = x \mathbin{\widehat{}_{ow}} (p * q)$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} 0 = 1_{ow}$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} 1 = x$
$[\![x \in U;\ y \in U;\ z \in U]\!] \Longrightarrow x *_{ow} (y +_{ow} z) = x *_{ow} y +_{ow} x *_{ow} z$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} Suc\ q = x *_{ow} x \mathbin{\widehat{}_{ow}} q$
$x \in U \Longrightarrow x \mathbin{\widehat{}_{ow}} (2 * n) = x \mathbin{\widehat{}_{ow}} n *_{ow} x \mathbin{\widehat{}_{ow}} n$
is *comm-semiring-1-class.semiring-normalization-rules*⟨*proof*⟩

tts-lemma *le-imp-power-dvd*:
assumes $a \in U$ **and** $m \leq n$
shows $a \mathbin{\widehat{}_{ow}} m$ «*dvd*» $a \mathbin{\widehat{}_{ow}} n$
is *comm-semiring-1-class.le-imp-power-dvd*⟨*proof*⟩

tts-lemma *dvd-0-left-iff*:
assumes $a \in U$
shows $(0_{ow}$ «*dvd*» $a) = (a = 0_{ow})$
is *comm-semiring-1-class.dvd-0-left-iff*⟨*proof*⟩

tts-lemma *dvd-power-same*:
assumes $x \in U$ **and** $y \in U$ **and** x «*dvd*» y
shows $x \mathbin{\widehat{}_{ow}} n$ «*dvd*» $y \mathbin{\widehat{}_{ow}} n$
is *comm-semiring-1-class.dvd-power-same*⟨*proof*⟩

tts-lemma *power-le-dvd*:
assumes $a \in U$ **and** $b \in U$ **and** $a \mathbin{\hat{}}_{ow} n$ «*dvd*» b **and** $m \leq n$
shows $a \mathbin{\hat{}}_{ow} m$ «*dvd*» b
is *comm-semiring-1-class.power-le-dvd*⟨*proof*⟩

tts-lemma *dvd-0-right*:
assumes $a \in U$
shows a «*dvd*» 0_{ow}
is *comm-semiring-1-class.dvd-0-right*⟨*proof*⟩

tts-lemma *dvd-0-left*:
assumes $a \in U$ **and** 0_{ow} «*dvd*» a
shows $a = 0_{ow}$
is *comm-semiring-1-class.dvd-0-left*⟨*proof*⟩

tts-lemma *dvd-power*:
assumes $x \in U$ **and** $0 < n \vee x = 1_{ow}$
shows x «*dvd*» $x \mathbin{\hat{}}_{ow} n$
is *comm-semiring-1-class.dvd-power*⟨*proof*⟩

tts-lemma *dvd-add*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** a «*dvd*» b **and** a «*dvd*» c
shows a «*dvd*» $b +_{ow} c$
is *comm-semiring-1-class.dvd-add*⟨*proof*⟩

end

end

3.10.10 Cancellative rigs

Definitions and common properties

locale *semiring-1-cancel-ow* =
semiring-ow U plus times +
cancel-comm-monoid-add-ow U plus minus zero +
zero-neq-one-ow U one zero +
monoid-mult-ow U one times
for $U :: 'ag\ set$ **and** *plus minus zero one times*
begin

sublocale *semiring-0-cancel-ow* ⟨*proof*⟩
sublocale *semiring-1-ow* ⟨*proof*⟩

end

lemma *semiring-1-cancel-ow*:
class.semiring-1-cancel = *semiring-1-cancel-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semiring-1-cancel-transfer*[*transfer-rule*]:
includes *lifting-syntax*

assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(=)
) (*semiring*-1-*cancel-ow* (*Collect* (*Domainp A*))) *class.semiring*-1-*cancel*
⟨*proof*⟩

end

3.10.11 Commutative cancellative rigs

Definitions and common properties

locale *comm-semiring*-1-*cancel-ow* =
comm-semiring-ow U plus times +
cancel-comm-monoid-add-ow U plus minus zero +
zero-neq-one-ow U one zero +
comm-monoid-mult-ow U times one
for *U* :: *'ag set* **and** *plus minus zero times one* +
assumes *right-diff-distrib'*[*algebra-simps*]:
⟦ $a \in U$; $b \in U$; $c \in U$ ⟧ ⟹ $a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$
begin

sublocale *semiring*-1-*cancel-ow* ⟨*proof*⟩
sublocale *comm-semiring*-0-*cancel-ow* ⟨*proof*⟩
sublocale *comm-semiring*-1-*ow* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-semiring*-1-*cancel-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(=)
)
(*comm-semiring*-1-*cancel-ow* (*Collect* (*Domainp A*)))
class.comm-semiring-1-*cancel*
⟨*proof*⟩

end

Relativization

context *comm-semiring-1-cancel-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *comm-semiring-1-cancel-ow-axioms* **and** *zero.not-empty*
applying [*OF plus-closed' minus-closed' zero-closed times-closed' one-closed*]
begin

tts-lemma *dvd-add-times-triv-right-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a \mathbin{«dvd»} b +_{ow} c *_{ow} a) = (a \mathbin{«dvd»} b)$
is *comm-semiring-1-cancel-class.dvd-add-times-triv-right-iff*⟨*proof*⟩

tts-lemma *dvd-add-times-triv-left-iff*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $(a \mathbin{«dvd»} c *_{ow} a +_{ow} b) = (a \mathbin{«dvd»} b)$
is *comm-semiring-1-cancel-class.dvd-add-times-triv-left-iff*⟨*proof*⟩

tts-lemma *dvd-add-triv-right-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(a \mathbin{«dvd»} b +_{ow} a) = (a \mathbin{«dvd»} b)$
is *comm-semiring-1-cancel-class.dvd-add-triv-right-iff*⟨*proof*⟩

tts-lemma *dvd-add-triv-left-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(a \mathbin{«dvd»} a +_{ow} b) = (a \mathbin{«dvd»} b)$
is *comm-semiring-1-cancel-class.dvd-add-triv-left-iff*⟨*proof*⟩

tts-lemma *left-diff-distrib'*:
assumes $b \in U$ **and** $c \in U$ **and** $a \in U$
shows $(b -_{ow} c) *_{ow} a = b *_{ow} a -_{ow} c *_{ow} a$
is *comm-semiring-1-cancel-class.left-diff-distrib'*⟨*proof*⟩

tts-lemma *dvd-add-right-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a \mathbin{«dvd»} b$
shows $(a \mathbin{«dvd»} b +_{ow} c) = (a \mathbin{«dvd»} c)$
is *comm-semiring-1-cancel-class.dvd-add-right-iff*⟨*proof*⟩

tts-lemma *dvd-add-left-iff*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a \mathbin{«dvd»} c$
shows $(a \mathbin{«dvd»} b +_{ow} c) = (a \mathbin{«dvd»} b)$
is *comm-semiring-1-cancel-class.dvd-add-left-iff*⟨*proof*⟩

end

end

3.11 Relativization of the results about rings

3.11.1 Rings

Definitions and common properties

locale *ring-ow* =
semiring-ow U plus times + *ab-group-add-ow U plus zero minus uminus*
for *U* :: *'ag set* **and** *plus zero minus uminus times*
begin

sublocale *semiring*-0-*cancel-ow* ⟨*proof*⟩

end

lemma *ring-ow*: *class.ring* = *ring-ow UNIV*
⟨*proof*⟩

lemma *ring-ow-UNIV-axioms*: *ring-ow* (*UNIV*::*'a*::*ring set*) (+) 0 (−) *uminus* (∗)
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *ring-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
(=)
)
(*ring-ow* (*Collect* (*Domainp A*))) *class.ring*
⟨*proof*⟩

end

Relativization

context *ring-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ring-ow-axioms* **and** *zero.not-empty*
applying
[
OF
plus-closed'
zero-closed
minus-closed'
add.inverse-closed''

times-closed′
]
begin

tts-lemma *right-diff-distrib*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$
is *Rings.ring-class.right-diff-distrib*⟨*proof*⟩

tts-lemma *minus-mult-commute*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} a *_{ow} b = a *_{ow} -_{ow} b$
is *Rings.ring-class.minus-mult-commute*⟨*proof*⟩

tts-lemma *left-diff-distrib*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a -_{ow} b) *_{ow} c = a *_{ow} c -_{ow} b *_{ow} c$
is *Rings.ring-class.left-diff-distrib*⟨*proof*⟩

tts-lemma *mult-minus-right*:
assumes $a \in U$ **and** $b \in U$
shows $a *_{ow} -_{ow} b = -_{ow} (a *_{ow} b)$
is *Rings.ring-class.mult-minus-right*⟨*proof*⟩

tts-lemma *minus-mult-right*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} (a *_{ow} b) = a *_{ow} -_{ow} b$
is *Rings.ring-class.minus-mult-right*⟨*proof*⟩

tts-lemma *minus-mult-minus*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} a *_{ow} -_{ow} b = a *_{ow} b$
is *Rings.ring-class.minus-mult-minus*⟨*proof*⟩

tts-lemma *mult-minus-left*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} a *_{ow} b = -_{ow} (a *_{ow} b)$
is *Rings.ring-class.mult-minus-left*⟨*proof*⟩

tts-lemma *minus-mult-left*:
assumes $a \in U$ **and** $b \in U$
shows $-_{ow} (a *_{ow} b) = -_{ow} a *_{ow} b$
is *Rings.ring-class.minus-mult-left*⟨*proof*⟩

tts-lemma *ring-distribs*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows
$a *_{ow} (b +_{ow} c) = a *_{ow} b +_{ow} a *_{ow} c$
$(a +_{ow} b) *_{ow} c = a *_{ow} c +_{ow} b *_{ow} c$
$(a -_{ow} b) *_{ow} c = a *_{ow} c -_{ow} b *_{ow} c$
$a *_{ow} (b -_{ow} c) = a *_{ow} b -_{ow} a *_{ow} c$
is *Rings.ring-class.ring-distribs*⟨*proof*⟩

tts-lemma *eq-add-iff2*:
assumes $a \in U$ **and** $e \in U$ **and** $c \in U$ **and** $b \in U$ **and** $d \in U$
shows $(a *_{ow} e +_{ow} c = b *_{ow} e +_{ow} d) = (c = (b -_{ow} a) *_{ow} e +_{ow} d)$
is *Rings.ring-class.eq-add-iff2*⟨*proof*⟩

tts-lemma *eq-add-iff1*:
assumes $a \in U$ **and** $e \in U$ **and** $c \in U$ **and** $b \in U$ **and** $d \in U$
shows $(a *_{ow} e +_{ow} c = b *_{ow} e +_{ow} d) = ((a -_{ow} b) *_{ow} e +_{ow} c = d)$
is *Rings.ring-class.eq-add-iff1*⟨*proof*⟩

tts-lemma *mult-diff-mult*:
assumes $x \in U$ **and** $y \in U$ **and** $a \in U$ **and** $b \in U$
shows $x *_{ow} y -_{ow} a *_{ow} b = x *_{ow} (y -_{ow} b) +_{ow} (x -_{ow} a) *_{ow} b$
is *Real.mult-diff-mult*⟨*proof*⟩

end

end

3.11.2 Commutative rings

Definitions and common properties

locale *comm-ring-ow* =
comm-semiring-ow U plus times + *ab-group-add-ow U plus zero minus uminus*
for U :: *'ag set* **and** *plus zero minus uminus times*
begin

sublocale *ring-ow* ⟨*proof*⟩
sublocale *comm-semiring-0-cancel-ow* ⟨*proof*⟩

end

lemma *comm-ring-ow*: *class.comm-ring* = *comm-ring-ow UNIV*
⟨*proof*⟩

Transfer rules

context
includes *lifting-syntax*
begin

lemma *comm-ring-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
(=)
)
(*comm-ring-ow* (*Collect* (*Domainp A*))) *class.comm-ring*
⟨*proof*⟩

end

Relativization

context *comm-ring-ow*
begin

tts-context

tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *comm-ring-ow-axioms* **and** *zero.not-empty*
applying
[
OF
plus-closed'
zero-closed
minus-closed'
add.inverse-closed''
times-closed'
]
begin

tts-lemma *square-diff-square-factored*:
assumes $x \in U$ **and** $y \in U$
shows $x *_{ow} x -_{ow} y *_{ow} y = (x +_{ow} y) *_{ow} (x -_{ow} y)$
is *comm-ring-class.square-diff-square-factored*⟨*proof*⟩

end

end

3.11.3 Rings with identity

Definitions and common properties

locale *ring*-1-*ow* =
ring-ow U plus zero minus uminus times +
zero-neq-one-ow U one zero +
monoid-mult-ow U one times
for *U* :: *'ag set* **and** *one times plus zero minus uminus*
begin

sublocale *semiring*-1-*cancel-ow* ⟨*proof*⟩

end

lemma *ring*-1-*ow*: *class.ring*-1 = *ring*-1-*ow UNIV*
⟨*proof*⟩

lemma *ring*-1-*ow-UNIV-axioms*:
ring-1-*ow* (*UNIV*::*'a*::*ring*-1 *set*) 1 (∗) (+) 0 (−) *uminus*
⟨*proof*⟩

ud ‹*ring*-1.*iszero*› (‹(*with* - : «*iszero*» -)› [1000, 1000] 10)
ud *iszero'* ‹*iszero*›
ud ‹*ring*-1.*of-int*›
(‹(*with* - - - - : «*of'-int*» -)› [1000, 999, 998, 997, 1000] 10)
ud *of-int'* ‹*of-int*›
ud ‹*ring*-1.*Ints*› (‹(*with* - - - - : ℤ)› [1000, 999, 998, 997] 10)
ud *Ints'* ‹*Ints*›
ud ‹*diffs*› (‹(*with* - - - - : «*diffs*» -)› [1000, 999, 998, 997, 1000] 10)

ctr parametricity
in *iszero-ow*: *iszero.with-def*
and *of-int-ow*: *of-int.with-def*
and *Ints-ow*: *Ints.with-def*

and *diffs-ow*: *diffs.with-def*

context *ring*-1-*ow*
begin

abbreviation *iszero* **where** *iszero* ≡ *iszero.with* 0_{ow}
abbreviation *of-int* **where** *of-int* ≡ *of-int.with* 1_{ow} $(+_{ow})$ 0_{ow} $(-_{ow})$
abbreviation *Ints* (‹«ℤ»›) **where** «ℤ» ≡ *Ints.with* 1_{ow} $(+_{ow})$ 0_{ow} $(-_{ow})$
notation *Ints* (‹«ℤ»›)

end

context *ring*-1
begin

lemma *Int-ss-UNIV*: ℤ ⊆ *UNIV* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *ring*-1-*transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
A ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A*) ===>
(=)
)
(*ring*-1-*ow* (*Collect* (*Domainp A*))) *class.ring*-1
⟨*proof*⟩

end

Relativization

declare *dvd.with*[*ud-with del*]
declare *dvd′.with*[*ud-with del*]

context *ring*-1-*ow*
begin

tts-context
tts: (*?′a* **to** *U*)
substituting *ring*-1-*ow-axioms* **and** *zero.not-empty*
eliminating through *simp*
begin

tts-lemma *Int-ss-UNIV*[*simp*]: «ℤ» ⊆ *U*
is *Int-ss-UNIV*⟨*proof*⟩

end

lemma *Int-closed*[*simp*,*intro*]: $a \in «\mathbb{Z}» \Longrightarrow a \in U$ ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *ring-1-ow-axioms* **and** *zero.not-empty*
eliminating through *auto*
begin

tts-lemma *iszero-0*: *iszero* 0_{ow}
is *ring-1-class.iszero-0*⟨*proof*⟩

tts-lemma *not-iszero-1*: ¬ *iszero* 1_{ow}
is *ring-1-class.not-iszero-1*⟨*proof*⟩

tts-lemma *Nats-subset-Ints*: «$\mathbb{N}$» ⊆ «$\mathbb{Z}$»
is *Int.ring-1-class.Nats-subset-Ints*⟨*proof*⟩

tts-lemma *Ints-1*: $1_{ow} \in «\mathbb{Z}»$
is *Int.ring-1-class.Ints-1*⟨*proof*⟩

tts-lemma *Ints-0*: $0_{ow} \in «\mathbb{Z}»$
is *Int.ring-1-class.Ints-0*⟨*proof*⟩

tts-lemma *not-iszero-neg-1*: ¬ *iszero* $(-_{ow}\ 1_{ow})$
is *Num.ring-1-class.not-iszero-neg-1*⟨*proof*⟩

tts-lemma *of-int-1*: *of-int* 1 = 1_{ow}
is *Int.ring-1-class.of-int-1*⟨*proof*⟩

tts-lemma *of-int-0*: *of-int* 0 = 0_{ow}
is *Int.ring-1-class.of-int-0*⟨*proof*⟩

tts-lemma *Ints-of-int*: *of-int z* ∈ «$\mathbb{Z}$»
is *Int.ring-1-class.Ints-of-int*⟨*proof*⟩

tts-lemma *Ints-of-nat*: *of-nat n* ∈ «$\mathbb{Z}$»
is *Int.ring-1-class.Ints-of-nat*⟨*proof*⟩

tts-lemma *of-int-of-nat-eq*:
shows *local.of-int* (*with* 1 (+) 0 : «*of-nat*» *n*) = *local.of-nat n*
is *Int.ring-1-class.of-int-of-nat-eq*⟨*proof*⟩

tts-lemma *of-int-of-bool*:
of-int (*with* 1 0 : «*of-bool*» *P*) = *of-bool P*
is *Int.ring-1-class.of-int-of-bool*⟨*proof*⟩

tts-lemma *of-int-minus*: *of-int* (− *z*) = $-_{ow}$ *of-int z*
is *Int.ring-1-class.of-int-minus*⟨*proof*⟩

tts-lemma *mult-minus1-right*:
assumes *z* ∈ *U*
shows $z *_{ow} -_{ow} 1_{ow} = -_{ow} z$
is *Num.ring-1-class.mult-minus1-right*⟨*proof*⟩

tts-lemma *mult-minus*1:
assumes $z \in U$
shows $-_{ow}\ 1_{ow} *_{ow} z = -_{ow}\ z$
is *Num.ring*-1-*class.mult-minus*1⟨*proof*⟩

tts-lemma *eq-iff-iszero-diff*:
assumes $x \in U$ **and** $y \in U$
shows $(x = y) = iszero\ (x -_{ow} y)$
is *Num.ring*-1-*class.eq-iff-iszero-diff*⟨*proof*⟩

tts-lemma *minus-in-Ints-iff*:
assumes $x \in U$
shows $(-_{ow}\ x \in «\mathbb{Z}») = (x \in «\mathbb{Z}»)$
is *Int.ring*-1-*class.minus-in-Ints-iff*⟨*proof*⟩

tts-lemma *mult-of-int-commute*:
assumes $y \in U$
shows *of-int* $x *_{ow} y = y *_{ow}$ *of-int* x
is *Int.ring*-1-*class.mult-of-int-commute*⟨*proof*⟩

tts-lemma *of-int-power*:
of-int $((with\ 1\ (*) : z\ \widehat{}_{ow}\ n)) =$ *of-int* $z\ \widehat{}_{ow}\ n$
is *Int.ring*-1-*class.of-int-power*⟨*proof*⟩

tts-lemma *Ints-minus*:
assumes $a \in «\mathbb{Z}»$
shows $-_{ow}\ a \in «\mathbb{Z}»$
is *Int.ring*-1-*class.Ints-minus*⟨*proof*⟩

tts-lemma *of-int-diff*: *of-int* $(w - z) =$ *of-int* $w -_{ow}$ *of-int* z
is *Int.ring*-1-*class.of-int-diff*⟨*proof*⟩

tts-lemma *of-int-add*: *of-int* $(w + z) =$ *of-int* $w +_{ow}$ *of-int* z
is *Int.ring*-1-*class.of-int-add*⟨*proof*⟩

tts-lemma *of-int-mult*: *of-int* $(w * z) =$ *of-int* $w *_{ow}$ *of-int* z
is *Int.ring*-1-*class.of-int-mult*⟨*proof*⟩

tts-lemma *power-minus*1-*even*: $(-_{ow}\ 1_{ow})\ \widehat{}_{ow}\ (2 * n) = 1_{ow}$
is *Power.ring*-1-*class.power-minus*1-*even*⟨*proof*⟩

tts-lemma *Ints-power*:
assumes $a \in «\mathbb{Z}»$
shows $a\ \widehat{}_{ow}\ n \in «\mathbb{Z}»$
is *Int.ring*-1-*class.Ints-power*⟨*proof*⟩

tts-lemma *of-nat-nat*:
assumes $0 \leq z$
shows *of-nat* $(nat\ z) =$ *of-int* z
is *Int.ring*-1-*class.of-nat-nat*⟨*proof*⟩

tts-lemma *power*2-*minus*:
assumes $a \in U$
shows $(-_{ow}\ a)\ \widehat{}_{ow}\ 2 = a\ \widehat{}_{ow}\ 2$
is *Power.ring*-1-*class.power*2-*minus*⟨*proof*⟩

tts-lemma *power-minus*1-*odd*:
shows $(-_{ow}\ 1_{ow})\ \widehat{}_{ow}\ Suc\ (2 * n) = -_{ow}\ 1_{ow}$

is *Power.ring-1-class.power-minus1-odd⟨proof⟩*

tts-lemma *power-minus*:
assumes $a \in U$
shows $(-_{ow}\ a)\ \hat{}_{ow}\ n = (-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n *_{ow} a\ \hat{}_{ow}\ n$
is *Power.ring-1-class.power-minus⟨proof⟩*

tts-lemma *square-diff-one-factored*:
assumes $x \in U$
shows $x *_{ow} x -_{ow} 1_{ow} = (x +_{ow} 1_{ow}) *_{ow} (x -_{ow} 1_{ow})$
is *Rings.ring-1-class.square-diff-one-factored⟨proof⟩*

tts-lemma *neg-one-even-power*:
assumes *even n*
shows $(-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n = 1_{ow}$
is *Parity.ring-1-class.neg-one-even-power⟨proof⟩*

tts-lemma *minus-one-power-iff*:
$(-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n = (\textit{if even } n \textit{ then } 1_{ow} \textit{ else } -_{ow}\ 1_{ow})$
is *Parity.ring-1-class.minus-one-power-iff⟨proof⟩*

tts-lemma *Nats-altdef1*: «$\mathbb{N}$» = $\{x \in U.\ \exists\, y{\geq}0.\ x = \textit{of-int } y\}$
is *Int.ring-1-class.Nats-altdef1⟨proof⟩*

tts-lemma *Ints-induct*:
assumes $q \in$ «$\mathbb{Z}$» **and** $\bigwedge z.\ P\ (\textit{of-int } z)$
shows $P\ q$
is *Int.ring-1-class.Ints-induct⟨proof⟩*

tts-lemma *of-int-of-nat*:
shows
$\textit{of-int } k = (\textit{if } k < 0 \textit{ then } -_{ow}\ \textit{of-nat } (\textit{nat } (-\ k)) \textit{ else of-nat } (\textit{nat } k))$
is *Int.ring-1-class.of-int-of-nat⟨proof⟩*

tts-lemma *Ints-diff*:
assumes $a \in$ «$\mathbb{Z}$» **and** $b \in$ «$\mathbb{Z}$»
shows $a -_{ow} b \in$ «$\mathbb{Z}$»
is *Int.ring-1-class.Ints-diff⟨proof⟩*

tts-lemma *Ints-add*:
assumes $a \in$ «$\mathbb{Z}$» **and** $b \in$ «$\mathbb{Z}$»
shows $a +_{ow} b \in$ «$\mathbb{Z}$»
is *Int.ring-1-class.Ints-add⟨proof⟩*

tts-lemma *Ints-mult*:
assumes $a \in$ «$\mathbb{Z}$» **and** $b \in$ «$\mathbb{Z}$»
shows $a *_{ow} b \in$ «$\mathbb{Z}$»
is *Int.ring-1-class.Ints-mult⟨proof⟩*

tts-lemma *power-minus-even′*:
assumes $a \in U$ **and** *even n*
shows $(-_{ow}\ a)\ \hat{}_{ow}\ n = a\ \hat{}_{ow}\ n$
is *Parity.ring-1-class.power-minus-even⟨proof⟩*

tts-lemma *power-minus-even*:
assumes $a \in U$
shows $(-_{ow}\ a)\ \hat{}_{ow}\ (2 * n) = a\ \hat{}_{ow}\ (2 * n)$
is *Power.ring-1-class.power-minus-even⟨proof⟩*

tts-lemma *power-minus-odd*:
assumes $a \in U$ **and** *odd* n
shows $(-_{ow}\ a)\ \widehat{}_{ow}\ n = -_{ow}\ (a\ \widehat{}_{ow}\ n)$
is *Parity.ring-1-class.power-minus-odd*⟨*proof*⟩

tts-lemma *uminus-power-if*:
assumes $a \in U$
shows $(-_{ow}\ a)\ \widehat{}_{ow}\ n = (\textit{if even } n \textit{ then } a\ \widehat{}_{ow}\ n \textit{ else } -_{ow}\ (a\ \widehat{}_{ow}\ n))$
is *Parity.ring-1-class.uminus-power-if*⟨*proof*⟩

tts-lemma *neg-one-power-add-eq-neg-one-power-diff*:
assumes $k \leq n$
shows $(-_{ow}\ 1_{ow})\ \widehat{}_{ow}\ (n + k) = (-_{ow}\ 1_{ow})\ \widehat{}_{ow}\ (n - k)$
is *Parity.ring-1-class.neg-one-power-add-eq-neg-one-power-diff*⟨*proof*⟩

tts-lemma *neg-one-odd-power*:
assumes *odd* n
shows $(-_{ow}\ 1_{ow})\ \widehat{}_{ow}\ n = -_{ow}\ 1_{ow}$
is *Parity.ring-1-class.neg-one-odd-power*⟨*proof*⟩

tts-lemma *Ints-cases*:
assumes $q \in$ «$\mathbb{Z}$» **and** $\bigwedge z.\ q = \textit{of-int } z \Longrightarrow \textit{thesis}$
shows *thesis*
is *Int.ring-1-class.Ints-cases*⟨*proof*⟩

end

end

lemmas [*ud-with*] = *dvd.with dvd'.with*

3.11.4 Commutative rings with identity

Definitions and common properties

locale *comm-ring-1-ow* =
comm-ring-ow U plus zero minus uminus times +
zero-neq-one-ow U one zero +
comm-monoid-mult-ow U times one
for $U :: 'ag\ set$ **and** *times one plus zero minus uminus*
begin

sublocale *ring-1-ow* ⟨*proof*⟩
sublocale *comm-semiring-1-cancel-ow*
⟨*proof*⟩

end

lemma *comm-ring-1-ow*: *class.comm-ring-1* = *comm-ring-1-ow UNIV*
⟨*proof*⟩

lemma *comm-ring-1-ow-UNIV-axioms*:
comm-ring-1-ow (*UNIV*::'*a*::*comm-ring-1 set*) (∗) 1 (+) 0 (−) *uminus*
⟨*proof*⟩

Transfer rules

context

includes *lifting-syntax*
begin

lemma *comm-ring-1-transfer*[*transfer-rule*]:
assumes[*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A*) ===>
(=)
) (*comm-ring-1-ow* (*Collect* (*Domainp A*))) *class.comm-ring-1*
⟨*proof*⟩

end

Relativization

context *comm-ring-1-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *comm-ring-1-ow-axioms* **and** *zero.not-empty*
applying
[
OF
times-closed'
one-closed
plus-closed'
zero-closed
minus-closed'
add.inverse-closed''
]
begin

tts-lemma *ring-normalization-rules*:
assumes $x \in U$
shows $-_{ow}\ x = -_{ow}\ 1_{ow} *_{ow} x\ y \in U \Longrightarrow x -_{ow} y = x +_{ow} -_{ow} y$
is *comm-ring-1-class.ring-normalization-rules*⟨*proof*⟩

tts-lemma *left-minus-one-mult-self*:
assumes $a \in U$
shows $(-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n *_{ow} ((-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n *_{ow} a) = a$
is *Power.comm-ring-1-class.left-minus-one-mult-self*⟨*proof*⟩

tts-lemma *minus-power-mult-self*:
assumes $a \in U$
shows $(-_{ow}\ a)\ \hat{}_{ow}\ n *_{ow} (-_{ow}\ a)\ \hat{}_{ow}\ n = a\ \hat{}_{ow}\ (2 * n)$
is *Power.comm-ring-1-class.minus-power-mult-self*⟨*proof*⟩

tts-lemma *minus-one-mult-self*: $(-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n *_{ow} (-_{ow}\ 1_{ow})\ \hat{}_{ow}\ n = 1_{ow}$
is *comm-ring-1-class.minus-one-mult-self*⟨*proof*⟩

tts-lemma *power2-commute*:
assumes $x \in U$ **and** $y \in U$
shows $(x -_{ow} y) \hat{}_{ow} 2 = (y -_{ow} x) \hat{}_{ow} 2$
is *comm-ring-1-class.power2-commute*⟨*proof*⟩

tts-lemma *minus-dvd-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(-_{ow} x \mathbin{«dvd»} y) = (x \mathbin{«dvd»} y)$
is *comm-ring-1-class.minus-dvd-iff*⟨*proof*⟩

tts-lemma *dvd-minus-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(x \mathbin{«dvd»} -_{ow} y) = (x \mathbin{«dvd»} y)$
is *comm-ring-1-class.dvd-minus-iff*⟨*proof*⟩

tts-lemma *dvd-diff*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x \mathbin{«dvd»} y$ **and** $x \mathbin{«dvd»} z$
shows $x \mathbin{«dvd»} y -_{ow} z$
is *comm-ring-1-class.dvd-diff*⟨*proof*⟩

end

end

3.12 Relativization of the results about semilattices

3.12.1 Commutative bands

Definitions and common properties

locale *semilattice-ow = abel-semigroup-ow U f*
  for *U :: 'al set* **and** *f +*
  assumes *idem[simp]*: $x \in U \Longrightarrow x *_{ow} x = x$

locale *semilattice-set-ow =*
  semilattice-ow U f **for** *U :: 'al set* **and** *f* (**infixl** ‹$*_{ow}$› 70)

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *semilattice-transfer[transfer-rule]*:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*A* ===> *A* ===> *A*) ===> (=))
      (λ*f*. *semilattice-ow* (*Collect* (*Domainp A*)) *f*) *semilattice*
  ⟨*proof*⟩

lemma *semilattice-set-transfer[transfer-rule]*:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*A* ===> *A* ===> *A*) ===> (=))
      (λ*f*. *semilattice-set-ow* (*Collect* (*Domainp A*)) *f*) *semilattice-set*
  ⟨*proof*⟩

end

Relativization

context *semilattice-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  substituting *semilattice-ow-axioms*
  eliminating through *simp*
begin

tts-lemma *left-idem*:
  assumes $a \in U$ **and** $b \in U$
  shows $a *_{ow} (a *_{ow} b) = a *_{ow} b$
    is *semilattice.left-idem*⟨*proof*⟩

tts-lemma *right-idem*:
  assumes $a \in U$ **and** $b \in U$
  shows $a *_{ow} b *_{ow} b = a *_{ow} b$
    is *semilattice.right-idem*⟨*proof*⟩

end

end

3.12.2 Simple upper and lower semilattices

Definitions and common properties

locale *semilattice-order-ow* = *semilattice-ow U f*
  for U :: *'al set* **and** f +
  fixes *le* :: ['*al*, '*al*] ⇒ *bool* (**infix** ‹$\leq_{ow}$› 50)
    and *ls* :: ['*al*, '*al*] ⇒ *bool* (**infix** ‹$<_{ow}$› 50)
  assumes *order-iff*: ⟦ $a \in U$; $b \in U$ ⟧ ⟹ $a \leq_{ow} b \longleftrightarrow a = a *_{ow} b$
    and *strict-order-iff*: ⟦ $a \in U$; $b \in U$ ⟧ ⟹ $a <_{ow} b \longleftrightarrow a = a *_{ow} b \wedge a \neq b$
begin

sublocale *ordering-ow U* ‹$(\leq_{ow})$› ‹$(<_{ow})$›
⟨*proof*⟩

notation *le* (**infix** ‹$\leq_{ow}$› 50)
  and *ls* (**infix** ‹$<_{ow}$› 50)

end

locale *semilattice-order-set-ow* =
  semilattice-order-ow U f le ls + *semilattice-set-ow U f*
  for U :: *'al set* **and** *f le ls*

locale *inf-ow* =
  fixes U :: *'al set* **and** *inf* (**infixl** ‹$\sqcap_{ow}$› 70)
  assumes *inf-closed*[*simp*]: ⟦ $x \in U$; $y \in U$ ⟧ ⟹ $x \sqcap_{ow} y \in U$
begin

notation *inf* (**infixl** ‹$\sqcap_{ow}$› 70)

lemma *inf-closed'*[*simp*]: ∀ $x \in U$. ∀ $y \in U$. $x \sqcap_{ow} y \in U$ ⟨*proof*⟩

end

locale *inf-pair-ow* = inf_1: *inf-ow* U_1 inf_1 + inf_2: *inf-ow* U_2 inf_2
  for U_1 :: *'al set* **and** inf_1
    and U_2 :: *'bl set* **and** inf_2
begin

notation inf_1 (**infixl** ‹$\sqcap_{ow.1}$› 70)
notation inf_2 (**infixl** ‹$\sqcap_{ow.2}$› 70)

end

locale *semilattice-inf-ow* = *inf-ow U inf* + *order-ow U le ls*
  for U :: *'al set* **and** *inf le ls* +
  assumes *inf-le1*[*simp*]: ⟦ $x \in U$; $y \in U$ ⟧ ⟹ $x \sqcap_{ow} y \leq_{ow} x$
    and *inf-le2*[*simp*]: ⟦ $x \in U$; $y \in U$ ⟧ ⟹ $x \sqcap_{ow} y \leq_{ow} y$
    and *inf-greatest*:
      ⟦ $x \in U$; $y \in U$; $z \in U$; $x \leq_{ow} y$; $x \leq_{ow} z$ ⟧ ⟹ $x \leq_{ow} y \sqcap_{ow} z$
begin

sublocale *inf*: *semilattice-order-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$›
⟨*proof*⟩

sublocale *Inf-fin*: *semilattice-order-set-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ⟨*proof*⟩

end

locale *semilattice-inf-pair-ow* =
  sl-inf$_1$: *semilattice-inf-ow* U_1 inf_1 le_1 ls_1 +
  sl-inf$_2$: *semilattice-inf-ow* U_2 inf_2 le_2 ls_2
  for U_1 :: $'al$ *set* **and** inf_1 le_1 ls_1
    and U_2 :: $'bl$ *set* **and** inf_2 le_2 ls_2
begin

sublocale *inf-pair-ow* ⟨*proof*⟩
sublocale *order-pair-ow* ⟨*proof*⟩

end

locale *sup-ow* =
  fixes U :: $'ao$ *set* **and** *sup* :: [$'ao$, $'ao$] ⇒ $'ao$ (**infixl** ‹$\sqcup_{ow}$› 70)
  assumes *sup-closed*[*simp*]: ⟦ $x \in U$; $y \in U$ ⟧ ⟹ *sup* x $y \in U$
begin

notation *sup* (**infixl** ‹$\sqcup_{ow}$› 70)

lemma *sup-closed'*[*simp*]: ∀ x∈U. ∀ y∈U. $x \sqcup_{ow} y \in U$ ⟨*proof*⟩

end

locale *sup-pair-ow* = *sup*$_1$: *sup-ow* U_1 sup_1 + *sup*$_2$: *sup-ow* U_2 sup_2
  for U_1 :: $'al$ *set* **and** sup_1
    and U_2 :: $'bl$ *set* **and** sup_2
begin

notation sup_1 (**infixl** ‹$\sqcup_{ow.1}$› 70)
notation sup_2 (**infixl** ‹$\sqcup_{ow.2}$› 70)

end

locale *semilattice-sup-ow* = *sup-ow* U *sup* + *order-ow* U *le ls*
  for U :: $'al$ *set* **and** *sup le ls* +
  assumes *sup-ge*1[*simp*]: ⟦ $x \in U$; $y \in U$ ⟧ ⟹ $x \leq_{ow} x \sqcup_{ow} y$
    and *sup-ge*2[*simp*]: ⟦ $y \in U$; $x \in U$ ⟧ ⟹ $y \leq_{ow} x \sqcup_{ow} y$
    and *sup-least*:
      ⟦ $y \in U$; $x \in U$; $z \in U$; $y \leq_{ow} x$; $z \leq_{ow} x$ ⟧ ⟹ $y \sqcup_{ow} z \leq_{ow} x$
begin

sublocale *sup*: *semilattice-order-ow* U ‹$(\sqcup_{ow})$› ‹$(\geq_{ow})$› ‹$(>_{ow})$›
⟨*proof*⟩

sublocale *Sup-fin*: *semilattice-order-set-ow* U *sup* $(\geq_{ow})$ $(>_{ow})$ ⟨*proof*⟩

end

locale *semilattice-sup-pair-ow* =
  sl-sup$_1$: *semilattice-sup-ow* U_1 sup_1 le_1 ls_1 +
  sl-sup$_2$: *semilattice-sup-ow* U_2 sup_2 le_2 ls_2
  for U_1 :: $'al$ *set* **and** sup_1 le_1 ls_1
    and U_2 :: $'bl$ *set* **and** sup_2 le_2 ls_2
begin

sublocale *sup-pair-ow* ⟨*proof*⟩

sublocale *order-pair-ow* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semilattice-order-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(=)
) (*semilattice-order-ow* (*Collect* (*Domainp A*))) *semilattice-order*
⟨*proof*⟩

lemma *semilattice-order-set-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(=)
) (*semilattice-order-set-ow* (*Collect* (*Domainp A*))) *semilattice-order-set*
⟨*proof*⟩

lemma *semilattice-inf-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(=)
) (*semilattice-inf-ow* (*Collect* (*Domainp A*))) *class.semilattice-inf*
⟨*proof*⟩

lemma *semilattice-sup-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(=)
) (*semilattice-sup-ow* (*Collect* (*Domainp A*))) *class.semilattice-sup*
⟨*proof*⟩

end

Relativization

context *semilattice-order-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *semilattice-order-ow-axioms*
eliminating through *simp*
begin

tts-lemma *cobounded1*:
assumes $a \in U$ **and** $b \in U$
shows $a *_{ow} b \leq_{ow} a$
is *semilattice-order.cobounded1*⟨*proof*⟩

tts-lemma *cobounded2*:
assumes $a \in U$ **and** $b \in U$
shows $a *_{ow} b \leq_{ow} b$
is *semilattice-order.cobounded2*⟨*proof*⟩

tts-lemma *absorb-iff1*:
assumes $a \in U$ **and** $b \in U$
shows $(a \leq_{ow} b) = (a *_{ow} b = a)$
is *semilattice-order.absorb-iff1*⟨*proof*⟩

tts-lemma *absorb-iff2*:
assumes $b \in U$ **and** $a \in U$
shows $(b \leq_{ow} a) = (a *_{ow} b = b)$
is *semilattice-order.absorb-iff2*⟨*proof*⟩

tts-lemma *strict-coboundedI1*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a <_{ow} c$
shows $a *_{ow} b <_{ow} c$
is *semilattice-order.strict-coboundedI1*⟨*proof*⟩

tts-lemma *strict-coboundedI2*:
assumes $b \in U$ **and** $c \in U$ **and** $a \in U$ **and** $b <_{ow} c$
shows $a *_{ow} b <_{ow} c$
is *semilattice-order.strict-coboundedI2*⟨*proof*⟩

tts-lemma *absorb1*:
assumes $a \in U$ **and** $b \in U$ **and** $a \leq_{ow} b$
shows $a *_{ow} b = a$
is *semilattice-order.absorb1*⟨*proof*⟩

tts-lemma *coboundedI1*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$ **and** $a \leq_{ow} c$
shows $a *_{ow} b \leq_{ow} c$
is *semilattice-order.coboundedI1*⟨*proof*⟩

tts-lemma *absorb2*:
assumes $b \in U$ **and** $a \in U$ **and** $b \leq_{ow} a$
shows $a *_{ow} b = b$
is *semilattice-order.absorb2*⟨*proof*⟩

tts-lemma *coboundedI2*:

assumes $b \in U$ **and** $c \in U$ **and** $a \in U$ **and** $b \leq_{ow} c$
shows $a *_{ow} b \leq_{ow} c$
is *semilattice-order.coboundedI2*⟨*proof*⟩

tts-lemma *orderI*:
assumes $a \in U$ **and** $b \in U$ **and** $a = a *_{ow} b$
shows $a \leq_{ow} b$
is *semilattice-order.orderI*⟨*proof*⟩

tts-lemma *bounded-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a \leq_{ow} b *_{ow} c) = (a \leq_{ow} b \wedge a \leq_{ow} c)$
is *semilattice-order.bounded-iff*⟨*proof*⟩

tts-lemma *boundedI*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $a \leq_{ow} b$ **and** $a \leq_{ow} c$
shows $a \leq_{ow} b *_{ow} c$
is *semilattice-order.boundedI*⟨*proof*⟩

tts-lemma *orderE*:
assumes $a \in U$ **and** $b \in U$ **and** $a \leq_{ow} b$ **and** $a = a *_{ow} b \Longrightarrow$ *thesis*
shows *thesis*
is *semilattice-order.orderE*⟨*proof*⟩

tts-lemma *mono*:
assumes $a \in U$
and $c \in U$
and $b \in U$
and $d \in U$
and $a \leq_{ow} c$
and $b \leq_{ow} d$
shows $a *_{ow} b \leq_{ow} c *_{ow} d$
is *semilattice-order.mono*⟨*proof*⟩

tts-lemma *strict-boundedE*:
assumes $a \in U$
and $b \in U$
and $c \in U$
and $a <_{ow} b *_{ow} c$
obtains $a <_{ow} b$ **and** $a <_{ow} c$
given *semilattice-order.strict-boundedE* ⟨*proof*⟩

tts-lemma *boundedE*:
assumes $a \in U$
and $b \in U$
and $c \in U$
and $a \leq_{ow} b *_{ow} c$
obtains $a \leq_{ow} b$ **and** $a \leq_{ow} c$
given *semilattice-order.boundedE* ⟨*proof*⟩

end

end

context *semilattice-inf-ow*
begin

tts-context

tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *semilattice-inf-ow-axioms*
eliminating through *simp*
begin

tts-lemma *le-iff-inf*:
assumes $x \in U$ **and** $y \in U$
shows $(x \leq_{ow} y) = (x \sqcap_{ow} y = x)$
is *semilattice-inf-class.le-iff-inf*⟨*proof*⟩

tts-lemma *less-infI1*:
assumes $a \in U$ **and** $x \in U$ **and** $b \in U$ **and** $a <_{ow} x$
shows $a \sqcap_{ow} b <_{ow} x$
is *semilattice-inf-class.less-infI1*⟨*proof*⟩

tts-lemma *less-infI2*:
assumes $b \in U$ **and** $x \in U$ **and** $a \in U$ **and** $b <_{ow} x$
shows $a \sqcap_{ow} b <_{ow} x$
is *semilattice-inf-class.less-infI2*⟨*proof*⟩

tts-lemma *le-infI1*:
assumes $a \in U$ **and** $x \in U$ **and** $b \in U$ **and** $a \leq_{ow} x$
shows $a \sqcap_{ow} b \leq_{ow} x$
is *semilattice-inf-class.le-infI1*⟨*proof*⟩

tts-lemma *le-infI2*:
assumes $b \in U$ **and** $x \in U$ **and** $a \in U$ **and** $b \leq_{ow} x$
shows $a \sqcap_{ow} b \leq_{ow} x$
is *semilattice-inf-class.le-infI2*⟨*proof*⟩

tts-lemma *le-inf-iff*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(x \leq_{ow} y \sqcap_{ow} z) = (x \leq_{ow} y \wedge x \leq_{ow} z)$
is *semilattice-inf-class.le-inf-iff*⟨*proof*⟩

tts-lemma *le-infI*:
assumes $x \in U$ **and** $a \in U$ **and** $b \in U$ **and** $x \leq_{ow} a$ **and** $x \leq_{ow} b$
shows $x \leq_{ow} a \sqcap_{ow} b$
is *semilattice-inf-class.le-infI*⟨*proof*⟩

tts-lemma *le-infE*:
assumes $x \in U$
and $a \in U$
and $b \in U$
and $x \leq_{ow} a \sqcap_{ow} b$
and $[\![x \leq_{ow} a;\ x \leq_{ow} b]\!] \Longrightarrow P$
shows P
is *semilattice-inf-class.le-infE*⟨*proof*⟩

tts-lemma *inf-mono*:
assumes $a \in U$
and $c \in U$
and $b \in U$
and $d \in U$
and $a \leq_{ow} c$
and $b \leq_{ow} d$
shows $a \sqcap_{ow} b \leq_{ow} c \sqcap_{ow} d$

is *semilattice-inf-class.inf-mono*⟨*proof*⟩

end

end

context *semilattice-sup-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *semilattice-sup-ow-axioms*
eliminating through *simp*
begin

tts-lemma *le-iff-sup*:
assumes $x \in U$ **and** $y \in U$
shows $(x \leq_{ow} y) = (x \sqcup_{ow} y = y)$
is *semilattice-sup-class.le-iff-sup*⟨*proof*⟩

tts-lemma *less-supI1*:
assumes $x \in U$ **and** $a \in U$ **and** $b \in U$ **and** $x <_{ow} a$
shows $x <_{ow} a \sqcup_{ow} b$
is *semilattice-sup-class.less-supI1*⟨*proof*⟩

tts-lemma *less-supI2*:
assumes $x \in U$ **and** $b \in U$ **and** $a \in U$ **and** $x <_{ow} b$
shows $x <_{ow} a \sqcup_{ow} b$
is *semilattice-sup-class.less-supI2*⟨*proof*⟩

tts-lemma *le-supI1*:
assumes $x \in U$ **and** $a \in U$ **and** $b \in U$ **and** $x \leq_{ow} a$
shows $x \leq_{ow} a \sqcup_{ow} b$
is *semilattice-sup-class.le-supI1*⟨*proof*⟩

tts-lemma *le-supI2*:
assumes $x \in U$ **and** $b \in U$ **and** $a \in U$ **and** $x \leq_{ow} b$
shows $x \leq_{ow} a \sqcup_{ow} b$
is *semilattice-sup-class.le-supI2*⟨*proof*⟩

tts-lemma *le-sup-iff*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(x \sqcup_{ow} y \leq_{ow} z) = (x \leq_{ow} z \wedge y \leq_{ow} z)$
is *semilattice-sup-class.le-sup-iff*⟨*proof*⟩

tts-lemma *le-supI*:
assumes $a \in U$ **and** $x \in U$ **and** $b \in U$ **and** $a \leq_{ow} x$ **and** $b \leq_{ow} x$
shows $a \sqcup_{ow} b \leq_{ow} x$
is *semilattice-sup-class.le-supI*⟨*proof*⟩

tts-lemma *le-supE*:
assumes $a \in U$
and $b \in U$
and $x \in U$
and $a \sqcup_{ow} b \leq_{ow} x$
and $[\![a \leq_{ow} x;\ b \leq_{ow} x]\!] \Longrightarrow P$
shows P

is *semilattice-sup-class.le-supE*⟨*proof*⟩

tts-lemma *sup-unique*:
assumes $\forall x{\in}U.\ \forall y{\in}U.\ f\ x\ y \in U$
and $x \in U$
and $y \in U$
and $\bigwedge x\ y.\ [\![x \in U;\ y \in U]\!] \Longrightarrow x \le_{ow} f\ x\ y$
and $\bigwedge x\ y.\ [\![x \in U;\ y \in U]\!] \Longrightarrow y \le_{ow} f\ x\ y$
and $\bigwedge x\ y\ z.\ [\![x \in U;\ y \in U;\ z \in U;\ y \le_{ow} x;\ z \le_{ow} x]\!] \Longrightarrow f\ y\ z \le_{ow} x$
shows $x \sqcup_{ow} y = f\ x\ y$
is *semilattice-sup-class.sup-unique*⟨*proof*⟩

tts-lemma *sup-mono*:
assumes $a \in U$
and $c \in U$
and $b \in U$
and $d \in U$
and $a \le_{ow} c$
and $b \le_{ow} d$
shows $a \sqcup_{ow} b \le_{ow} c \sqcup_{ow} d$
is *semilattice-sup-class.sup-mono*⟨*proof*⟩

end

end

3.12.3 Bounded semilattices

Definitions and common properties

locale *semilattice-neutral-ow* = *semilattice-ow U f* + *comm-monoid-ow U f z*
for *U* :: *'al set* **and** *f z*

locale *semilattice-neutral-order-ow* =
sl-neut: *semilattice-neutral-ow U f z* +
sl-ord: *semilattice-order-ow U f le ls*
for *U* :: *'al set* **and** *f z le ls*
begin

sublocale *order-top-ow U* ‹$(\le_{ow})$› ‹$(<_{ow})$› ‹$\mathbf{1}_{ow}$›
⟨*proof*⟩

end

locale *bounded-semilattice-inf-top-ow* =
semilattice-inf-ow U inf le ls + *order-top-ow U le ls top*
for *U* :: *'al set* **and** *inf le ls top*
begin

sublocale *inf-top*: *semilattice-neutral-order-ow U* ‹$(\sqcap_{ow})$› ‹$\top_{ow}$› ‹$(\le_{ow})$› ‹$(<_{ow})$›
⟨*proof*⟩

end

locale *bounded-semilattice-sup-bot-ow* =
semilattice-sup-ow U sup le ls + *order-bot-ow U bot le ls*
for *U* :: *'al set* **and** *sup le ls bot*
begin

sublocale *sup-bot*: *semilattice-neutral-order-ow U* ‹$(\sqcup_{ow})$› ‹$\bot_{ow}$› ‹$(\geq_{ow})$› ‹$(>_{ow})$›
⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *semilattice-neutral-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*semilattice-neutral-ow* (*Collect* (*Domainp A*))) *semilattice-neutr*
⟨*proof*⟩

lemma *semilattice-neutr-order-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
A ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(=)
)
(*semilattice-neutral-order-ow* (*Collect* (*Domainp A*)))
semilattice-neutr-order
⟨*proof*⟩

lemma *bounded-semilattice-inf-top-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
A ===>
(=)
)
(*bounded-semilattice-inf-top-ow* (*Collect* (*Domainp A*)))
class.bounded-semilattice-inf-top
⟨*proof*⟩

lemma *bounded-semilattice-sup-bot-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
A ===>
(=)
)
(*bounded-semilattice-sup-bot-ow* (*Collect* (*Domainp A*)))

class.bounded-semilattice-sup-bot
⟨*proof*⟩

end

3.13 Relativization of the results about lattices

3.13.1 Simple lattices

Definitions and common properties

locale *lattice-ow* =
  semilattice-inf-ow U inf le ls + *semilattice-sup-ow U sup le ls*
  for *U* :: *'al set* **and** *inf le ls sup*

locale *lattice-pair-ow* =
  $lattice_1$: *lattice-ow* U_1 inf_1 le_1 ls_1 sup_1 +
  $lattice_2$: *lattice-ow* U_2 inf_2 le_2 ls_2 sup_2
  for U_1 :: *'al set* **and** inf_1 le_1 ls_1 sup_1
    and U_2 :: *'bl set* **and** inf_2 le_2 ls_2 sup_2
begin

sublocale *semilattice-inf-pair-ow* ⟨*proof*⟩
sublocale *semilattice-sup-pair-ow* ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *lattice-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (
      (*A* ===> *A* ===> *A*) ===>
      (*A* ===> *A* ===> (=)) ===>
      (*A* ===> *A* ===> (=)) ===>
      (*A* ===> *A* ===> *A*) ===>
      (=)
)
    (*lattice-ow* (*Collect* (*Domainp A*))) *class.lattice*
  ⟨*proof*⟩

end

Relativization

context *lattice-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  rewriting *ctr-simps*
  substituting *lattice-ow-axioms*
  eliminating through *simp*
begin

tts-lemma *inf-sup-aci*:
  assumes $x \in U$ **and** $y \in U$
  shows
    $x \sqcap_{ow} y = y \sqcap_{ow} x$

$z \in U \Longrightarrow x \sqcap_{ow} y \sqcap_{ow} z = x \sqcap_{ow} (y \sqcap_{ow} z)$
$z \in U \Longrightarrow x \sqcap_{ow} (y \sqcap_{ow} z) = y \sqcap_{ow} (x \sqcap_{ow} z)$
$x \sqcap_{ow} (x \sqcap_{ow} y) = x \sqcap_{ow} y$
$x \sqcup_{ow} y = y \sqcup_{ow} x$
$z \in U \Longrightarrow x \sqcup_{ow} y \sqcup_{ow} z = x \sqcup_{ow} (y \sqcup_{ow} z)$
$z \in U \Longrightarrow x \sqcup_{ow} (y \sqcup_{ow} z) = y \sqcup_{ow} (x \sqcup_{ow} z)$
$x \sqcup_{ow} (x \sqcup_{ow} y) = x \sqcup_{ow} y$
is *lattice-class.inf-sup-aci*⟨*proof*⟩

tts-lemma *inf-sup-absorb*:
assumes $x \in U$ **and** $y \in U$
shows $x \sqcap_{ow} (x \sqcup_{ow} y) = x$
is *lattice-class.inf-sup-absorb*⟨*proof*⟩

tts-lemma *sup-inf-absorb*:
assumes $x \in U$ **and** $y \in U$
shows $x \sqcup_{ow} (x \sqcap_{ow} y) = x$
is *lattice-class.sup-inf-absorb*⟨*proof*⟩

tts-lemma *bdd-above-insert*:
assumes $a \in U$ **and** $A \subseteq U$
shows *local.bdd-above* (*insert a A*) = *local.bdd-above A*
is *lattice-class.bdd-above-insert*⟨*proof*⟩

tts-lemma *bdd-below-insert*:
assumes $a \in U$ **and** $A \subseteq U$
shows *local.bdd-below* (*insert a A*) = *local.bdd-below A*
is *lattice-class.bdd-below-insert*⟨*proof*⟩

tts-lemma *distrib-sup-le*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcup_{ow} (y \sqcap_{ow} z) \leq_{ow} x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
is *lattice-class.distrib-sup-le*⟨*proof*⟩

tts-lemma *distrib-inf-le*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z) \leq_{ow} x \sqcap_{ow} (y \sqcup_{ow} z)$
is *lattice-class.distrib-inf-le*⟨*proof*⟩

tts-lemma *distrib-imp*1:
assumes $x \in U$
and $y \in U$
and $z \in U$
and
$\bigwedge x\ y\ z.\ [\![x \in U;\ y \in U;\ z \in U]\!] \Longrightarrow$
$x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
shows $x \sqcup_{ow} (y \sqcap_{ow} z) = x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
is *lattice-class.distrib-imp*1⟨*proof*⟩

tts-lemma *distrib-imp*2:
assumes $x \in U$
and $y \in U$
and $z \in U$
and
$\bigwedge x\ y\ z.\ [\![x \in U;\ y \in U;\ z \in U]\!] \Longrightarrow$
$x \sqcup_{ow} (y \sqcap_{ow} z) = x \sqcup_{ow} y \sqcap_{ow} (x \sqcup_{ow} z)$
shows $x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
is *lattice-class.distrib-imp*2⟨*proof*⟩

tts-lemma *bdd-above-Un*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows *local.bdd-above* $(A \cup B)$ = $(local.bdd\text{-}above\ A \wedge local.bdd\text{-}above\ B)$
is *lattice-class.bdd-above-Un*⟨*proof*⟩

tts-lemma *bdd-below-Un*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows *local.bdd-below* $(A \cup B)$ = $(local.bdd\text{-}below\ A \wedge local.bdd\text{-}below\ B)$
is *lattice-class.bdd-below-Un*⟨*proof*⟩

tts-lemma *bdd-above-image-sup*:
assumes *range* $f \subseteq U$ **and** *range* $g \subseteq U$
shows *local.bdd-above* $((\lambda x.\ f\ x \sqcup_{ow} g\ x)\ `\ A)$ =
$(local.bdd\text{-}above\ (f\ `\ A) \wedge local.bdd\text{-}above\ (g\ `\ A))$
is *lattice-class.bdd-above-image-sup*⟨*proof*⟩

tts-lemma *bdd-below-image-inf*:
assumes *range* $f \subseteq U$ **and** *range* $g \subseteq U$
shows *local.bdd-below* $((\lambda x.\ f\ x \sqcap_{ow} g\ x)\ `\ A)$ =
$(local.bdd\text{-}below\ (f\ `\ A) \wedge local.bdd\text{-}below\ (g\ `\ A))$
is *lattice-class.bdd-below-image-inf*⟨*proof*⟩

end

tts-context
tts: $(?'a$ **to** $U)$
rewriting *ctr-simps*
substituting *lattice-ow-axioms*
eliminating through *simp*
begin

tts-lemma *bdd-above-UN*:
assumes $U \neq \{\}$ **and** *range* $A \subseteq Pow\ U$ **and** *finite* I
shows *bdd-above* $(\bigcup\ (A\ `\ I))$ = $(\forall\ x{\in}I.\ bdd\text{-}above\ (A\ x))$
is *lattice-class.bdd-above-UN* ⟨*proof*⟩

tts-lemma *bdd-below-UN*:
assumes $U \neq \{\}$ **and** *range* $A \subseteq Pow\ U$ **and** *finite* I
shows *local.bdd-below* $(\bigcup\ (A\ `\ I))$ = $(\forall\ x{\in}I.\ local.bdd\text{-}below\ (A\ x))$
is *lattice-class.bdd-below-UN*⟨*proof*⟩

tts-lemma *bdd-above-finite*:
assumes $U \neq \{\}$ **and** $A \subseteq U$ **and** *finite* A
shows *local.bdd-above* A
is *lattice-class.bdd-above-finite*⟨*proof*⟩

tts-lemma *bdd-below-finite*:
assumes $U \neq \{\}$ **and** $A \subseteq U$ **and** *finite* A
shows *local.bdd-below* A
is *lattice-class.bdd-below-finite*⟨*proof*⟩

end

end

3.13.2 Bounded lattices

Definitions and common properties

locale *bounded-lattice-bot-ow* =
  lattice-ow U inf le ls sup + *order-bot-ow U bot le ls*
  for U :: $'al$ *set* **and** *inf le ls sup bot*
begin

sublocale *bounded-semilattice-sup-bot-ow U* ‹$(\sqcup_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$\bot_{ow}$› ⟨*proof*⟩

end

locale *bounded-lattice-bot-pair-ow* =
  blb_1: *bounded-lattice-bot-ow* U_1 inf_1 le_1 ls_1 sup_1 bot_1 +
  blb_2: *bounded-lattice-bot-ow* U_2 inf_2 le_2 ls_2 sup_2 bot_2
  for U_1 :: $'al$ *set* **and** inf_1 le_1 ls_1 sup_1 bot_1
    and U_2 :: $'bl$ *set* **and** inf_2 le_2 ls_2 sup_2 bot_2
begin

sublocale *lattice-pair-ow* ⟨*proof*⟩
sublocale *order-bot-pair-ow* U_1 bot_1 le_1 ls_1 U_2 bot_2 le_2 ls_2 ⟨*proof*⟩

end

locale *bounded-lattice-top-ow* =
  lattice-ow U inf le ls sup + *order-top-ow U le ls top*
  for U :: $'al$ *set* **and** *inf le ls sup top*
begin

sublocale *bounded-semilattice-inf-top-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$\top_{ow}$› ⟨*proof*⟩

end

locale *bounded-lattice-top-pair-ow* =
  blb_1: *bounded-lattice-top-ow* U_1 inf_1 le_1 ls_1 sup_1 top_1 +
  blb_2: *bounded-lattice-top-ow* U_2 inf_2 le_2 ls_2 sup_2 top_2
  for U_1 :: $'al$ *set* **and** inf_1 le_1 ls_1 sup_1 top_1
    and U_2 :: $'bl$ *set* **and** inf_2 le_2 ls_2 sup_2 top_2
begin

sublocale *lattice-pair-ow* ⟨*proof*⟩
sublocale *order-top-pair-ow* U_1 le_1 ls_1 top_1 U_2 le_2 ls_2 top_2 ⟨*proof*⟩

end

locale *bounded-lattice-ow* =
  lattice-ow U inf le ls sup +
  order-bot-ow U bot le ls +
  order-top-ow U le ls top
  for U :: $'al$ *set* **and** *inf le ls sup bot top*
begin

sublocale *bounded-lattice-bot-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$(\sqcup_{ow})$› ‹$\bot_{ow}$› ⟨*proof*⟩
sublocale *bounded-lattice-top-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$(\sqcup_{ow})$› ‹$\top_{ow}$› ⟨*proof*⟩

end

locale *bounded-lattice-pair-ow* =

bl$_1$: *bounded-lattice-ow* U_1 *inf*$_1$ *le*$_1$ *ls*$_1$ *sup*$_1$ *bot*$_1$ *top*$_1$ +
bl$_2$: *bounded-lattice-ow* U_2 *inf*$_2$ *le*$_2$ *ls*$_2$ *sup*$_2$ *bot*$_2$ *top*$_2$
for U_1 :: *'al set* **and** *inf*$_1$ *le*$_1$ *ls*$_1$ *sup*$_1$ *bot*$_1$ *top*$_1$
and U_2 :: *'bl set* **and** *inf*$_2$ *le*$_2$ *ls*$_2$ *sup*$_2$ *bot*$_2$ *top*$_2$
begin

sublocale *bounded-lattice-bot-pair-ow* ⟨*proof*⟩
sublocale *bounded-lattice-top-pair-ow* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *bounded-lattice-bot-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(=)
)
(*bounded-lattice-bot-ow* (*Collect* (*Domainp A*)))
class.bounded-lattice-bot
⟨*proof*⟩

lemma *bounded-lattice-top-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows

(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
(=)
)
(*bounded-lattice-top-ow* (*Collect* (*Domainp A*)))
class.bounded-lattice-top
⟨*proof*⟩

lemma *bounded-lattice-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> *A*) ===>
A ===>
A ===>

(=)
)
(*bounded-lattice-ow* (*Collect* (*Domainp A*))) *class.bounded-lattice*
⟨*proof*⟩

end

Relativization

context *bounded-lattice-bot-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *bounded-lattice-bot-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
applying [*OF inf-closed' sup-closed' bot-closed*]
begin

tts-lemma *inf-bot-left*:
assumes $x \in U$
shows $\bot_{ow} \sqcap_{ow} x = \bot_{ow}$
is *bounded-lattice-bot-class.inf-bot-left*⟨*proof*⟩

tts-lemma *inf-bot-right*:
assumes $x \in U$
shows $x \sqcap_{ow} \bot_{ow} = \bot_{ow}$
is *bounded-lattice-bot-class.inf-bot-right*⟨*proof*⟩

end

end

context *bounded-lattice-top-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *bounded-lattice-top-ow-axioms* **and** *inf-top.sl-neut.not-empty*
applying [*OF inf-closed' sup-closed' top-closed*]
begin

tts-lemma *sup-top-left*:
assumes $x \in U$
shows $\top_{ow} \sqcup_{ow} x = \top_{ow}$
is *bounded-lattice-top-class.sup-top-left*⟨*proof*⟩

tts-lemma *sup-top-right*:
assumes $x \in U$
shows $x \sqcup_{ow} \top_{ow} = \top_{ow}$
is *bounded-lattice-top-class.sup-top-right*⟨*proof*⟩

end

end

context *bounded-lattice-ow*

begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *bounded-lattice-ow-axioms*
applying [*OF sup-bot.sl-neut.not-empty*, *simplified*]
begin

tts-lemma *atLeastAtMost-eq-UNIV-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(\{x..{}_{ow} y\} = U) = (x = \bot_{ow} \wedge y = \top_{ow})$
is *bounded-lattice-class.atLeastAtMost-eq-UNIV-iff*⟨*proof*⟩

end

end

3.13.3 Distributive lattices

Definitions and common properties

locale *distrib-lattice-ow* =
lattice-ow U inf le ls sup **for** U :: *'al set* **and** *inf le ls sup* +
assumes *sup-inf-distrib*1:
$[\![x \in U; y \in U; z \in U]\!] \Longrightarrow x \sqcup_{ow} (y \sqcap_{ow} z) = (x \sqcup_{ow} y) \sqcap_{ow} (x \sqcup_{ow} z)$

Transfer rules

context
includes *lifting-syntax*
begin

lemma *distrib-lattice-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> *A*) ===>
(=)
)
(*distrib-lattice-ow* (*Collect* (*Domainp A*))) *class.distrib-lattice*
⟨*proof*⟩

end

Relativization

context *distrib-lattice-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *distrib-lattice-ow-axioms*
eliminating through *simp*
begin

tts-lemma *inf-sup-distrib*1:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $x \sqcap_{ow} (y \sqcup_{ow} z) = x \sqcap_{ow} y \sqcup_{ow} (x \sqcap_{ow} z)$
is *distrib-lattice-class.inf-sup-distrib*1⟨*proof*⟩

tts-lemma *inf-sup-distrib*2:
assumes $y \in U$ **and** $z \in U$ **and** $x \in U$
shows $y \sqcup_{ow} z \sqcap_{ow} x = y \sqcap_{ow} x \sqcup_{ow} (z \sqcap_{ow} x)$
is *distrib-lattice-class.inf-sup-distrib*2⟨*proof*⟩

tts-lemma *sup-inf-distrib*2:
assumes $y \in U$ **and** $z \in U$ **and** $x \in U$
shows $y \sqcap_{ow} z \sqcup_{ow} x = y \sqcup_{ow} x \sqcap_{ow} (z \sqcup_{ow} x)$
is *distrib-lattice-class.sup-inf-distrib*2⟨*proof*⟩

end

end

3.14 Relativization of the results about complete lattices

3.14.1 Simple complete lattices

Definitions and common properties

locale *Inf-ow* =
  fixes U :: *'al set* **and** *Inf* (‹$\bigsqcap_{ow}$›)
  assumes *Inf-closed*[*simp*]: $\bigsqcap_{ow}$ ‘ *Pow U* ⊆ *U*
begin

notation *Inf* (‹$\bigsqcap_{ow}$›)

lemma *Inf-closed'*[*simp*]: ∀ *x*⊆*U*. $\bigsqcap_{ow}$ *x* ∈ *U* ⟨*proof*⟩

end

locale *Inf-pair-ow* = *Inf*$_1$: *Inf-ow U*$_1$ *Inf*$_1$ + *Inf*$_2$: *Inf-ow U*$_2$ *Inf*$_2$
  for U_1 :: *'al set* **and** *Inf*$_1$ **and** U_2 :: *'bl set* **and** *Inf*$_2$
begin

notation *Inf*$_1$ (‹$\bigsqcap_{ow.1}$›)
notation *Inf*$_2$ (‹$\bigsqcap_{ow.2}$›)

end

locale *Sup-ow* =
  fixes U :: *'al set* **and** *Sup* (‹$\bigsqcup_{ow}$›)
  assumes *Sup-closed*[*simp*]: $\bigsqcup_{ow}$ ‘ *Pow U* ⊆ *U*
begin

notation *Sup* (‹$\bigsqcup_{ow}$›)

lemma *Sup-closed'*[*simp*]: ∀ *x*⊆*U*. $\bigsqcup_{ow}$ *x* ∈ *U* ⟨*proof*⟩

end

locale *Sup-pair-ow* = *Sup*$_1$: *Sup-ow U*$_1$ *Sup*$_1$ + *Sup*$_2$: *Sup-ow U*$_2$ *Sup*$_2$
  for U_1 :: *'al set* **and** *Sup*$_1$ **and** U_2 :: *'bl set* **and** *Sup*$_2$
begin

notation *Sup*$_1$ (‹$\bigsqcup_{ow.1}$›)
notation *Sup*$_2$ (‹$\bigsqcup_{ow.2}$›)

end

locale *complete-lattice-ow* =
  lattice-ow U inf le ls sup +
  Inf-ow U Inf +
  Sup-ow U Sup +
  bot-ow U bot +
  top-ow U top
  for U :: *'al set* **and** *Inf Sup inf le ls sup bot top* +
  assumes *Inf-lower*: ⟦ *A* ⊆ *U*; *x* ∈ *A* ⟧ ⟹ $\bigsqcap_{ow}$ *A* ≤$_{ow}$ *x*
    and *Inf-greatest*:
      ⟦ *A* ⊆ *U*; *z* ∈ *U*; (⋀*x*. *x* ∈ *A* ⟹ *z* ≤$_{ow}$ *x*) ⟧ ⟹ *z* ≤$_{ow}$ $\bigsqcap_{ow}$ *A*
    and *Sup-upper*: ⟦ *x* ∈ *A*; *A* ⊆ *U* ⟧ ⟹ *x* ≤$_{ow}$ $\bigsqcup_{ow}$ *A*
    and *Sup-least*:
      ⟦ *A* ⊆ *U*; *z* ∈ *U*; (⋀*x*. *x* ∈ *A* ⟹ *x* ≤$_{ow}$ *z*) ⟧ ⟹ $\bigsqcup_{ow}$ *A* ≤$_{ow}$ *z*

and *Inf-empty*[*simp*]: $\bigsqcap_{ow}\ \{\} = \top_{ow}$
and *Sup-empty*[*simp*]: $\bigsqcup_{ow}\ \{\} = \bot_{ow}$
begin

sublocale *bounded-lattice-ow U* ‹$(\sqcap_{ow})$› ‹$(\leq_{ow})$› ‹$(<_{ow})$› ‹$(\sqcup_{ow})$› ‹$\bot_{ow}$› ‹$\top_{ow}$›
⟨*proof*⟩

tts-register-sbts ‹$\bot_{ow}$› | *U*
⟨*proof*⟩

tts-register-sbts ‹$\top_{ow}$› | *U*
⟨*proof*⟩

end

locale *complete-lattice-pair-ow* =
cl_1: *complete-lattice-ow* U_1 Inf_1 Sup_1 inf_1 le_1 ls_1 sup_1 bot_1 top_1 +
cl_2: *complete-lattice-ow* U_2 Inf_2 Sup_2 inf_2 le_2 ls_2 sup_2 bot_2 top_2
for U_1 :: *'al set* **and** Inf_1 Sup_1 inf_1 le_1 ls_1 sup_1 bot_1 top_1
and U_2 :: *'bl set* **and** Inf_2 Sup_2 inf_2 le_2 ls_2 sup_2 bot_2 top_2
begin

sublocale *bounded-lattice-pair-ow* ⟨*proof*⟩
sublocale *Inf-pair-ow* ⟨*proof*⟩
sublocale *Sup-pair-ow* ⟨*proof*⟩

end

Transfer rules

context
includes *lifting-syntax*
begin

lemma *complete-lattice-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
(*rel-set A* ===> *A*) ===>
(*rel-set A* ===> *A*) ===>
(*A* ===> *A* ===> *A*) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> (=)) ===>
(*A* ===> *A* ===> *A*) ===>
A ===> *A* ===>
(=)
)
(*complete-lattice-ow* (*Collect* (*Domainp A*))) *class.complete-lattice*
⟨*proof*⟩

end

Relativization

context *complete-lattice-ow*
begin

tts-context

tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating ‹*?a* ∈ *?A*› **and** ‹*?A* ⊆ *?B*› **through** *auto*
applying [*OF Inf-closed' Sup-closed' inf-closed' sup-closed'*]
begin

tts-lemma *Inf-atLeast*:
assumes $x \in U$
shows $\bigsqcap_{ow} \{x..._{ow}\} = x$
is *complete-lattice-class.Inf-atLeast*⟨*proof*⟩

tts-lemma *Inf-atMost*:
assumes $x \in U$
shows $\bigsqcap_{ow} \{.._{ow}x\} = \bot_{ow}$
is *complete-lattice-class.Inf-atMost*⟨*proof*⟩

tts-lemma *Sup-atLeast*:
assumes $x \in U$
shows $\bigsqcup_{ow} \{x..._{ow}\} = \top_{ow}$
is *complete-lattice-class.Sup-atLeast*⟨*proof*⟩

tts-lemma *Sup-atMost*:
assumes $y \in U$
shows $\bigsqcup_{ow} \{.._{ow}y\} = y$
is *complete-lattice-class.Sup-atMost*⟨*proof*⟩

tts-lemma *Inf-insert*:
assumes $a \in U$ **and** $A \subseteq U$
shows $\bigsqcap_{ow} (\mathit{insert}\ a\ A) = a \sqcap_{ow} \bigsqcap_{ow} A$
is *complete-lattice-class.Inf-insert*⟨*proof*⟩

tts-lemma *Sup-insert*:
assumes $a \in U$ **and** $A \subseteq U$
shows $\bigsqcup_{ow} (\mathit{insert}\ a\ A) = a \sqcup_{ow} \bigsqcup_{ow} A$
is *complete-lattice-class.Sup-insert*⟨*proof*⟩

tts-lemma *Inf-atMostLessThan*:
assumes $x \in U$ **and** $\top_{ow} <_{ow} x$
shows $\bigsqcap_{ow} \{..<_{ow}x\} = \bot_{ow}$
is *complete-lattice-class.Inf-atMostLessThan*⟨*proof*⟩

tts-lemma *Sup-greaterThanAtLeast*:
assumes $x \in U$ **and** $x <_{ow} \top_{ow}$
shows $\bigsqcup_{ow} \{x<_{ow}..\} = \top_{ow}$
is *complete-lattice-class.Sup-greaterThanAtLeast*⟨*proof*⟩

tts-lemma *le-Inf-iff*:
assumes $b \in U$ **and** $A \subseteq U$
shows $(b \leq_{ow} \bigsqcap_{ow} A) = \mathit{Ball}\ A\ ((\leq_{ow})\ b)$
is *complete-lattice-class.le-Inf-iff*⟨*proof*⟩

tts-lemma *Sup-le-iff*:
assumes $A \subseteq U$ **and** $b \in U$
shows $(\bigsqcup_{ow} A \leq_{ow} b) = (\forall\, x{\in}A.\ x \leq_{ow} b)$
is *complete-lattice-class.Sup-le-iff*⟨*proof*⟩

tts-lemma *Inf-atLeastLessThan*:

assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\bigsqcap_{ow}$ (*on* U *with* $(\leq_{ow})$ $(<_{ow})$: $\{x..<y\}$) = x
is *complete-lattice-class.Inf-atLeastLessThan*⟨*proof*⟩

tts-lemma *Sup-greaterThanAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_{ow} y$
shows $\bigsqcup_{ow} \{x<_{ow}..y\} = y$
is *complete-lattice-class.Sup-greaterThanAtMost*⟨*proof*⟩

tts-lemma *Inf-atLeastAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $\bigsqcap_{ow} \{x.._{ow}y\} = x$
is *complete-lattice-class.Inf-atLeastAtMost*⟨*proof*⟩

tts-lemma *Sup-atLeastAtMost*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y$
shows $\bigsqcup_{ow} \{x.._{ow}y\} = y$
is *complete-lattice-class.Sup-atLeastAtMost*⟨*proof*⟩

tts-lemma *Inf-lower*2:
assumes $A \subseteq U$ **and** $v \in U$ **and** $u \in A$ **and** $u \leq_{ow} v$
shows $\bigsqcap_{ow} A \leq_{ow} v$
is *complete-lattice-class.Inf-lower*2⟨*proof*⟩

tts-lemma *Sup-upper*2:
assumes $A \subseteq U$ **and** $v \in U$ **and** $u \in A$ **and** $v \leq_{ow} u$
shows $v \leq_{ow} \bigsqcup_{ow} A$
is *complete-lattice-class.Sup-upper*2⟨*proof*⟩

tts-lemma *Inf-less-eq*:
assumes $A \subseteq U$ **and** $u \in U$ **and** $\bigwedge v.\ v \in A \Longrightarrow v \leq_{ow} u$ **and** $A \neq \{\}$
shows $\bigsqcap_{ow} A \leq_{ow} u$
given *complete-lattice-class.Inf-less-eq* ⟨*proof*⟩

tts-lemma *less-eq-Sup*:
assumes $A \subseteq U$ **and** $u \in U$ **and** $\bigwedge v.\ v \in A \Longrightarrow u \leq_{ow} v$ **and** $A \neq \{\}$
shows $u \leq_{ow} \bigsqcup_{ow} A$
given *complete-lattice-class.less-eq-Sup* ⟨*proof*⟩

tts-lemma *Sup-eqI*:
assumes $A \subseteq U$
and $x \in U$
and $\bigwedge y.\ y \in A \Longrightarrow y \leq_{ow} x$
and $\bigwedge y.\ [\![y \in U;\ \bigwedge z.\ z \in A \Longrightarrow z \leq_{ow} y]\!] \Longrightarrow x \leq_{ow} y$
shows $\bigsqcup_{ow} A = x$
given *complete-lattice-class.Sup-eqI*
⟨*proof*⟩

tts-lemma *Inf-eqI*:
assumes $A \subseteq U$
and $x \in U$
and $\bigwedge i.\ i \in A \Longrightarrow x \leq_{ow} i$
and $\bigwedge y.\ [\![y \in U;\ \bigwedge i.\ i \in A \Longrightarrow y \leq_{ow} i]\!] \Longrightarrow y \leq_{ow} x$
shows $\bigsqcap_{ow} A = x$
given *complete-lattice-class.Inf-eqI*
⟨*proof*⟩

tts-lemma *Inf-union-distrib*:

assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\bigsqcap_{ow} (A \cup B) = \bigsqcap_{ow} A \sqcap_{ow} \bigsqcap_{ow} B$
is *complete-lattice-class.Inf-union-distrib*⟨*proof*⟩

tts-lemma *Sup-union-distrib*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\bigsqcup_{ow} (A \cup B) = \bigsqcup_{ow} A \sqcup_{ow} \bigsqcup_{ow} B$
is *complete-lattice-class.Sup-union-distrib*⟨*proof*⟩

tts-lemma *Sup-inter-less-eq*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\bigsqcup_{ow} (A \cap B) \leq_{ow} \bigsqcup_{ow} A \sqcap_{ow} \bigsqcup_{ow} B$
is *complete-lattice-class.Sup-inter-less-eq*⟨*proof*⟩

tts-lemma *less-eq-Inf-inter*:
assumes $A \subseteq U$ **and** $B \subseteq U$
shows $\bigsqcap_{ow} A \sqcup_{ow} \bigsqcap_{ow} B \leq_{ow} \bigsqcap_{ow} (A \cap B)$
is *complete-lattice-class.less-eq-Inf-inter*⟨*proof*⟩

tts-lemma *Sup-subset-mono*:
assumes $B \subseteq U$ **and** $A \subseteq B$
shows $\bigsqcup_{ow} A \leq_{ow} \bigsqcup_{ow} B$
is *complete-lattice-class.Sup-subset-mono*⟨*proof*⟩

tts-lemma *Inf-superset-mono*:
assumes $A \subseteq U$ **and** $B \subseteq A$
shows $\bigsqcap_{ow} A \leq_{ow} \bigsqcap_{ow} B$
is *complete-lattice-class.Inf-superset-mono*⟨*proof*⟩

tts-lemma *Sup-bot-conv*:
assumes $A \subseteq U$
shows
$(\bigsqcup_{ow} A = \bot_{ow}) = (\forall\, x{\in}A.\ x = \bot_{ow})$
$(\bot_{ow} = \bigsqcup_{ow} A) = (\forall\, x{\in}A.\ x = \bot_{ow})$
is *complete-lattice-class.Sup-bot-conv*⟨*proof*⟩

tts-lemma *Inf-top-conv*:
assumes $A \subseteq U$
shows
$(\bigsqcap_{ow} A = \top_{ow}) = (\forall\, x{\in}A.\ x = \top_{ow})$
$(\top_{ow} = \bigsqcap_{ow} A) = (\forall\, x{\in}A.\ x = \top_{ow})$
is *complete-lattice-class.Inf-top-conv*⟨*proof*⟩

tts-lemma *Inf-le-Sup*:
assumes $A \subseteq U$ **and** $A \neq \{\}$
shows $\bigsqcap_{ow} A \leq_{ow} \bigsqcup_{ow} A$
is *complete-lattice-class.Inf-le-Sup*⟨*proof*⟩

tts-lemma *INF-UNIV-bool-expand*:
assumes *range* $A \subseteq U$
shows $\bigsqcap_{ow}$ (*range* A) = A *True* $\sqcap_{ow}$ A *False*
is *complete-lattice-class.INF-UNIV-bool-expand*⟨*proof*⟩

tts-lemma *SUP-UNIV-bool-expand*:
assumes *range* $A \subseteq U$
shows $\bigsqcup_{ow}$ (*range* A) = A *True* $\sqcup_{ow}$ A *False*
is *complete-lattice-class.SUP-UNIV-bool-expand*⟨*proof*⟩

tts-lemma *Sup-mono*:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** $\bigwedge a.\ a \in A \Longrightarrow Bex\ B\ ((\leq_{ow})\ a)$
shows $\bigsqcup_{ow}\ A \leq_{ow} \bigsqcup_{ow}\ B$
given *complete-lattice-class.Sup-mono* ⟨*proof*⟩

tts-lemma *Inf-mono*:
assumes $B \subseteq U$
and $A \subseteq U$
and $\bigwedge b.\ b \in B \Longrightarrow \exists\, x{\in}A.\ x \leq_{ow} b$
shows $\bigsqcap_{ow}\ A \leq_{ow} \bigsqcap_{ow}\ B$
given *complete-lattice-class.Inf-mono* ⟨*proof*⟩

tts-lemma *Sup-Inf-le*:
assumes $A \subseteq Pow\ U$
shows $\bigsqcup_{ow}$
(
$\bigsqcap_{ow}$ ‘ $\{x.\ x \subseteq U \wedge (\exists f{\in}\{f.\ f\ `\ Pow\ U \subseteq U\}.\ x = f\ `\ A \wedge (\forall\, Y{\in}A.\ f\ Y \in Y))\}$
) $\leq_{ow} \bigsqcap_{ow}$ ($\bigsqcup_{ow}$ ‘ A)
is *complete-lattice-class.Sup-Inf-le*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
applying [
OF Inf-closed' Sup-closed' inf-closed' sup-closed' bot-closed top-closed
]
begin

tts-lemma *Inf-UNIV*: $\bigsqcap_{ow}\ U = \bot_{ow}$
is *complete-lattice-class.Inf-UNIV*⟨*proof*⟩

tts-lemma *Sup-UNIV*: $\bigsqcup_{ow}\ U = \top_{ow}$
is *complete-lattice-class.Sup-UNIV*⟨*proof*⟩

end

context
fixes U_2 :: *'b set*
begin

lemmas [*transfer-rule*] =
image-transfer[**where** *?'a='b*]

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through (*insert Sup-least*, *auto*)
begin

tts-lemma *SUP-upper2*:
assumes $A \subseteq U_2$
and $u \in U$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $i \in A$

and $u \leq_{ow} f\ i$
shows $u \leq_{ow} \bigsqcup_{ow} (f\ `\ A)$
is *complete-lattice-class.SUP-upper2*⟨*proof*⟩

tts-lemma *INF-lower2*:
assumes $A \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $u \in U$
and $i \in A$
and $f\ i \leq_{ow} u$
shows $\bigsqcap_{ow} (f\ `\ A) \leq_{ow} u$
is *complete-lattice-class.INF-lower2*⟨*proof*⟩

tts-lemma *less-INF-D*:
assumes $y \in U$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $A \subseteq U_2$
and $y <_{ow} \bigsqcap_{ow} (f\ `\ A)$
and $i \in A$
shows $y <_{ow} f\ i$
is *complete-lattice-class.less-INF-D*⟨*proof*⟩

tts-lemma *SUP-lessD*:
assumes $\forall\, x{\in}U_2.\ f\ x \in U$
and $A \subseteq U_2$
and $y \in U$
and $\bigsqcup_{ow} (f\ `\ A) <_{ow} y$
and $i \in A$
shows $f\ i <_{ow} y$
is *complete-lattice-class.SUP-lessD*⟨*proof*⟩

tts-lemma *SUP-le-iff*:
assumes $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $A \subseteq U_2$ **and** $u \in U$
shows $(\bigsqcup_{ow} (f\ `\ A) \leq_{ow} u) = (\forall\, x{\in}A.\ f\ x \leq_{ow} u)$
is *complete-lattice-class.SUP-le-iff*⟨*proof*⟩

end

end

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through (*auto dest*: *top-le*)
begin

tts-lemma *INF-eqI*:
assumes $A \subseteq U_2$
and $x \in U$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $\bigwedge i.\ [\![i \in U_2;\ i \in A]\!] \Longrightarrow x \leq_{ow} f\ i$
and $\bigwedge y.\ [\![y \in U;\ \bigwedge i.\ [\![i \in U_2;\ i \in A]\!] \Longrightarrow y \leq_{ow} f\ i]\!] \Longrightarrow y \leq_{ow} x$
shows $\bigsqcap_{ow} (f\ `\ A) = x$
is *complete-lattice-class.INF-eqI*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through (*auto simp*: *order.eq-iff*)
begin

tts-lemma *SUP-eqI*:
assumes $A \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $x \in U$
and $\bigwedge i.\ [\![i \in U_2;\ i \in A]\!] \Longrightarrow f\ i \leq_{ow} x$
and $\bigwedge y.\ [\![y \in U;\ \bigwedge i.\ [\![i \in U_2;\ i \in A]\!] \Longrightarrow f\ i \leq_{ow} y]\!] \Longrightarrow x \leq_{ow} y$
shows $\bigsqcup_{ow}\ (f\ `\ A) = x$
is *complete-lattice-class.SUP-eqI*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *INF-le-SUP*:
assumes $A \subseteq U_2$ **and** $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $A \neq \{\}$
shows $\bigsqcap_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcup_{ow}\ (f\ `\ A)$
is *complete-lattice-class.INF-le-SUP*⟨*proof*⟩

tts-lemma *INF-inf-const*1:
assumes $I \subseteq U_2$ **and** $x \in U$ **and** $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $I \neq \{\}$
shows $\bigsqcap_{ow}\ ((\lambda i.\ x \sqcap_{ow} f\ i)\ `\ I) = x \sqcap_{ow} \bigsqcap_{ow}\ (f\ `\ I)$
is *complete-lattice-class.INF-inf-const*1⟨*proof*⟩

tts-lemma *INF-inf-const*2:
assumes $I \subseteq U_2$ **and** $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $x \in U$ **and** $I \neq \{\}$
shows $\bigsqcap_{ow}\ ((\lambda i.\ f\ i \sqcap_{ow} x)\ `\ I) = \bigsqcap_{ow}\ (f\ `\ I) \sqcap_{ow} x$
is *complete-lattice-class.INF-inf-const*2⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *auto*
begin

tts-lemma *INF-eq-const*:
assumes $I \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $I \neq \{\}$
and $\bigwedge i.\ i \in I \Longrightarrow f\ i = x$
shows $\bigsqcap_{ow}\ (f\ `\ I) = x$
given *complete-lattice-class.INF-eq-const*
⟨*proof*⟩

tts-lemma *SUP-eq-const*:
assumes $I \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $I \neq \{\}$
and $\bigwedge i.\ i \in I \Longrightarrow f\ i = x$
shows $\bigsqcup_{ow}\ (f\ `\ I) = x$
given *complete-lattice-class.SUP-eq-const*
⟨*proof*⟩

tts-lemma *SUP-eq-iff*:
assumes $I \subseteq U_2$
and $c \in U$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $I \neq \{\}$
and $\bigwedge i.\ i \in I \Longrightarrow c \leq_{ow} f\ i$
shows $(\bigsqcup_{ow}\ (f\ `\ I) = c) = (\forall\, x{\in}I.\ f\ x = c)$
given *complete-lattice-class.SUP-eq-iff*
⟨*proof*⟩

tts-lemma *INF-eq-iff*:
assumes $I \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $c \in U$
and $I \neq \{\}$
and $\bigwedge i.\ i \in I \Longrightarrow f\ i \leq_{ow} c$
shows $(\bigsqcap_{ow}\ (f\ `\ I) = c) = (\forall\, x{\in}I.\ f\ x = c)$
given *complete-lattice-class.INF-eq-iff*
⟨*proof*⟩

end

context
fixes $U_2 :: 'b\ set$
begin

lemmas [*transfer-rule*] =
image-transfer[**where** *?'a='b*]

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through (*auto simp*: *sup-bot.top-unique top-le intro*: *Sup-least*)
begin

tts-lemma *INF-insert*:
assumes $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $a \in U_2$ **and** $A \subseteq U_2$
shows $\bigsqcap_{ow}\ (f\ `\ insert\ a\ A) = f\ a \sqcap_{ow} \bigsqcap_{ow}\ (f\ `\ A)$
is *complete-lattice-class.INF-insert*⟨*proof*⟩

tts-lemma *SUP-insert*:
assumes $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $a \in U_2$ **and** $A \subseteq U_2$
shows $\bigsqcup_{ow}\ (f\ `\ insert\ a\ A) = f\ a \sqcup_{ow} \bigsqcup_{ow}\ (f\ `\ A)$
is *complete-lattice-class.SUP-insert*⟨*proof*⟩

tts-lemma *le-INF-iff*:
assumes $u \in U$ **and** $\forall\, x{\in}U_2.\ f\ x \in U$ **and** $A \subseteq U_2$

shows $(u \leq_{ow} \bigsqcap_{ow} (f \mathbin{`} A)) = (\forall x{\in}A.\ u \leq_{ow} f\ x)$
is *complete-lattice-class.le-INF-iff*⟨*proof*⟩

tts-lemma *INF-union*:
assumes $\forall x{\in}U_2.\ M\ x \in U$ **and** $A \subseteq U_2$ **and** $B \subseteq U_2$
shows $\bigsqcap_{ow} (M \mathbin{`} (A \cup B)) = \bigsqcap_{ow} (M \mathbin{`} A) \sqcap_{ow} \bigsqcap_{ow} (M \mathbin{`} B)$
is *complete-lattice-class.INF-union*⟨*proof*⟩

tts-lemma *SUP-union*:
assumes $\forall x{\in}U_2.\ M\ x \in U$ **and** $A \subseteq U_2$ **and** $B \subseteq U_2$
shows $\bigsqcup_{ow} (M \mathbin{`} (A \cup B)) = \bigsqcup_{ow} (M \mathbin{`} A) \sqcup_{ow} \bigsqcup_{ow} (M \mathbin{`} B)$
is *complete-lattice-class.SUP-union*⟨*proof*⟩

tts-lemma *SUP-bot-conv*:
assumes $\forall x{\in}U_2.\ B\ x \in U$ **and** $A \subseteq U_2$
shows
$(\bigsqcup_{ow} (B \mathbin{`} A) = \bot_{ow}) = (\forall x{\in}A.\ B\ x = \bot_{ow})$
$(\bot_{ow} = \bigsqcup_{ow} (B \mathbin{`} A)) = (\forall x{\in}A.\ B\ x = \bot_{ow})$
is *complete-lattice-class.SUP-bot-conv*⟨*proof*⟩

tts-lemma *INF-top-conv*:
assumes $\forall x{\in}U_2.\ B\ x \in U$ **and** $A \subseteq U_2$
shows
$(\bigsqcap_{ow} (B \mathbin{`} A) = \top_{ow}) = (\forall x{\in}A.\ B\ x = \top_{ow})$
$(\top_{ow} = \bigsqcap_{ow} (B \mathbin{`} A)) = (\forall x{\in}A.\ B\ x = \top_{ow})$
is *complete-lattice-class.INF-top-conv*⟨*proof*⟩

tts-lemma *SUP-upper*:
assumes $A \subseteq U_2$ **and** $\forall x{\in}U_2.\ f\ x \in U$ **and** $i \in A$
shows $f\ i \leq_{ow} \bigsqcup_{ow} (f \mathbin{`} A)$
is *complete-lattice-class.SUP-upper*⟨*proof*⟩

tts-lemma *INF-lower*:
assumes $A \subseteq U_2$ **and** $\forall x{\in}U_2.\ f\ x \in U$ **and** $i \in A$
shows $\bigsqcap_{ow} (f \mathbin{`} A) \leq_{ow} f\ i$
is *complete-lattice-class.INF-lower*⟨*proof*⟩

tts-lemma *INF-inf-distrib*:
assumes $\forall x{\in}U_2.\ f\ x \in U$ **and** $A \subseteq U_2$ **and** $\forall x{\in}U_2.\ g\ x \in U$
shows $\bigsqcap_{ow} (f \mathbin{`} A) \sqcap_{ow} \bigsqcap_{ow} (g \mathbin{`} A) = \bigsqcap_{ow} ((\lambda a.\ f\ a \sqcap_{ow} g\ a) \mathbin{`} A)$
is *complete-lattice-class.INF-inf-distrib*⟨*proof*⟩

tts-lemma *SUP-sup-distrib*:
assumes $\forall x{\in}U_2.\ f\ x \in U$ **and** $A \subseteq U_2$ **and** $\forall x{\in}U_2.\ g\ x \in U$
shows $\bigsqcup_{ow} (f \mathbin{`} A) \sqcup_{ow} \bigsqcup_{ow} (g \mathbin{`} A) = \bigsqcup_{ow} ((\lambda a.\ f\ a \sqcup_{ow} g\ a) \mathbin{`} A)$
is *complete-lattice-class.SUP-sup-distrib*⟨*proof*⟩

tts-lemma *SUP-subset-mono*:
assumes $B \subseteq U_2$
and $\forall x{\in}U_2.\ f\ x \in U$
and $\forall x{\in}U_2.\ g\ x \in U$
and $A \subseteq B$
and $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq_{ow} g\ x$
shows $\bigsqcup_{ow} (f \mathbin{`} A) \leq_{ow} \bigsqcup_{ow} (g \mathbin{`} B)$
given *complete-lattice-class.SUP-subset-mono*
⟨*proof*⟩

tts-lemma *INF-superset-mono*:

assumes $A \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $\forall\, x{\in}U_2.\ g\ x \in U$
and $B \subseteq A$
and $\bigwedge x.\ [\![x \in U_2;\ x \in B]\!] \Longrightarrow f\ x \leq_{ow} g\ x$
shows $\bigsqcap_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcap_{ow}\ (g\ `\ B)$
given *complete-lattice-class.INF-superset-mono*
⟨*proof*⟩

tts-lemma *INF-absorb*:
assumes $I \subseteq U_2$ **and** $\forall\, x{\in}U_2.\ A\ x \in U$ **and** $k \in I$
shows $A\ k \sqcap_{ow} \bigsqcap_{ow}\ (A\ `\ I) = \bigsqcap_{ow}\ (A\ `\ I)$
is *complete-lattice-class.INF-absorb*⟨*proof*⟩

tts-lemma *SUP-absorb*:
assumes $I \subseteq U_2$ **and** $\forall\, x{\in}U_2.\ A\ x \in U$ **and** $k \in I$
shows $A\ k \sqcup_{ow} \bigsqcup_{ow}\ (A\ `\ I) = \bigsqcup_{ow}\ (A\ `\ I)$
is *complete-lattice-class.SUP-absorb*⟨*proof*⟩

end

end

context
fixes $f :: 'b \Rightarrow 'al$ **and** $U_2 :: 'b\ set$
assumes *f*[*transfer-rule*]: $\forall\, x \in U_2.\ f\ x = \bot_{ow}$
begin

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹U_2::*'b set*›)
sbterms: (‹*Orderings.bot*::*?'a*::*complete-lattice*› **to** ‹$\bot_{ow}$›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *SUP-bot*:
assumes $A \subseteq U_2$
shows $\bigsqcup_{ow}\ (f\ `\ A) = \bot_{ow}$
is *complete-lattice-class.SUP-bot*[*folded const-fun-def*]⟨*proof*⟩

end

end

context
fixes $f :: 'b \Rightarrow 'al$ **and** $U_2 :: 'b\ set$
assumes *f*[*transfer-rule*]: $\forall\, x \in U_2.\ f\ x = \top_{ow}$
begin

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹U_2::*'b set*›)
sbterms: (‹*Orderings.top*::*?'a*::*complete-lattice*› **to** ‹$\top_{ow}$›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *INF-top*:
assumes $A \subseteq U_2$
shows $\bigsqcap_{ow}\ (f\ `\ A) = \top_{ow}$
is *complete-lattice-class.INF-top*[*folded const-fun-def*]⟨*proof*⟩

end

end

context
fixes $f :: 'b \Rightarrow 'al$ **and** c **and** F **and** $U_2 :: 'b\ set$
assumes *c-closed*[*transfer-rule*]: $c \in U$
assumes *f*[*transfer-rule*]: $\forall\, x \in U_2.\ f\ x = c$
begin

tts-register-sbts ‹c› | U
⟨*proof*⟩

tts-context
tts: ($?'a$ **to** U) **and** ($?'b$ **to** ‹U_2::$'b$ *set*›)
sbterms: (‹$?c$::$?'a$::*complete-lattice*› **to** ‹c›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *INF-constant*:
assumes $A \subseteq U_2$
shows $\bigsqcap_{ow}\ (f\ `\ A) = (\textit{if}\ A = \{\}\ \textit{then}\ \top_{ow}\ \textit{else}\ c)$
is *complete-lattice-class.INF-constant*[*folded const-fun-def*]⟨*proof*⟩

tts-lemma *SUP-constant*:
assumes $A \subseteq U_2$
shows $\bigsqcup_{ow}\ (f\ `\ A) = (\textit{if}\ A = \{\}\ \textit{then}\ \bot_{ow}\ \textit{else}\ c)$
is *complete-lattice-class.SUP-constant*[*folded const-fun-def*]⟨*proof*⟩

end

tts-context
tts: ($?'a$ **to** U) **and** ($?'b$ **to** ‹U_2::$'b$ *set*›)
sbterms: (‹$?f$::$?'a$::*complete-lattice*› **to** ‹c›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating through *simp*
begin

tts-lemma *INF-const*:
assumes $A \subseteq U_2$ **and** $A \neq \{\}$
shows $\bigsqcap_{ow}\ ((\lambda i.\ c)\ `\ A) = c$
is *complete-lattice-class.INF-const*⟨*proof*⟩

tts-lemma *SUP-const*:
assumes $A \subseteq U_2$ **and** $A \neq \{\}$
shows $\bigsqcup_{ow}\ ((\lambda i.\ c)\ `\ A) = c$
is *complete-lattice-class.SUP-const*⟨*proof*⟩

end

end

end

context *complete-lattice-ow*
begin

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹*U*$_2$::*'b set*›) **and** (*?'c* **to** ‹*U*$_3$::*'c set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*
eliminating ‹*?U* ≠ {}› **through** (*force simp*: *subset-iff INF-top equals*0*D*)
applying [
OF Inf-closed' Sup-closed' inf-closed' sup-closed' bot-closed top-closed
]
begin

tts-lemma *SUP-eq*:
assumes *A* ⊆ *U*$_2$
and *B* ⊆ *U*$_3$
and ∀ *x*∈*U*$_2$. *f* (*x*::*'a*) ∈ *U*
and ∀ *x*∈*U*$_3$. *g x* ∈ *U*
and ⋀*i*. *i* ∈ *A* ⟹ ∃ *x*∈*B*. *f i* ≤$_{ow}$ *g x*
and ⋀*j*. *j* ∈ *B* ⟹ ∃ *x*∈*A*. *g j* ≤$_{ow}$ *f x*
shows ⨆$_{ow}$ (*f* ‘ *A*) = ⨆$_{ow}$ (*g* ‘ *B*)
given *complete-lattice-class.SUP-eq*
⟨*proof*⟩

tts-lemma *INF-eq*:
assumes *A* ⊆ *U*$_2$
and *B* ⊆ *U*$_3$
and ∀ *x*∈*U*$_3$. *g x* ∈ *U*
and ∀ *x*∈*U*$_2$. *f* (*x*::*'a*) ∈ *U*
and ⋀*i*. *i* ∈ *A* ⟹ ∃ *x*∈*B*. *g x* ≤$_{ow}$ *f i*
and ⋀*j*. *j* ∈ *B* ⟹ ∃ *x*∈*A*. *f x* ≤$_{ow}$ *g j*
shows ⨅$_{ow}$ (*f* ‘ *A*) = ⨅$_{ow}$ (*g* ‘ *B*)
given *complete-lattice-class.INF-eq*
⟨*proof*⟩

end

end

context *complete-lattice-ow*
begin

context
fixes *U*$_2$ :: *'b set* **and** *U*$_3$ :: *'c set*
begin

lemmas [*transfer-rule*] =
image-transfer[**where** *?'a*=*'b*]
image-transfer[**where** *?'a*=*'c*]

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹*U*$_2$::*'b set*›) **and** (*?'c* **to** ‹*U*$_3$::*'c set*›)
rewriting *ctr-simps*
substituting *complete-lattice-ow-axioms* **and** *sup-bot.sl-neut.not-empty*

applying [
OF - - Inf-closed′ Sup-closed′ inf-closed′ sup-closed′ bot-closed top-closed
]
begin

tts-lemma *ne-INF-commute*:
assumes $U_2 \neq \{\}$
and $U_3 \neq \{\}$
and $\forall\, x{\in}U_2.\ \forall\, y{\in}U_3.\ f\ (x::'b)\ y \in U$
and $B \subseteq U_3$
and $A \subseteq U_2$
shows $\bigsqcap_{ow}\ ((\lambda i.\ \bigsqcap_{ow}\ (f\ i\ `\ B))\ `\ A) = \bigsqcap_{ow}\ ((\lambda j.\ \bigsqcap_{ow}\ ((\lambda i.\ f\ i\ j)\ `\ A))\ `\ B)$
is *complete-lattice-class.INF-commute⟨proof⟩*

tts-lemma *ne-SUP-commute*:
assumes $U_2 \neq \{\}$
and $U_3 \neq \{\}$
and $\forall\, x{\in}U_2.\ \forall\, y{\in}U_3.\ f\ (x::'b)\ y \in U$
and $B \subseteq U_3$
and $A \subseteq U_2$
shows $\bigsqcup_{ow}\ ((\lambda i.\ \bigsqcup_{ow}\ (f\ i\ `\ B))\ `\ A) = \bigsqcup_{ow}\ ((\lambda j.\ \bigsqcup_{ow}\ ((\lambda i.\ f\ i\ j)\ `\ A))\ `\ B)$
is *complete-lattice-class.SUP-commute⟨proof⟩*

tts-lemma *ne-SUP-mono*:
assumes $U_2 \neq \{\}$
and $U_3 \neq \{\}$
and $A \subseteq U_2$
and $B \subseteq U_3$
and $\forall\, x{\in}U_2.\ f\ (x::'b) \in U$
and $\forall\, x{\in}U_3.\ g\ x \in U$
and $\bigwedge n.\ [\![n \in U_2;\ n \in A]\!] \Longrightarrow \exists\, x{\in}B.\ f\ n \leq_{ow} g\ x$
shows $\bigsqcup_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcup_{ow}\ (g\ `\ B)$
is *complete-lattice-class.SUP-mono⟨proof⟩*

tts-lemma *ne-INF-mono*:
assumes $U_2 \neq \{\}$
and $U_3 \neq \{\}$
and $B \subseteq U_2$
and $A \subseteq U_3$
and $\forall\, x{\in}U_3.\ f\ x \in U$
and $\forall\, x{\in}U_2.\ g\ (x::'b) \in U$
and $\bigwedge m.\ [\![m \in U_2;\ m \in B]\!] \Longrightarrow \exists\, x{\in}A.\ f\ x \leq_{ow} g\ m$
shows $\bigsqcap_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcap_{ow}\ (g\ `\ B)$
is *complete-lattice-class.INF-mono⟨proof⟩*

end

end

lemma *INF-commute*:
assumes $\forall\, x{\in}U_2.\ \forall\, y{\in}U_3.\ f\ x\ y \in U$ **and** $B \subseteq U_3$ **and** $A \subseteq U_2$
shows
$\bigsqcap_{ow}\ ((\lambda x.\ \bigsqcap_{ow}\ (f\ x\ `\ B))\ `\ A) = \bigsqcap_{ow}\ ((\lambda j.\ \bigsqcap_{ow}\ ((\lambda i.\ f\ i\ j)\ `\ A))\ `\ B)$
⟨proof⟩

lemma *SUP-commute*:
assumes $\forall\, x{\in}U_2.\ \forall\, y{\in}U_3.\ f\ x\ y \in U$ **and** $B \subseteq U_3$ **and** $A \subseteq U_2$
shows

$\bigsqcup_{ow}\ ((\lambda x.\ \bigsqcup_{ow}\ (f\ x\ `\ B))\ `\ A) = \bigsqcup_{ow}\ ((\lambda j.\ \bigsqcup_{ow}\ ((\lambda i.\ f\ i\ j)\ `\ A))\ `\ B)$
⟨*proof*⟩

lemma *SUP-mono*:
assumes $A \subseteq U_2$
and $B \subseteq U_3$
and $\forall\, x{\in}U_2.\ f\ x \in U$
and $\forall\, x{\in}U_3.\ g\ x \in U$
and $\bigwedge n.\ n \in A \Longrightarrow \exists\, m{\in}B.\ f\ n \leq_{ow} g\ m$
shows $\bigsqcup_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcup_{ow}\ (g\ `\ B)$
⟨*proof*⟩

lemma *INF-mono*:
assumes $B \subseteq U_2$
and $A \subseteq U_3$
and $\forall\, x{\in}U_3.\ f\ x \in U$
and $\forall\, x{\in}U_2.\ g\ x \in U$
and $\bigwedge m.\ m \in B \Longrightarrow \exists\, n{\in}A.\ f\ n \leq_{ow} g\ m$
shows $\bigsqcap_{ow}\ (f\ `\ A) \leq_{ow} \bigsqcap_{ow}\ (g\ `\ B)$
⟨*proof*⟩

end

context *complete-lattice-pair-ow*
begin

context
fixes $U_3 :: 'c\ set$
begin

lemmas [*transfer-rule*] =
image-transfer[**where** *?'a='c*]

end

end

3.15 Relativization of the results about linear orders

3.15.1 Linear orders

Definitions and further properties

locale *linorder-ow* = *order-ow* +
  assumes *linear*: $[\![\ x \in U;\ y \in U\ ]\!] \Longrightarrow x \leq_{ow} y \vee y \leq_{ow} x$
begin

sublocale *min*:
  semilattice-order-ow U ‹$(\lambda x\ y.\ (with\ (\leq_{ow}) : «min»\ x\ y))$› ‹$(\leq_{ow})$› ‹$(<_{ow})$›
  ⟨*proof*⟩

sublocale *max*:
  semilattice-order-ow U ‹$(\lambda x\ y.\ (with\ (\leq_{ow}) : «max»\ x\ y))$› ‹$(\geq_{ow})$› ‹$(>_{ow})$›
  ⟨*proof*⟩

end

locale *ord-linorder-ow* =
  ord?: *ord-ow* U_1 le_1 ls_1 + *lo?*: *linorder-ow* U_2 le_2 ls_2
  for U_1 :: $'ao\ set$ **and** le_1 ls_1 **and** U_2 :: $'bo\ set$ **and** le_2 ls_2
begin

sublocale *ord-order-ow* ⟨*proof*⟩

end

locale *preorder-linorder-ow* =
  po?: *preorder-ow* U_1 le_1 ls_1 + *lo?*: *linorder-ow* U_2 le_2 ls_2
  for U_1 :: $'ao\ set$ **and** le_1 ls_1 **and** U_2 :: $'bo\ set$ **and** le_2 ls_2
begin

sublocale *preorder-order-ow* ⟨*proof*⟩

end

locale *order-linorder-ow* =
  order?: *order-ow* U_1 le_1 ls_1 + *lo?*: *linorder-ow* U_2 le_2 ls_2
  for U_1 :: $'ao\ set$ **and** le_1 ls_1 **and** U_2 :: $'bo\ set$ **and** le_2 ls_2
begin

sublocale *order-pair-ow* ⟨*proof*⟩

end

locale *linorder-pair-ow* =
  $lo_1?$: *linorder-ow* U_1 le_1 ls_1 + $lo_2?$: *linorder-ow* U_2 le_2 ls_2
  for U_1 :: $'ao\ set$ **and** le_1 ls_1 **and** U_2 :: $'bo\ set$ **and** le_2 ls_2
begin

sublocale *order-linorder-ow* ⟨*proof*⟩

end

Transfer rules

context

includes *lifting-syntax*
begin

lemma *linorder-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
(*linorder-ow* (*Collect* (*Domainp A*))) *class.linorder*
⟨*proof*⟩

end

Relativization

context *linorder-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *linorder-ow-axioms*
eliminating through *simp*
begin

tts-lemma *le-less-linear*:
assumes $x \in U$ **and** $y \in U$
shows $x \leq_{ow} y \vee y <_{ow} x$
is *linorder-class.le-less-linear*⟨*proof*⟩

tts-lemma *not-less*:
assumes $x \in U$ **and** $y \in U$
shows $(\neg\ x <_{ow} y) = (y \leq_{ow} x)$
is *linorder-class.not-less*⟨*proof*⟩

tts-lemma *not-le*:
assumes $x \in U$ **and** $y \in U$
shows $(\neg\ x \leq_{ow} y) = (y <_{ow} x)$
is *linorder-class.not-le*⟨*proof*⟩

tts-lemma *lessThan-minus-lessThan*:
assumes $n \in U$ **and** $m \in U$
shows $\{..<_{ow} n\} - \{..<_{ow} m\} = (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{m..<n\})$
is *linorder-class.lessThan-minus-lessThan*⟨*proof*⟩

tts-lemma *Ici-subset-Ioi-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(\{a..._{ow}\} \subseteq \{b<_{ow}...\}) = (b <_{ow} a)$
is *linorder-class.Ici-subset-Ioi-iff*⟨*proof*⟩

tts-lemma *Iic-subset-Iio-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(\{..._{ow} a\} \subseteq \{..<_{ow} b\}) = (a <_{ow} b)$
is *linorder-class.Iic-subset-Iio-iff*⟨*proof*⟩

tts-lemma *leI*:
assumes $x \in U$ **and** $y \in U$ **and** $\neg\ x <_{ow} y$
shows $y \leq_{ow} x$
is *linorder-class.leI*⟨*proof*⟩

tts-lemma *not-le-imp-less*:
assumes $y \in U$ **and** $x \in U$ **and** $\neg\ y \leq_{ow} x$
shows $x <_{ow} y$
is *linorder-class.not-le-imp-less⟨proof⟩*

tts-lemma *Int-atMost*:
assumes $a \in U$ **and** $b \in U$
shows $\{..{}_{ow}a\} \cap \{..{}_{ow}b\} = \{..{}_{ow}min\ a\ b\}$
is *linorder-class.Int-atMost⟨proof⟩*

tts-lemma *lessThan-Int-lessThan*:
assumes $a \in U$ **and** $b \in U$
shows $\{a<{}_{ow}..\} \cap \{b<{}_{ow}..\} = \{max\ a\ b<{}_{ow}..\}$
is *linorder-class.lessThan-Int-lessThan⟨proof⟩*

tts-lemma *greaterThan-Int-greaterThan*:
assumes $a \in U$ **and** $b \in U$
shows $\{..<{}_{ow}a\} \cap \{..<{}_{ow}b\} = \{..<{}_{ow}min\ a\ b\}$
is *linorder-class.greaterThan-Int-greaterThan⟨proof⟩*

tts-lemma *less-linear*:
assumes $x \in U$ **and** $y \in U$
shows $x <_{ow} y \vee x = y \vee y <_{ow} x$
is *linorder-class.less-linear⟨proof⟩*

tts-lemma *Int-atLeastAtMostR2*:
assumes $a \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{a..{}_{ow}\} \cap \{c..{}_{ow}d\} = \{max\ a\ c..{}_{ow}d\}$
is *linorder-class.Int-atLeastAtMostR2⟨proof⟩*

tts-lemma *Int-atLeastAtMostR1*:
assumes $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{..{}_{ow}b\} \cap \{c..{}_{ow}d\} = \{c..{}_{ow}min\ b\ d\}$
is *linorder-class.Int-atLeastAtMostR1⟨proof⟩*

tts-lemma *Int-atLeastAtMostL2*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $\{a..{}_{ow}b\} \cap \{c..{}_{ow}\} = \{max\ a\ c..{}_{ow}b\}$
is *linorder-class.Int-atLeastAtMostL2⟨proof⟩*

tts-lemma *Int-atLeastAtMostL1*:
assumes $a \in U$ **and** $b \in U$ **and** $d \in U$
shows $\{a..{}_{ow}b\} \cap \{..{}_{ow}d\} = \{a..{}_{ow}min\ b\ d\}$
is *linorder-class.Int-atLeastAtMostL1⟨proof⟩*

tts-lemma *neq-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(x \neq y) = (x <_{ow} y \vee y <_{ow} x)$
is *linorder-class.neq-iff⟨proof⟩*

tts-lemma *not-less-iff-gr-or-eq*:
assumes $x \in U$ **and** $y \in U$
shows $(\neg\ x <_{ow} y) = (y <_{ow} x \vee x = y)$
is *linorder-class.not-less-iff-gr-or-eq⟨proof⟩*

tts-lemma *max-min-distrib2*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$

shows *max a* (*min b c*) = *min* (*max a b*) (*max a c*)
is *linorder-class.max-min-distrib*2⟨*proof*⟩

tts-lemma *max-min-distrib*1:
assumes $b \in U$ **and** $c \in U$ **and** $a \in U$
shows *max* (*min b c*) *a* = *min* (*max b a*) (*max c a*)
is *linorder-class.max-min-distrib*1⟨*proof*⟩

tts-lemma *min-max-distrib*2:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows *min a* (*max b c*) = *max* (*min a b*) (*min a c*)
is *linorder-class.min-max-distrib*2⟨*proof*⟩

tts-lemma *min-max-distrib*1:
assumes $b \in U$ **and** $c \in U$ **and** $a \in U$
shows *min* (*max b c*) *a* = *max* (*min b a*) (*min c a*)
is *linorder-class.min-max-distrib*1⟨*proof*⟩

tts-lemma *atLeastAtMost-diff-ends*:
assumes $a \in U$ **and** $b \in U$
shows $\{a..{}_{ow}b\} - \{a, b\} = \{a<{}_{ow}..<{}_{ow}b\}$
is *linorder-class.atLeastAtMost-diff-ends*⟨*proof*⟩

tts-lemma *less-max-iff-disj*:
assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z <_{ow} max\ x\ y) = (z <_{ow} x \vee z <_{ow} y)$
is *linorder-class.less-max-iff-disj*⟨*proof*⟩

tts-lemma *min-less-iff-conj*:
assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z <_{ow} min\ x\ y) = (z <_{ow} x \wedge z <_{ow} y)$
is *linorder-class.min-less-iff-conj*⟨*proof*⟩

tts-lemma *max-less-iff-conj*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(max\ x\ y <_{ow} z) = (x <_{ow} z \wedge y <_{ow} z)$
is *linorder-class.max-less-iff-conj*⟨*proof*⟩

tts-lemma *min-less-iff-disj*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(min\ x\ y <_{ow} z) = (x <_{ow} z \vee y <_{ow} z)$
is *linorder-class.min-less-iff-disj*⟨*proof*⟩

tts-lemma *le-max-iff-disj*:
assumes $z \in U$ **and** $x \in U$ **and** $y \in U$
shows $(z \leq_{ow} max\ x\ y) = (z \leq_{ow} x \vee z \leq_{ow} y)$
is *linorder-class.le-max-iff-disj*⟨*proof*⟩

tts-lemma *min-le-iff-disj*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$
shows $(min\ x\ y \leq_{ow} z) = (x \leq_{ow} z \vee y \leq_{ow} z)$
is *linorder-class.min-le-iff-disj*⟨*proof*⟩

tts-lemma *antisym-conv*3:
assumes $y \in U$ **and** $x \in U$ **and** $\neg\ y <_{ow} x$
shows $(\neg\ x <_{ow} y) = (x = y)$
is *linorder-class.antisym-conv*3⟨*proof*⟩

tts-lemma *Int-atLeastAtMost*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{a..{}_{ow}b\} \cap \{c..{}_{ow}d\} = \{max\ a\ c..{}_{ow}min\ b\ d\}$
is *linorder-class.Int-atLeastAtMost⟨proof⟩*

tts-lemma *Int-atLeastLessThan*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows
$(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) \cap (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{c..<d\}) =$
$(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{(max\ a\ c)..<(min\ b\ d)\})$
is *linorder-class.Int-atLeastLessThan⟨proof⟩*

tts-lemma *Int-greaterThanAtMost*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{a<{}_{ow}..b\} \cap \{c<{}_{ow}..d\} = \{max\ a\ c<{}_{ow}..min\ b\ d\}$
is *linorder-class.Int-greaterThanAtMost⟨proof⟩*

tts-lemma *Int-greaterThanLessThan*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $\{a<{}_{ow}..<{}_{ow}b\} \cap \{c<{}_{ow}..<{}_{ow}d\} = \{max\ a\ c<{}_{ow}..<{}_{ow}min\ b\ d\}$
is *linorder-class.Int-greaterThanLessThan⟨proof⟩*

tts-lemma *le-cases*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_{ow} y \Longrightarrow P$ **and** $y \leq_{ow} x \Longrightarrow P$
shows P
is *linorder-class.le-cases⟨proof⟩*

tts-lemma *split-max*:
assumes $i \in U$ **and** $j \in U$
shows $P\ (max\ i\ j) = ((i \leq_{ow} j \longrightarrow P\ j) \wedge (\neg\ i \leq_{ow} j \longrightarrow P\ i))$
is *linorder-class.split-max⟨proof⟩*

tts-lemma *split-min*:
assumes $i \in U$ **and** $j \in U$
shows $P\ (min\ i\ j) = ((i \leq_{ow} j \longrightarrow P\ i) \wedge (\neg\ i \leq_{ow} j \longrightarrow P\ j))$
is *linorder-class.split-min⟨proof⟩*

tts-lemma *Ioc-subset-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<{}_{ow}..b\} \subseteq \{c<{}_{ow}..d\}) = (b \leq_{ow} a \vee b \leq_{ow} d \wedge c \leq_{ow} a)$
is *linorder-class.Ioc-subset-iff⟨proof⟩*

tts-lemma *atLeastLessThan-subset-iff*:
assumes $a \in U$
and $b \in U$
and $c \in U$
and $d \in U$
and $(on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{a..<b\}) \subseteq (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{c..<d\})$
shows $b \leq_{ow} a \vee b \leq_{ow} d \wedge c \leq_{ow} a$
is *linorder-class.atLeastLessThan-subset-iff⟨proof⟩*

tts-lemma *Ioc-inj*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<{}_{ow}..b\} = \{c<{}_{ow}..d\}) = (b \leq_{ow} a \wedge d \leq_{ow} c \vee a = c \wedge b = d)$
is *linorder-class.Ioc-inj⟨proof⟩*

tts-lemma *neqE*:
assumes $x \in U$

and $y \in U$
and $x \neq y$
and $x <_{ow} y \Longrightarrow R$
and $y <_{ow} x \Longrightarrow R$
shows R
is *linorder-class.neqE*⟨*proof*⟩

tts-lemma *Ioc-disjoint*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows
$(\{a<_{ow}..b\} \cap \{c<_{ow}..d\} = \{\}) = (b \leq_{ow} a \vee d \leq_{ow} c \vee b \leq_{ow} c \vee d \leq_{ow} a)$
is *linorder-class.Ioc-disjoint*⟨*proof*⟩

tts-lemma *linorder-cases*:
assumes $x \in U$
and $y \in U$
and $x <_{ow} y \Longrightarrow P$
and $x = y \Longrightarrow P$
and $y <_{ow} x \Longrightarrow P$
shows P
is *linorder-class.linorder-cases*⟨*proof*⟩

tts-lemma *le-cases*3:
assumes $x \in U$
and $y \in U$
and $z \in U$
and $[\![x \leq_{ow} y; y \leq_{ow} z]\!] \Longrightarrow P$
and $[\![y \leq_{ow} x; x \leq_{ow} z]\!] \Longrightarrow P$
and $[\![x \leq_{ow} z; z \leq_{ow} y]\!] \Longrightarrow P$
and $[\![z \leq_{ow} y; y \leq_{ow} x]\!] \Longrightarrow P$
and $[\![y \leq_{ow} z; z \leq_{ow} x]\!] \Longrightarrow P$
and $[\![z \leq_{ow} x; x \leq_{ow} y]\!] \Longrightarrow P$
shows P
is *linorder-class.le-cases*3⟨*proof*⟩

end

end

3.15.2 Dense linear orders

Definitions and further properties

locale *dense-linorder-ow = linorder-ow U le ls + dense-order-ow U le ls*
for $U :: 'ao\ set$ **and** *le* (**infix** ‹$\leq_{ow}$› 50) **and** *ls* (**infix** ‹$<_{ow}$› 50)

Transfer rules

context
includes *lifting-syntax*
begin

lemma *dense-linorder-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
(*dense-linorder-ow* (*Collect* (*Domainp A*))) *class.dense-linorder*
⟨*proof*⟩

end

Relativization

context *dense-linorder-ow*
begin

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *dense-linorder-ow-axioms*
eliminating through *simp*
begin

tts-lemma *infinite-Icc*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows *infinite* $\{a..{}_{ow}b\}$
is *dense-linorder-class.infinite-Icc*⟨*proof*⟩

tts-lemma *infinite-Ico*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows *infinite* (*on* U *with* $(\leq_{ow})$ $(<_{ow})$: $\{a..<b\}$)
is *dense-linorder-class.infinite-Ico*⟨*proof*⟩

tts-lemma *infinite-Ioc*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows *infinite* $\{a<{}_{ow}..b\}$
is *dense-linorder-class.infinite-Ioc*⟨*proof*⟩

tts-lemma *infinite-Ioo*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_{ow} b$
shows *infinite* $\{a<{}_{ow}..<{}_{ow}b\}$
is *dense-linorder-class.infinite-Ioo*⟨*proof*⟩

tts-lemma *atLeastLessThan-subseteq-atLeastAtMost-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows
((*on* U *with* $(\leq_{ow})$ $(<_{ow})$: $\{a..<b\}$) $\subseteq$ $\{c..{}_{ow}d\}$) =
$(a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.atLeastLessThan-subseteq-atLeastAtMost-iff*⟨*proof*⟩

tts-lemma *greaterThanAtMost-subseteq-atLeastAtMost-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<{}_{ow}..b\} \subseteq \{c..{}_{ow}d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.greaterThanAtMost-subseteq-atLeastAtMost-iff*⟨*proof*⟩

tts-lemma *greaterThanAtMost-subseteq-atLeastLessThan-iff*:
assumes $a \in U$
and $b \in U$
and $c \in U$
and $d \in U$
shows $(\{a<{}_{ow}..b\} \subseteq (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{c..<d\})) =$
$(a <_{ow} b \longrightarrow b <_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.greaterThanAtMost-subseteq-atLeastLessThan-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-subseteq-atLeastAtMost-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<{}_{ow}..<{}_{ow}b\} \subseteq \{c..{}_{ow}d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$

is *dense-linorder-class.greaterThanLessThan-subseteq-atLeastAtMost-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-subseteq-atLeastLessThan-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows
$(\{a<_{ow}..<_{ow}b\} \subseteq (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{c..<d\})) =$
$(a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.greaterThanLessThan-subseteq-atLeastLessThan-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-subseteq-greaterThanAtMost-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<_{ow}..<_{ow}b\} \subseteq \{c<_{ow}..d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.greaterThanLessThan-subseteq-greaterThanAtMost-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-subseteq-greaterThanLessThan*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a<_{ow}..<_{ow}b\} \subseteq \{c<_{ow}..<_{ow}d\}) = (a <_{ow} b \longrightarrow b \leq_{ow} d \wedge c \leq_{ow} a)$
is *dense-linorder-class.greaterThanLessThan-subseteq-greaterThanLessThan*⟨*proof*⟩

end

end

3.16 Relativization of the results about simple topological spaces

3.16.1 Definitions and common properties

Some of the entities that are presented in this subsection were copied from the theory *HOL–Types-To-Sets/Examples/T2-Spaces*.

locale *topological-space-ow* =
  fixes $U :: 'at\ set$ **and** $\tau :: 'at\ set \Rightarrow bool$
  assumes *open-UNIV*[*simp*, *intro*]: $\tau\ U$
  assumes *open-Int*[*intro*]:
    $[\![\ S \subseteq U;\ T \subseteq U;\ \tau\ S;\ \tau\ T\ ]\!] \Longrightarrow \tau\ (S \cap T)$
  assumes *open-Union*[*intro*]:
    $[\![\ K \subseteq Pow\ U;\ \forall\ S{\in}K.\ \tau\ S\ ]\!] \Longrightarrow \tau\ (\bigcup K)$
begin

context
  includes *lifting-syntax*
begin

tts-register-sbts τ | U
⟨*proof*⟩

end

end

locale *topological-space-pair-ow* =
  ts_1: *topological-space-ow* $U_1\ \tau_1$ + ts_2: *topological-space-ow* $U_2\ \tau_2$
  for $U_1 :: 'at\ set$ **and** τ_1 **and** $U_2 :: 'bt\ set$ **and** τ_2

locale *topological-space-triple-ow* =
  ts_1: *topological-space-ow* $U_1\ \tau_1$ +
  ts_2: *topological-space-ow* $U_2\ \tau_2$ +
  ts_3: *topological-space-ow* $U_3\ \tau_3$
  for $U_1 :: 'at\ set$ **and** τ_1
    and $U_2 :: 'bt\ set$ **and** τ_2
    and $U_3 :: 'ct\ set$ **and** τ_3
begin

sublocale tsp_{12}: *topological-space-pair-ow* $U_1\ \tau_1\ U_2\ \tau_2$ ⟨*proof*⟩
sublocale tsp_{13}: *topological-space-pair-ow* $U_1\ \tau_1\ U_3\ \tau_3$ ⟨*proof*⟩
sublocale tsp_{23}: *topological-space-pair-ow* $U_2\ \tau_2\ U_3\ \tau_3$ ⟨*proof*⟩
sublocale tsp_{21}: *topological-space-pair-ow* $U_2\ \tau_2\ U_1\ \tau_1$ ⟨*proof*⟩
sublocale tsp_{31}: *topological-space-pair-ow* $U_3\ \tau_3\ U_1\ \tau_1$ ⟨*proof*⟩
sublocale tsp_{32}: *topological-space-pair-ow* $U_3\ \tau_3\ U_2\ \tau_2$ ⟨*proof*⟩

end

inductive *generate-topology-on* :: [$'at\ set\ set$, $'at\ set$, $'at\ set$] ⇒ *bool*
  (
    ‹(*in'-topology'-generated'-by - on -* : «*open*» -)›
    [1000, 1000, 1000] 10
)
  for $S :: 'at\ set\ set$
  where
    UNIV: (*in-topology-generated-by S on U* : «*open*» *U*)

```
  | Int: (in-topology-generated-by S on U : «open» (a ∩ b))
    if (in-topology-generated-by S on U : «open» a)
      and (in-topology-generated-by S on U : «open» b)
      and a ⊆ U
      and b ⊆ U
  | UN: (in-topology-generated-by S on U : «open» (⋃ K))
    if K ⊆ Pow U
      and (⋀k. k ∈ K ⟹ (in-topology-generated-by S on U : «open» k))
  | Basis: (in-topology-generated-by S on U : «open» s)
      if s ∈ S and s ⊆ U

lemma gto-imp-ss: (in-topology-generated-by S on U : «open» A) ⟹ A ⊆ U
  ⟨proof⟩

lemma gt-eq-gto: generate-topology = (λS. generate-topology-on S UNIV)
⟨proof⟩

ud ‹topological-space.closed› (‹(with - : «closed» -)› [1000, 1000] 10)
ud closed' ‹closed›
ud ‹topological-space.compact› (‹(with - : «compact» -)› [1000, 1000] 10)
ud compact' ‹compact›
ud ‹topological-space.connected› (‹(with - : «connected» -)› [1000, 1000] 10)
ud connected' ‹connected›
ud ‹topological-space.islimpt› (‹(with - : - «islimpt» -)› [1000, 1000, 1000] 60)
ud islimpt' ‹topological-space-class.islimpt›
ud ‹interior› (‹(with - : «interior» -)› [1000, 1000] 10)
ud ‹closure› (‹(with - : «closure» -)› [1000, 1000] 10)
ud ‹frontier› (‹(with - : «frontier» -)› [1000, 1000] 10)
ud ‹countably-compact› (‹(with - : «countably'-compact» -)› [1000, 1000] 10)

definition topological-basis-with :: ['a set ⇒ bool, 'a set set] ⇒ bool
  (‹(with - : «topological'-basis» -)› [1000, 1000] 10)
  where
  (with τ : «topological-basis» B) =
    (⋃ B = UNIV ∧ (∀ b ∈ B. τ b) ∧ (∀ q. τ q ⟶ (∃ B'⊆B. ⋃ B' = q)))

ctr relativization
  synthesis ctr-simps
  assumes [transfer-domain-rule, transfer-rule]: Domainp A = (λx. x ∈ U)
    and [transfer-rule]: bi-unique A right-total A
  trp (?'a A)
  in closed-ow: closed.with-def
    (‹(on - with - : «closed» -)› [1000, 1000] 10)
    and compact-ow: compact.with-def
      (‹(on - with - : «compact» -)› [1000, 1000, 1000] 10)
    and connected-ow: connected.with-def
      (‹(on - with - : «connected» -)› [1000, 1000, 1000] 10)
    and islimpt-ow: islimpt.with-def
      (‹(on - with - : - «islimpt» -)› [1000, 1000, 1000, 1000] 10)
    and interior-ow: interior.with-def
      (‹(on - with - : «interior» -)› [1000, 1000, 1000] 10)
    and closure-ow: closure.with-def
      (‹(on - with - : «closure» -)› [1000, 1000, 1000] 10)
    and frontier-ow: frontier.with-def
      (‹(on - with - : «frontier» -)› [1000, 1000, 1000] 10)
    and countably-compact-ow: countably-compact.with-def
      (‹(on - with - : «countably'-compact» -)› [1000, 1000, 1000] 10)
```

context *topological-space-ow*
begin

abbreviation *closed* **where** *closed* ≡ *closed-ow U τ*
abbreviation *compact* **where** *compact* ≡ *compact-ow U τ*
abbreviation *connected* **where** *connected* ≡ *connected-ow U τ*
abbreviation *islimpt* (**infixr** ‹«*islimpt*»› 60)
where *x* «*islimpt*» *S* ≡ *on U with τ* : *x* «*islimpt*» *S*
abbreviation *interior* **where** *interior* ≡ *interior-ow U τ*
abbreviation *closure* **where** *closure* ≡ *closure-ow U τ*
abbreviation *frontier* **where** *frontier* ≡ *frontier-ow U τ*
abbreviation *countably-compact*
where *countably-compact* ≡ *countably-compact-ow U τ*

end

context
includes *lifting-syntax*
begin

private lemma *Domainp-fun-rel-eq-subset*:
fixes *A* :: [′*a*, ′*c*] ⇒ *bool*
fixes *B* :: [′*b*, ′*d*] ⇒ *bool*
assumes *bi-unique A bi-unique B*
shows
Domainp (*A* ===> *B*) =
(λ*f*. *f* ‘ (*Collect* (*Domainp A*)) ⊆ (*Collect* (*Domainp B*)))
⟨*proof*⟩ **lemma** *Ex-rt-bu-transfer*[*transfer-rule*]:
fixes *A* :: [′*a*, ′*c*] ⇒ *bool*
fixes *B* :: [′*b*, ′*d*] ⇒ *bool*
assumes [*transfer-rule*]: *bi-unique A right-total A bi-unique B*
shows
(((*B* ===> *A*) ===> (=)) ===> (=))
(*Bex* (*Collect* (λ*f*. *f* ‘ (*Collect* (*Domainp B*)) ⊆ (*Collect* (*Domainp A*)))))
Ex
⟨*proof*⟩

end

definition *topological-basis-ow* ::
[′*a set*, ′*a set* ⇒ *bool*, ′*a set set*] ⇒ *bool*
(‹(*on - with - :* «*topological′-basis*» -)› [1000, 1000, 1000] 10)
where
(*on U with τ* : «*topological-basis*» *B*) =
(⋃ *B* = *U* ∧ (∀ *b* ∈ *B*. *τ b*) ∧ (∀ *q* ⊆ *U*. *τ q* ⟶ (∃ *B′*⊆ *B*. ⋃ *B′* = *q*)))

context *topological-space*
begin

lemma *topological-basis-with*[*ud-with*]:
topological-basis = *topological-basis-with open*
⟨*proof*⟩

end

3.16.2 Transfer rules

Some of the entities that are presented in this subsectionwere copied from *HOL−Types-To-Sets/Examples/T2-Sp*

context
includes *lifting-syntax*
begin

lemmas *vimage-transfer*[*transfer-rule*] = *vimage-transfer*

lemma *topological-space-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*rel-set A* ===> (=)) ===> (=))
(*topological-space-ow* (*Collect* (*Domainp A*))) *class.topological-space*
⟨*proof*⟩

lemma *generate-topology-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*rel-set* (*rel-set A*)) ===> (*rel-set A* ===> (=)))
(λ*B*. *generate-topology-on B* (*Collect* (*Domainp A*))) *generate-topology*
⟨*proof*⟩

lemma *topological-basis-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*rel-set A* ===> (=)) ===> (*rel-set* (*rel-set A*)) ===> (=))
(*topological-basis-ow* (*Collect* (*Domainp A*))) *topological-basis-with*
⟨*proof*⟩

end

3.16.3 Relativization

tts-context
tts: (*?'a* **to** ‹U_1::'*a set*›) **and** (*?'b* **to** ‹U_2::'*b set*›)
rewriting *ctr-simps*
begin

tts-lemma *generate-topology-Union*:
assumes $U_1 \neq \{\}$
and $U_2 \neq \{\}$
and $I \subseteq U_1$
and $S \subseteq Pow\ U_2$
and $\forall\, x{\in}U_1.\ K\ (x{::}'a) \subseteq U_2$
and
$\bigwedge k.\ [\![k \in U_1;\ k \in I]\!] \Longrightarrow$
in-topology-generated-by S on U_2 : «*open*» (*K k*)
shows *in-topology-generated-by S on* U_2 : «*open*» ($\bigcup$ (*K* ‘ *I*))
is *generate-topology-Union*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** ‹*U*::'*a set*›)
rewriting *ctr-simps*
eliminating through
(*unfold topological-space-ow-def*; *auto intro*: *generate-topology-on.intros*)
begin

tts-lemma *topological-space-generate-topology*:

shows *topological-space-ow U* (*generate-topology-on S U*)
is *topological-space-generate-topology*⟨*proof*⟩

end

context *topological-space-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through (*metis open-UNIV*)
begin

tts-lemma *open-empty*[*simp*]:
shows *τ* {}
is *topological-space-class.open-empty*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through
(
unfold
closed-ow-def
compact-ow-def
connected-ow-def
interior-ow-def
closure-ow-def
frontier-ow-def,
auto
)
begin

tts-lemma *closed-empty*[*simp*]: *closed* {}
is *topological-space-class.closed-empty*⟨*proof*⟩

tts-lemma *closed-UNIV*[*simp*]: *closed U*
is *topological-space-class.closed-UNIV*⟨*proof*⟩

tts-lemma *compact-empty*[*simp*]: *compact* {}
is *topological-space-class.compact-empty*⟨*proof*⟩

tts-lemma *connected-empty*[*simp*]: *connected* {}
is *topological-space-class.connected-empty*⟨*proof*⟩

tts-lemma *interior-empty*[*simp*]: *interior* {} = {}
is *interior-empty*⟨*proof*⟩

tts-lemma *closure-empty*[*simp*]: *closure* {} = {}
is *closure-empty*⟨*proof*⟩

tts-lemma *closure-UNIV*[*simp*]: *closure U* = *U*
is *closure-UNIV*⟨*proof*⟩

tts-lemma *frontier-empty*[*simp*]: *frontier* {} = {}
is *frontier-empty*⟨*proof*⟩

tts-lemma *frontier-UNIV*[*simp*]: *frontier U* = {}
is *frontier-UNIV*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through (*auto simp*: *UNIV inj-on-def*)
begin

tts-lemma *connected-Union*:
assumes $S \subseteq Pow\ U$ **and** $\bigwedge s.\ s \in S \Longrightarrow$ *connected s* **and** $\bigcap\ S \cap U \neq \{\}$
shows *connected* $(\bigcup\ S)$
given *Topological-Spaces.connected-Union*
⟨*proof*⟩

tts-lemma *connected-Un*:
assumes $s \subseteq U$
and $t \subseteq U$
and *connected s*
and *connected t*
and $s \cap t \neq \{\}$
shows *connected* $(s \cup t)$
is *Topological-Spaces.connected-Un*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **and** ‹*?A* ⊆ *?B*›
through (*auto simp*: *UNIV inj-on-def*)
begin

tts-lemma *connected-sing*:
assumes $x \in U$
shows *connected* $\{x\}$
is *topological-space-class.connected-sing*⟨*proof*⟩

tts-lemma *topological-basisE*:
assumes $B \subseteq Pow\ U$
and $O' \subseteq U$
and $x \in U$
and *on U with* τ : «*topological-basis*» *B*
and $\tau\ O'$
and $x \in O'$
and $\bigwedge B'.\ [\![B' \subseteq U;\ B' \in B;\ x \in B';\ B' \subseteq O']\!] \Longrightarrow$ *thesis*
shows *thesis*
is *topological-space-class.topological-basisE*⟨*proof*⟩

tts-lemma *islimptE*:

assumes $x \in U$
and $S \subseteq U$
and $T \subseteq U$
and x «*islimpt*» S
and $x \in T$
and $\tau\ T$
and $\bigwedge y.\ [\![y \in U;\ y \in S;\ y \in T;\ y \neq x]\!] \Longrightarrow thesis$
shows *thesis*
is *Elementary-Topology.islimptE*⟨*proof*⟩

tts-lemma *islimpt-subset*:
assumes $x \in U$ **and** $T \subseteq U$ **and** x «*islimpt*» S **and** $S \subseteq T$
shows x «*islimpt*» T
is *Elementary-Topology.islimpt-subset*⟨*proof*⟩

tts-lemma *islimpt-UNIV-iff*:
assumes $x \in U$
shows x «*islimpt*» $U = (\neg\ \tau\ \{x\})$
is *Elementary-Topology.islimpt-UNIV-iff*⟨*proof*⟩

tts-lemma *islimpt-punctured*:
assumes $x \in U$ **and** $S \subseteq U$
shows x «*islimpt*» $S = x$ «*islimpt*» $S - \{x\}$
is *Elementary-Topology.islimpt-punctured*⟨*proof*⟩

tts-lemma *islimpt-EMPTY*:
assumes $x \in U$
shows $\neg\ x$ «*islimpt*» $\{\}$
is *Elementary-Topology.islimpt-EMPTY*⟨*proof*⟩

tts-lemma *islimpt-Un*:
assumes $x \in U$ **and** $S \subseteq U$ **and** $T \subseteq U$
shows x «*islimpt*» $S \cup T = (x$ «*islimpt*» $S \vee x$ «*islimpt*» $T)$
is *Elementary-Topology.islimpt-Un*⟨*proof*⟩

tts-lemma *interiorI*:
assumes $x \in U$ **and** $S \subseteq U$ **and** $\tau\ T$ **and** $x \in T$ **and** $T \subseteq S$
shows $x \in interior\ S$
is *Elementary-Topology.interiorI*⟨*proof*⟩

tts-lemma *islimpt-in-closure*:
assumes $x \in U$ **and** $S \subseteq U$
shows x «*islimpt*» $S = (x \in closure\ (S - \{x\}))$
is *Elementary-Topology.islimpt-in-closure*⟨*proof*⟩

tts-lemma *compact-sing*:
assumes $a \in U$
shows *compact* $\{a\}$
is *Elementary-Topology.compact-sing*⟨*proof*⟩

tts-lemma *compact-insert*:
assumes $s \subseteq U$ **and** $x \in U$ **and** *compact* s
shows *compact* $(insert\ x\ s)$
is *Elementary-Topology.compact-insert*⟨*proof*⟩

tts-lemma *open-Un*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\tau\ S$ **and** $\tau\ T$
shows $\tau\ (S \cup T)$

is *topological-space-class.open-Un⟨proof⟩*

tts-lemma *open-Inter*:
assumes $S \subseteq Pow\ U$ **and** *finite* S **and** $Ball\ S\ \tau$
shows $\tau\ (\bigcap\ S \cap U)$
is *topological-space-class.open-Inter⟨proof⟩*

tts-lemma *openI*:
assumes $S \subseteq U$ **and** $\bigwedge x.\ [\![x \in U;\ x \in S]\!] \Longrightarrow \exists\, y{\subseteq}U.\ \tau\ y \wedge y \subseteq S \wedge x \in y$
shows $\tau\ S$
given *topological-space-class.openI ⟨proof⟩*

tts-lemma *closed-Un*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** *closed* T
shows *closed* $(S \cup T)$
is *topological-space-class.closed-Un⟨proof⟩*

tts-lemma *closed-Int*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** *closed* T
shows *closed* $(S \cap T)$
is *topological-space-class.closed-Int⟨proof⟩*

tts-lemma *open-Collect-conj*:
assumes $\tau\ \{x.\ P\ x \wedge x \in U\}$ **and** $\tau\ \{x.\ Q\ x \wedge x \in U\}$
shows $\tau\ \{x \in U.\ P\ x \wedge Q\ x\}$
is *topological-space-class.open-Collect-conj⟨proof⟩*

tts-lemma *open-Collect-disj*:
assumes $\tau\ \{x.\ P\ x \wedge x \in U\}$
and $\tau\ \{x.\ Q\ x \wedge x \in U\}$
shows $\tau\ \{x \in U.\ P\ x \vee Q\ x\}$
is *topological-space-class.open-Collect-disj⟨proof⟩*

tts-lemma *open-Collect-imp*:
assumes *closed* $\{x.\ P\ x \wedge x \in U\}$
and $\tau\ \{x.\ Q\ x \wedge x \in U\}$
shows $\tau\ \{x \in U.\ P\ x \longrightarrow Q\ x\}$
is *topological-space-class.open-Collect-imp⟨proof⟩*

tts-lemma *open-Collect-const*: $\tau\ \{x.\ P \wedge x \in U\}$
is *topological-space-class.open-Collect-const⟨proof⟩*

tts-lemma *closed-Collect-conj*:
assumes *closed* $\{x.\ P\ x \wedge x \in U\}$ **and** *closed* $\{x.\ Q\ x \wedge x \in U\}$
shows *closed* $\{x \in U.\ P\ x \wedge Q\ x\}$
is *topological-space-class.closed-Collect-conj⟨proof⟩*

tts-lemma *closed-Collect-disj*:
assumes *closed* $\{x.\ P\ x \wedge x \in U\}$ **and** *closed* $\{x.\ Q\ x \wedge x \in U\}$
shows *closed* $\{x \in U.\ P\ x \vee Q\ x\}$
is *topological-space-class.closed-Collect-disj⟨proof⟩*

tts-lemma *closed-Collect-imp*:
assumes $\tau\ \{x.\ P\ x \wedge x \in U\}$ **and** *closed* $\{x.\ Q\ x \wedge x \in U\}$
shows *closed* $\{x \in U.\ P\ x \longrightarrow Q\ x\}$
is *topological-space-class.closed-Collect-imp⟨proof⟩*

tts-lemma *compact-Int-closed*:

assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *compact* S **and** *closed* T
shows *compact* $(S \cap T)$
is *topological-space-class.compact-Int-closed*⟨*proof*⟩

tts-lemma *compact-diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *compact* S **and** $\tau\ T$
shows *compact* $(S - T)$
is *topological-space-class.compact-diff*⟨*proof*⟩

tts-lemma *connectedD*:
assumes $U \subseteq U$
and $V \subseteq U$
and *connected* A
and $\tau\ U$
and $\tau\ V$
and $U \cap (V \cap A) = \{\}$
and $A \subseteq U \cup V$
shows $U \cap A = \{\} \vee V \cap A = \{\}$
is *topological-space-class.connectedD*⟨*proof*⟩

tts-lemma *topological-basis-open*:
assumes $B \subseteq$ *Pow* U **and** *on* U *with* τ : «*topological-basis*» B **and** $X \in B$
shows $\tau\ X$
is *topological-space-class.topological-basis-open*⟨*proof*⟩

tts-lemma *topological-basis-imp-subbasis*:
assumes $B \subseteq$ *Pow* U **and** *on* U *with* τ : «*topological-basis*» B
shows $\forall\ s{\subseteq}U.\ \tau\ s = (\textit{in-topology-generated-by}\ B\ \textit{on}\ U : \text{«}\textit{open}\text{»}\ s)$
is *topological-space-class.topological-basis-imp-subbasis*⟨*proof*⟩

tts-lemma *connected-closedD*:
assumes $A \subseteq U$
and $B \subseteq U$
and *connected* s
and $A \cap (B \cap s) = \{\}$
and $s \subseteq A \cup B$
and *closed* A
and *closed* B
shows $A \cap s = \{\} \vee B \cap s = \{\}$
is *Topological-Spaces.connected-closedD*⟨*proof*⟩

tts-lemma *connected-diff-open-from-closed*:
assumes $u \subseteq U$
and $s \subseteq t$
and $t \subseteq u$
and $\tau\ s$
and *closed* t
and *connected* u
and *connected* $(t - s)$
shows *connected* $(u - s)$
is *Topological-Spaces.connected-diff-open-from-closed*⟨*proof*⟩

tts-lemma *interior-maximal*:
assumes $S \subseteq U$ **and** $T \subseteq S$ **and** $\tau\ T$
shows $T \subseteq$ *interior* S
is *Elementary-Topology.interior-maximal*⟨*proof*⟩

tts-lemma *open-subset-interior*:

assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\tau\ S$
shows $(S \subseteq \mathit{interior}\ T) = (S \subseteq T)$
is *Elementary-Topology.open-subset-interior*⟨*proof*⟩

tts-lemma *interior-mono*:
assumes $T \subseteq U$ **and** $S \subseteq T$
shows $\mathit{interior}\ S \subseteq \mathit{interior}\ T$
is *Elementary-Topology.interior-mono*⟨*proof*⟩

tts-lemma *interior-Int*:
assumes $S \subseteq U$ **and** $T \subseteq U$
shows $\mathit{interior}\ (S \cap T) = \mathit{interior}\ S \cap \mathit{interior}\ T$
is *Elementary-Topology.interior-Int*⟨*proof*⟩

tts-lemma *interior-closed-Un-empty-interior*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\mathit{closed}\ S$ **and** $\mathit{interior}\ T = \{\}$
shows $\mathit{interior}\ (S \cup T) = \mathit{interior}\ S$
is *Elementary-Topology.interior-closed-Un-empty-interior*⟨*proof*⟩

tts-lemma *countably-compact-imp-acc-point*:
assumes *local.countably-compact s*
and *countable t*
and *infinite t*
and $t \subseteq s$
shows $\exists\, x{\in}s.\ \forall\, U{\in}\mathit{Pow}\ U.\ \tau\ U \wedge x \in U \longrightarrow \mathit{infinite}\ (U \cap t)$
is *Elementary-Topology.countably-compact-imp-acc-point*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}›
through (*auto simp*: *UNIV inj-on-def*)
begin

tts-lemma *first-countableI*:
assumes $\mathcal{A} \subseteq \mathit{Pow}\ U$
and $x \in U$
and *countable* $\mathcal{A}$
and $\bigwedge A.\ [\![A \in \mathcal{A}]\!] \Longrightarrow x \in A$
and $\bigwedge A.\ [\![A \in \mathcal{A}]\!] \Longrightarrow \tau\ A$
and $\bigwedge S.\ [\![\tau\ S;\ x \in S]\!] \Longrightarrow \exists\, A{\in}\mathcal{A}.\ A \subseteq S$
shows $\exists\, \mathcal{A}{\in}\{f.\ \mathit{range}\ f \subseteq \mathit{Pow}\ U\}.$
$(\forall\, i.\ \tau\ (\mathcal{A}\ (i{::}\mathit{nat})) \wedge x \in \mathcal{A}\ i) \wedge$
$(\forall\, S{\in}\mathit{Pow}\ U.\ \tau\ S \wedge x \in S \longrightarrow (\exists\, i.\ \mathcal{A}\ i \subseteq S))$
given *topological-space-class.first-countableI* ⟨*proof*⟩

tts-lemma *islimptI*:
assumes $x \in U$
and $S \subseteq U$
and $\bigwedge T.\ [\![x \in T;\ \tau\ T]\!] \Longrightarrow \exists\, y{\in}S.\ y \in T \wedge y \neq x$
shows x «*islimpt*» S
given *Elementary-Topology.islimptI*
⟨*proof*⟩

tts-lemma *interiorE*:

assumes $x \in U$
and $S \subseteq U$
and $x \in \mathit{interior}\ S$
and $\bigwedge T.\ [\![T \subseteq U;\ \tau\ T;\ x \in T;\ T \subseteq S]\!] \Longrightarrow \mathit{thesis}$
shows *thesis*
is *Elementary-Topology.interiorE*⟨*proof*⟩

tts-lemma *closure-iff-nhds-not-empty*:
assumes $x \in U$ **and** $X \subseteq U$
shows
$(x \in \mathit{closure}\ X) =$
$(\forall\, y{\subseteq}U.\ \forall\, z{\subseteq}U.\ z \subseteq y \longrightarrow \tau\ z \longrightarrow x \in z \longrightarrow X \cap y \neq \{\})$
given *Elementary-Topology.closure-iff-nhds-not-empty* ⟨*proof*⟩

tts-lemma *basis-dense*:
assumes $B \subseteq \mathit{Pow}\ U$
and $\forall\, x{\subseteq}U.\ f\ x \in U$
and *on U with* τ : «*topological-basis*» B
and $\bigwedge B'.\ [\![B' \subseteq U;\ B' \neq \{\}]\!] \Longrightarrow f\ B' \in B'$
shows $\forall\, x{\subseteq}U.\ \tau\ x \longrightarrow x \neq \{\} \longrightarrow (\exists\, y{\in}B.\ f\ y \in x)$
given *topological-space-class.basis-dense* ⟨*proof*⟩

tts-lemma *inj-setminus*:
assumes $A \subseteq \mathit{Pow}\ U$
shows *inj-on* $(\lambda S.\ -\ S \cap U)\ A$
is *topological-space-class.inj-setminus*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **and** ‹*?S* ⊆ *U*› **through**
(
unfold
closed-ow-def
compact-ow-def
connected-ow-def
interior-ow-def
topological-basis-ow-def
closure-ow-def
frontier-ow-def
countably-compact-ow-def,
auto
)
begin

tts-lemma *closed-Inter*:
assumes $K \subseteq \mathit{Pow}\ U$ **and** *Ball K closed*
shows *closed* $(\bigcap\ K \cap U)$
is *topological-space-class.closed-Inter*⟨*proof*⟩

tts-lemma *closed-Union*:
assumes $S \subseteq \mathit{Pow}\ U$ **and** *finite S* **and** *Ball S closed*
shows *closed* $(\bigcup\ S)$
is *topological-space-class.closed-Union*⟨*proof*⟩

tts-lemma *open-closed*:
assumes $S \subseteq U$
shows $\tau\ S = closed\ (-\ S \cap U)$
is *topological-space-class.open-closed*⟨*proof*⟩

tts-lemma *closed-open*:
shows $closed\ S = \tau\ (-\ S \cap U)$
is *topological-space-class.closed-open*⟨*proof*⟩

tts-lemma *open-Diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\tau\ S$ **and** $closed\ T$
shows $\tau\ (S - T)$
is *topological-space-class.open-Diff*⟨*proof*⟩

tts-lemma *closed-Diff*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $closed\ S$ **and** $\tau\ T$
shows $closed\ (S - T)$
is *topological-space-class.closed-Diff*⟨*proof*⟩

tts-lemma *open-Compl*:
assumes $closed\ S$
shows $\tau\ (-\ S \cap U)$
is *topological-space-class.open-Compl*⟨*proof*⟩

tts-lemma *closed-Compl*:
assumes $S \subseteq U$ **and** $\tau\ S$
shows $closed\ (-\ S \cap U)$
is *topological-space-class.closed-Compl*⟨*proof*⟩

tts-lemma *open-Collect-neg*:
assumes $closed\ \{x \in U.\ P\ x\}$
shows $\tau\ \{x \in U.\ \neg\ P\ x\}$
given *topological-space-class.open-Collect-neg*
⟨*proof*⟩

tts-lemma *closed-Collect-neg*:
assumes $\tau\ \{x \in U.\ P\ x\}$
shows $closed\ \{x \in U.\ \neg\ P\ x\}$
given *topological-space-class.closed-Collect-neg*
⟨*proof*⟩

tts-lemma *closed-Collect-const*: $closed\ \{x \in U.\ P\}$
given *topological-space-class.closed-Collect-const*
⟨*proof*⟩

tts-lemma *connectedI*:
assumes
$\bigwedge A\ B.$
$[\![$
$A \subseteq U;$
$B \subseteq U;$
$\tau\ A;$
$\tau\ B;$
$A \cap U \neq \{\};$
$B \cap U \neq \{\};$
$A \cap (B \cap U) = \{\};$
$U \subseteq A \cup B$

$]] \Longrightarrow$ *False*
shows *connected U*
is *topological-space-class.connectedI⟨proof⟩*

tts-lemma *topological-basis*:
assumes $B \subseteq Pow\ U$
shows (*on U with τ* : «*topological-basis*» *B*) =
$(\forall\, x{\in}Pow\ U.\ \tau\ x = (\exists\, B'{\in}Pow\ (Pow\ U).\ B' \subseteq B \wedge \bigcup\ B' = x))$
is *topological-space-class.topological-basis⟨proof⟩*

tts-lemma *topological-basis-iff*:
assumes $B \subseteq Pow\ U$ **and** $\bigwedge B'.\ [\![B' \subseteq U;\ B' \in B]\!] \Longrightarrow \tau\ B'$
shows (*on U with τ* : «*topological-basis*» *B*) =
$(\forall\, O'{\in}Pow\ U.\ \tau\ O' \longrightarrow (\forall\, x{\in}O'.\ \exists\, B'{\in}B.\ B' \subseteq O' \wedge x \in B'))$
is *topological-space-class.topological-basis-iff⟨proof⟩*

tts-lemma *topological-basisI*:
assumes $B \subseteq Pow\ U$
and $\bigwedge B'.\ [\![B' \subseteq U;\ B' \in B]\!] \Longrightarrow \tau\ B'$
and $\bigwedge O'\ x.\ [\![O' \subseteq U;\ x \in U;\ \tau\ O';\ x \in O']\!] \Longrightarrow \exists\, y{\in}B.\ y \subseteq O' \wedge x \in y$
shows *on U with τ* : «*topological-basis*» *B*
is *topological-space-class.topological-basisI⟨proof⟩*

tts-lemma *closed-closure*:
assumes $S \subseteq U$
shows *closed* (*closure S*)
is *Elementary-Topology.closed-closure⟨proof⟩*

tts-lemma *closure-subset*: $S \subseteq$ *closure S*
is *Elementary-Topology.closure-subset⟨proof⟩*

tts-lemma *closure-eq*:
assumes $S \subseteq U$
shows (*closure S* = *S*) = *closed S*
is *Elementary-Topology.closure-eq⟨proof⟩*

tts-lemma *closure-closed*:
assumes $S \subseteq U$ **and** *closed S*
shows *closure S* = *S*
is *Elementary-Topology.closure-closed⟨proof⟩*

tts-lemma *closure-closure*:
assumes $S \subseteq U$
shows *closure* (*closure S*) = *closure S*
is *Elementary-Topology.closure-closure⟨proof⟩*

tts-lemma *closure-mono*:
assumes $T \subseteq U$ **and** $S \subseteq T$
shows *closure S* ⊆ *closure T*
is *Elementary-Topology.closure-mono⟨proof⟩*

tts-lemma *closure-minimal*:
assumes $T \subseteq U$ **and** $S \subseteq T$ **and** *closed T*
shows *closure S* ⊆ *T*
is *Elementary-Topology.closure-minimal⟨proof⟩*

tts-lemma *closure-unique*:
assumes $T \subseteq U$

and $S \subseteq T$
and *closed* T
and $\bigwedge T'.\ [\![T' \subseteq U;\ S \subseteq T';\ \textit{closed}\ T']\!] \Longrightarrow T \subseteq T'$
shows *closure* $S = T$
is *Elementary-Topology.closure-unique*⟨*proof*⟩

tts-lemma *closure-Un*:
assumes $S \subseteq U$ **and** $T \subseteq U$
shows *closure* $(S \cup T)$ = *closure* $S \cup$ *closure* T
is *Elementary-Topology.closure-Un*⟨*proof*⟩

tts-lemma *closure-eq-empty*: (*closure* $S = \{\}$) = $(S = \{\})$
is *Elementary-Topology.closure-eq-empty*⟨*proof*⟩

tts-lemma *closure-subset-eq*:
assumes $S \subseteq U$
shows (*closure* $S \subseteq S$) = *closed* S
is *Elementary-Topology.closure-subset-eq*⟨*proof*⟩

tts-lemma *open-Int-closure-eq-empty*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\tau\ S$
shows $(S \cap$ *closure* $T = \{\}) = (S \cap T = \{\})$
is *Elementary-Topology.open-Int-closure-eq-empty*⟨*proof*⟩

tts-lemma *open-Int-closure-subset*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** $\tau\ S$
shows $S \cap$ *closure* $T \subseteq$ *closure* $(S \cap T)$
is *Elementary-Topology.open-Int-closure-subset*⟨*proof*⟩

tts-lemma *closure-Un-frontier*: *closure* $S = S \cup$ *frontier* S
is *Elementary-Topology.closure-Un-frontier*⟨*proof*⟩

tts-lemma *compact-imp-countably-compact*:
assumes *compact* U
shows *countably-compact* U
is *Elementary-Topology.compact-imp-countably-compact*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating through *auto*
begin

tts-lemma *Heine-Borel-imp-Bolzano-Weierstrass*:
assumes $s \subseteq U$
and *compact* s
and *infinite* t
and $t \subseteq s$
shows $\exists x{\in}s.\ x$ «*islimpt*» t
is *Elementary-Topology.Heine-Borel-imp-Bolzano-Weierstrass*⟨*proof*⟩

end

tts-context

```
  tts: (?'a to U)
  rewriting ctr-simps
  substituting topological-space-ow-axioms
  eliminating ‹?U ≠ {}› through
    (
      unfold
        closed-ow-def
        compact-ow-def
        connected-ow-def
        interior-ow-def
        topological-basis-ow-def
        closure-ow-def
        frontier-ow-def
        countably-compact-ow-def,
      auto simp: connected-iff-const
    )
begin

tts-lemma connected-closed:
  assumes s ⊆ U
  shows connected s =
    (
      ¬(∃ A∈Pow U. ∃ B∈Pow U.
        closed A ∧
        closed B ∧
        s ⊆ A ∪ B ∧
        A ∩ (B ∩ s) = {} ∧
        A ∩ s ≠ {} ∧
        B ∩ s ≠ {})
    )
    is Topological-Spaces.connected-closed⟨proof⟩

tts-lemma closure-complement:
  assumes S ⊆ U
  shows closure (- S ∩ U) = - interior S ∩ U
    is Elementary-Topology.closure-complement⟨proof⟩

tts-lemma interior-complement:
  assumes S ⊆ U
  shows interior (- S ∩ U) = - closure S ∩ U
  is Elementary-Topology.interior-complement⟨proof⟩

tts-lemma interior-diff:
  assumes S ⊆ U and T ⊆ U
  shows interior (S - T) = interior S - closure T
  is Elementary-Topology.interior-diff⟨proof⟩

tts-lemma connected-imp-connected-closure:
  assumes S ⊆ U and connected S
  shows connected (closure S)
    is Elementary-Topology.connected-imp-connected-closure⟨proof⟩

tts-lemma frontier-closed:
  assumes S ⊆ U
  shows closed (frontier S)
    is Elementary-Topology.frontier-closed⟨proof⟩

tts-lemma frontier-Int:
```

assumes $S \subseteq U$ **and** $T \subseteq U$
shows *frontier* $(S \cap T)$ = *closure* $(S \cap T) \cap$ (*frontier* $S \cup$ *frontier* T)
is *Elementary-Topology.frontier-Int*⟨*proof*⟩

tts-lemma *frontier-closures*:
assumes $S \subseteq U$
shows *frontier* S = *closure* $S \cap$ *closure* $(-S \cap U)$
is *Elementary-Topology.frontier-closures*⟨*proof*⟩

tts-lemma *frontier-Int-subset*:
assumes $S \subseteq U$ **and** $T \subseteq U$
shows *frontier* $(S \cap T) \subseteq$ *frontier* $S \cup$ *frontier* T
is *Elementary-Topology.frontier-Int-subset*⟨*proof*⟩

tts-lemma *frontier-Int-closed*:
assumes $S \subseteq U$ **and** $T \subseteq U$ **and** *closed* S **and** *closed* T
shows *frontier* $(S \cap T)$ = *frontier* $S \cap T \cup S \cap$ *frontier* T
is *Elementary-Topology.frontier-Int-closed*⟨*proof*⟩

tts-lemma *frontier-subset-closed*:
assumes $S \subseteq U$ **and** *closed* S
shows *frontier* $S \subseteq S$
is *Elementary-Topology.frontier-subset-closed*⟨*proof*⟩

tts-lemma *frontier-subset-eq*:
assumes $S \subseteq U$
shows (*frontier* $S \subseteq S$) = *closed* S
is *Elementary-Topology.frontier-subset-eq*⟨*proof*⟩

tts-lemma *frontier-complement*:
assumes $S \subseteq U$
shows *frontier* $(-S \cap U)$ = *frontier* S
is *Elementary-Topology.frontier-complement*⟨*proof*⟩

tts-lemma *frontier-Un-subset*:
assumes $S \subseteq U$ **and** $T \subseteq U$
shows *frontier* $(S \cup T) \subseteq$ *frontier* $S \cup$ *frontier* T
is *Elementary-Topology.frontier-Un-subset*⟨*proof*⟩

tts-lemma *frontier-disjoint-eq*:
assumes $S \subseteq U$
shows (*frontier* $S \cap S = \{\}$) = τ S
is *Elementary-Topology.frontier-disjoint-eq*⟨*proof*⟩

tts-lemma *frontier-interiors*:
assumes $s \subseteq U$
shows *frontier* $s = -$ *interior* $s \cap U -$ *interior* $(-s \cap U)$
is *Elementary-Topology.frontier-interiors*⟨*proof*⟩

tts-lemma *frontier-interior-subset*:
assumes $S \subseteq U$
shows *frontier* (*interior* S) $\subseteq$ *frontier* S
is *Elementary-Topology.frontier-interior-subset*⟨*proof*⟩

tts-lemma *compact-Un*:
assumes $s \subseteq U$ **and** $t \subseteq U$ **and** *compact* s **and** *compact* t
shows *compact* $(s \cup t)$
is *Elementary-Topology.compact-Un*⟨*proof*⟩

tts-lemma *closed-Int-compact*:
assumes $s \subseteq U$ **and** $t \subseteq U$ **and** *closed* s **and** *compact* t
shows *compact* $(s \cap t)$
is *Elementary-Topology.closed-Int-compact⟨proof⟩*

tts-lemma *countably-compact-imp-compact*:
assumes $U \subseteq U$
and $B \subseteq Pow\ U$
and *countably-compact* U
and *countable* B
and *Ball* $B\ \tau$
and $\bigwedge T\ x.\ [\![T \subseteq U;\ x \in U;\ \tau\ T;\ x \in T;\ x \in U]\!] \Longrightarrow \exists\, y{\in}B.\ x \in y \wedge y \cap U \subseteq T$
shows *compact* U
is *Elementary-Topology.countably-compact-imp-compact⟨proof⟩*

end

tts-context
tts: $(?'a$ **to** $U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*insert closure-eq-empty*, *blast*)
begin

tts-lemma *closure-interior*:
assumes $S \subseteq U$
shows *closure* $S = -$ *interior* $(-\ S \cap U) \cap U$
is *Elementary-Topology.closure-interior⟨proof⟩*

end

tts-context
tts: $(?'a$ **to** $U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}›
through (*insert compact-empty*, *fastforce dest*: *subset-singletonD*)
begin

tts-lemma *compact-Union*:
assumes $S \subseteq Pow\ U$
and *finite* S
and $\bigwedge T.\ [\![T \subseteq U;\ T \in S]\!] \Longrightarrow$ *compact* T
shows *compact* $(\bigcup\ S)$
is *Elementary-Topology.compact-Union⟨proof⟩*

end

tts-context
tts: $(?'a$ **to** $U)$
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through**
(
insert
interior-empty
closure-ow-def

closed-UNIV
compact-empty
compact-ow-def,
auto
)
begin

tts-lemma *compactI*:
assumes $s \subseteq U$
and $\bigwedge C.\ [\![C \subseteq Pow\ U;\ Ball\ C\ \tau;\ s \subseteq \bigcup\ C]\!] \Longrightarrow$
$\exists\, x{\subseteq}Pow\ U.\ x \subseteq C \wedge finite\ x \wedge s \subseteq \bigcup\ x$
shows *compact s*
given *topological-space-class.compactI* ⟨*proof*⟩

tts-lemma *compactE*:
assumes $S \subseteq U$
and $\mathcal{T} \subseteq Pow\ U$
and *compact S*
and $S \subseteq \bigcup\ \mathcal{T}$
and $\bigwedge B.\ B \in \mathcal{T} \Longrightarrow \tau\ B$
and $\bigwedge \mathcal{T}'.\ [\![\mathcal{T}' \subseteq Pow\ U;\ \mathcal{T}' \subseteq \mathcal{T};\ finite\ \mathcal{T}';\ S \subseteq \bigcup\ \mathcal{T}']\!] \Longrightarrow thesis$
shows *thesis*
given *topological-space-class.compactE*
⟨*proof*⟩

tts-lemma *compact-fip*:
assumes $U \subseteq U$
shows *compact U* =
(
$\forall\, x{\subseteq}Pow\ U.$
Ball x closed ⟶
$(\forall\, y{\subseteq}Pow\ U.\ y \subseteq x \longrightarrow finite\ y \longrightarrow U \cap (\bigcap\ y \cap U) \neq \{\}) \longrightarrow$
$U \cap (\bigcap\ x \cap U) \neq \{\}$
)
given *topological-space-class.compact-fip* ⟨*proof*⟩

tts-lemma *compact-imp-fip*:
assumes $S \subseteq U$
and $Fa \subseteq Pow\ U$
and *compact S*
and $\bigwedge T.\ [\![T \subseteq U;\ T \in Fa]\!] \Longrightarrow closed\ T$
and $\bigwedge F'.\ [\![F' \subseteq Pow\ U;\ finite\ F';\ F' \subseteq Fa]\!] \Longrightarrow S \cap (\bigcap\ F' \cap U) \neq \{\}$
shows $S \cap (\bigcap\ Fa \cap U) \neq \{\}$
is *topological-space-class.compact-imp-fip*⟨*proof*⟩

tts-lemma *closed-limpt*:
assumes $S \subseteq U$
shows *closed S* = $(\forall\, x{\in}U.\ x\ «islimpt»\ S \longrightarrow x \in S)$
is *Elementary-Topology.closed-limpt*⟨*proof*⟩

tts-lemma *open-interior*:
assumes $S \subseteq U$
shows τ (*interior S*)
is *Elementary-Topology.open-interior*⟨*proof*⟩

tts-lemma *interior-subset*:
assumes $S \subseteq U$
shows *interior S* ⊆ *S*

is *Elementary-Topology.interior-subset⟨proof⟩*

tts-lemma *interior-open*:
assumes $S \subseteq U$ **and** $\tau\ S$
shows *interior* $S = S$
is *Elementary-Topology.interior-open⟨proof⟩*

tts-lemma *interior-eq*:
assumes $S \subseteq U$
shows $(\mathit{interior}\ S = S) = \tau\ S$
is *Elementary-Topology.interior-eq⟨proof⟩*

tts-lemma *interior-UNIV*: *interior* $U = U$
is *Elementary-Topology.interior-UNIV⟨proof⟩*

tts-lemma *interior-interior*:
assumes $S \subseteq U$
shows *interior* $(\mathit{interior}\ S) = \mathit{interior}\ S$
is *Elementary-Topology.interior-interior⟨proof⟩*

tts-lemma *interior-closure*:
assumes $S \subseteq U$
shows *interior* $S = -\ \mathit{closure}\ (-\ S \cap U) \cap U$
is *Elementary-Topology.interior-closure⟨proof⟩*

tts-lemma *finite-imp-compact*:
assumes $s \subseteq U$ **and** *finite* s
shows *compact* s
is *Elementary-Topology.finite-imp-compact⟨proof⟩*

tts-lemma *countably-compactE*:
assumes $s \subseteq U$
and $C \subseteq \mathit{Pow}\ U$
and *countably-compact* s
and $\mathit{Ball}\ C\ \tau$
and $s \subseteq \bigcup\ C$
and *countable* C
and $\bigwedge C'.\ [\![C' \subseteq \mathit{Pow}\ U;\ C' \subseteq C;\ \mathit{finite}\ C';\ s \subseteq \bigcup\ C']\!] \Longrightarrow \mathit{thesis}$
shows *thesis*
is *Elementary-Topology.countably-compactE⟨proof⟩*

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **and** ‹*?A* ⊆ *U*› **through** (*insert interior-empty, auto*)
begin

tts-lemma *interior-unique*:
assumes $S \subseteq U$
and $T \subseteq S$
and $\tau\ T$
and $\bigwedge T'.\ [\![T' \subseteq S;\ \tau\ T']\!] \Longrightarrow T' \subseteq T$
shows *interior* $S = T$
given *Elementary-Topology.interior-unique*
⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹*U*$_2$::*'b set*›)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*simp add*: *subset-iff filterlim-iff*)
begin

tts-lemma *open-UN*:
assumes *A* ⊆ *U*$_2$
and ∀ *x*∈*U*$_2$. *B x* ⊆ *U*
and ∀ *x*∈*A*. τ (*B x*)
shows τ (⋃ (*B* ‘ *A*))
is *topological-space-class.open-UN*⟨*proof*⟩

tts-lemma *open-Collect-ex*:
assumes ⋀*i*. *i* ∈ *U*$_2$ ⟹ τ {*x*. *P i x* ∧ *x* ∈ *U*}
shows τ {*x* ∈ *U*. ∃ *i*∈*U*$_2$. *P i x*}
is *open-Collect-ex*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹*U*$_2$::*'b set*›)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*unfold closed-ow-def finite-def*, *auto*)
begin

tts-lemma *open-INT*:
assumes *A* ⊆ *U*$_2$ **and** ∀ *x*∈*U*$_2$. *B x* ⊆ *U* **and** *finite A* **and** ∀ *x*∈*A*. τ (*B x*)
shows τ (⋂ (*B* ‘ *A*) ∩ *U*)
is *topological-space-class.open-INT*⟨*proof*⟩

tts-lemma *closed-INT*:
assumes *A* ⊆ *U*$_2$ **and** ∀ *x*∈*U*$_2$. *B x* ⊆ *U* **and** ∀ *x*∈*A*. *closed* (*B x*)
shows *closed* (⋂ (*B* ‘ *A*) ∩ *U*)
is *topological-space-class.closed-INT*⟨*proof*⟩

tts-lemma *closed-UN*:
assumes *A* ⊆ *U*$_2$
and ∀ *x*∈*U*$_2$. *B x* ⊆ *U*
and *finite A*
and ∀ *x*∈*A*. *closed* (*B x*)
shows *closed* (⋃ (*B* ‘ *A*))
is *topological-space-class.closed-UN*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*) **and** (*?'b* **to** ‹*U*$_2$::*'b set*›)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*insert closed-empty*, *auto*)
begin

tts-lemma *closed-Collect-all*:
assumes $\bigwedge i.\ i \in U_2 \Longrightarrow$ *local.closed* $\{x.\ P\ i\ x \wedge x \in U\}$
shows *local.closed* $\{x \in U.\ \forall\, i{\in}U_2.\ P\ i\ x\}$
is *topological-space-class.closed-Collect-all*⟨*proof*⟩

tts-lemma *compactE-image*:
assumes $S \subseteq U$
and $C \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \subseteq U$
and *compact* S
and $\bigwedge T.\ [\![T \in U_2;\ T \in C]\!] \Longrightarrow \tau\ (f\ T)$
and $S \subseteq \bigcup\ (f\ `\ C)$
and $\bigwedge C'.\ [\![C' \subseteq U_2;\ C' \subseteq C;$ *finite* $C';\ S \subseteq \bigcup\ (f\ `\ C')]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *topological-space-class.compactE-image*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U) **and** (*?'b* **to** ‹U_2::*'b set*›)
rewriting *ctr-simps*
substituting *topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*simp*, *blast* | *simp*)
begin

tts-lemma *ne-compact-imp-fip-image*:
assumes $s \subseteq U$
and $I \subseteq U_2$
and $\forall\, x{\in}U_2.\ f\ x \subseteq U$
and *compact* s
and $\bigwedge i.\ [\![i \in U_2;\ i \in I]\!] \Longrightarrow$ *closed* $(f\ i)$
and $\bigwedge I'.\ [\![I' \subseteq U_2;$ *finite* $I';\ I' \subseteq I]\!] \Longrightarrow s \cap (\bigcap\ (f\ `\ I') \cap U) \neq \{\}$
shows $s \cap (\bigcap\ (f\ `\ I) \cap U) \neq \{\}$
is *topological-space-class.compact-imp-fip-image*⟨*proof*⟩

end

end

3.16.4 Further results

lemma *topological-basis-closed*:
assumes *topological-basis-ow* $U\ \tau\ B$
shows $B \subseteq$ *Pow* U
⟨*proof*⟩

lemma *ts-open-eq-ts-open*:
assumes *topological-space-ow* $U\ \tau'$ **and** $\bigwedge s.\ s \subseteq U \Longrightarrow \tau'\ s = \tau\ s$
shows *topological-space-ow* $U\ \tau$
⟨*proof*⟩

lemma (**in** *topological-space-ow*) *topological-basis-closed*:
assumes *topological-basis-ow* $U\ \tau\ B$
shows $B \subseteq$ *Pow* U
⟨*proof*⟩

3.17 Relativization of the results related to the countability properties of topological spaces

3.17.1 First countable topological space

Definitions and common properties

locale *first-countable-topology-ow* =
  topological-space-ow U τ **for** U :: $'at$ *set* **and** τ +
  assumes *first-countable-basis*:
    (
      $\forall\, x{\in}U.$
      (
        $\exists\, B{::}nat \Rightarrow {'at}\ set.$
          $(\forall\, i.\ B\ i \subseteq U \wedge x \in B\ i \wedge \tau\ (B\ i)) \wedge$
          $(\forall\, S.\ S \subseteq U \wedge \tau\ S \wedge x \in S \longrightarrow (\exists\, i.\ B\ i \subseteq S))$
)
)

locale *ts-fct-ow* =
  ts: *topological-space-ow* U_1 τ_1 + *fct*: *first-countable-topology-ow* U_2 τ_2
  for U_1 :: $'at$ *set* **and** τ_1 **and** U_2 :: $'bt$ *set* **and** τ_2
begin

sublocale *topological-space-pair-ow* U_1 τ_1 U_2 τ_2 ⟨*proof*⟩

end

locale *first-countable-topology-pair-ow* =
  fct_1: *first-countable-topology-ow* U_1 τ_1 +
  fct_2: *first-countable-topology-ow* U_2 τ_2
  for U_1 :: $'at$ *set* **and** τ_1 **and** U_2 :: $'bt$ *set* **and** τ_2
begin

sublocale *ts-fct-ow* U_1 τ_1 U_2 τ_2 ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

private lemma *first-countable-topology-transfer-h*:
  $(\forall\, i.\ B\ i \subseteq$ *Collect* $($*Domainp* $A) \wedge x \in B\ i \wedge \tau\ (B\ i)) =$
    $(B$ ‘ *Collect top* $\subseteq \{Aa.\ Aa \subseteq$ *Collect* $($*Domainp* $A)\} \wedge$
    $(\forall\, i.\ x \in B\ i \wedge \tau\ (B\ i)))$
  ⟨*proof*⟩

lemma *first-countable-topology-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*rel-set A* ===> (=)) ===> (=))
      (*first-countable-topology-ow* (*Collect* (*Domainp A*)))
      class.first-countable-topology
  ⟨*proof*⟩

end

Relativization

context *first-countable-topology-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *first-countable-topology-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** *simp*
begin

tts-lemma *countable-basis-at-decseq*:
assumes $x \in U$
and $\bigwedge A.$
$[\![$
range $A \subseteq$ *Pow* U;
$\bigwedge i.\ \tau\ (A\ i)$;
$\bigwedge i.\ x \in A\ i$;
$\bigwedge S.\ [\![S \subseteq U;\ \tau\ S;\ x \in S]\!] \Longrightarrow \forall_F\ i$ *in sequentially*. $A\ i \subseteq S$
$]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *first-countable-topology-class.countable-basis-at-decseq*⟨*proof*⟩

tts-lemma *first-countable-basisE*:
assumes $x \in U$
and $\bigwedge \mathcal{A}.$
$[\![$
$\mathcal{A} \subseteq$ *Pow* U;
countable $\mathcal{A}$;
$\bigwedge A.\ [\![A \subseteq U;\ A \in \mathcal{A}]\!] \Longrightarrow x \in A$;
$\bigwedge A.\ [\![A \subseteq U;\ A \in \mathcal{A}]\!] \Longrightarrow \tau\ A$;
$\bigwedge S.\ [\![S \subseteq U;\ \tau\ S;\ x \in S]\!] \Longrightarrow \exists\, A{\in}\mathcal{A}.\ A \subseteq S]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *first-countable-topology-class.first-countable-basisE*⟨*proof*⟩

tts-lemma *first-countable-basis-Int-stableE*:
assumes $x \in U$
and $\bigwedge \mathcal{A}.$
$[\![$
$\mathcal{A} \subseteq$ *Pow* U;
countable $\mathcal{A}$;
$\bigwedge A.\ [\![A \subseteq U;\ A \in \mathcal{A}]\!] \Longrightarrow x \in A$;
$\bigwedge A.\ [\![A \subseteq U;\ A \in \mathcal{A}]\!] \Longrightarrow \tau\ A$;
$\bigwedge S.\ [\![S \subseteq U;\ \tau\ S;\ x \in S]\!] \Longrightarrow \exists\, A{\in}\mathcal{A}.\ A \subseteq S$;
$\bigwedge A\ B.\ [\![A \subseteq U;\ B \subseteq U;\ A \in \mathcal{A};\ B \in \mathcal{A}]\!] \Longrightarrow A \cap B \in \mathcal{A}$
$]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *first-countable-topology-class.first-countable-basis-Int-stableE*⟨*proof*⟩

end

end

3.17.2 Topological space with a countable basis

Definitions and common properties

locale *countable-basis-ow* =
  topological-space-ow U τ **for** *U* :: *'at set* **and** *τ* +
  fixes *B* :: *'at set set*
  assumes *is-basis*: *topological-basis-ow U τ B*
    and *countable-basis*: *countable B*
begin

lemma *B-ss-PowU*[*simp*]: *B* ⊆ *Pow U*
  ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *countable-basis-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    ((*rel-set A* ===> (=)) ===> *rel-set* (*rel-set A*) ===> (=))
      (*countable-basis-ow* (*Collect* (*Domainp A*))) *countable-basis*
⟨*proof*⟩

end

Relativization

context *countable-basis-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  rewriting *ctr-simps*
  substituting *countable-basis-ow-axioms*
  eliminating ‹*?U* ≠ {}› **through** *auto*
  applying [*OF B-ss-PowU*]
begin

tts-lemma *open-countable-basis-ex*:
  assumes *X* ⊆ *U* **and** *τ X*
  shows ∃ *B'*∈*Pow* (*Pow U*). *B'* ⊆ *B* ∧ *X* = ⋃ *B'*
    is *countable-basis.open-countable-basis-ex*⟨*proof*⟩

tts-lemma *countable-dense-exists*:
  ∃ *D*∈*Pow U*.
    countable D ∧
    (∀ *X*∈*Pow U*. *τ X* ⟶ *X* ≠ {} ⟶ (∃ *d*∈*D*. *d* ∈ *X*))
    is *countable-basis.countable-dense-exists*⟨*proof*⟩

tts-lemma *open-countable-basisE*:
  assumes *X* ⊆ *U*
    and *τ X*
    and ⋀*B'*. ⟦*B'* ⊆ *Pow U*; *B'* ⊆ *B*; *X* = ⋃ *B'*⟧ ⟹ *thesis*

shows *thesis*
is *countable-basis.open-countable-basisE⟨proof⟩*

tts-lemma *countable-dense-setE*:
assumes ⋀*D*.
[[*D* ⊆ *U*; *countable D*; ⋀*X*. [[*X* ⊆ *U*; *τ X*; *X* ≠ {}]] ⟹ ∃ *x*∈*D*. *x* ∈ *X*]] ⟹ *thesis*
shows *thesis*
is *countable-basis.countable-dense-setE⟨proof⟩*

end

end

3.17.3 Second countable topological space

Definitions and common properties

locale *second-countable-topology-ow* =
topological-space-ow U τ **for** *U* :: *'at set* **and** *τ* +
assumes *second-countable-basis*:
∃ *B*⊆*Pow U*. *countable B* ∧ (∀ *S*⊆*U*. *τ S* = *generate-topology-on B U S*)

Transfer rules

context
includes *lifting-syntax*
begin

lemma *second-countable-topology-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
((*rel-set A* ===> (=)) ===> (=))
(*second-countable-topology-ow* (*Collect* (*Domainp A*)))
class.second-countable-topology
⟨*proof*⟩

end

Relativization

context *second-countable-topology-ow*
begin

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *second-countable-topology-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*unfold topological-basis-ow-def*, *auto*)
begin

tts-lemma *ex-countable-basis*:
∃ *B*∈*Pow* (*Pow U*). *countable B* ∧ (*on U with τ* : «*topological-basis*» *B*)
is *Elementary-Topology.ex-countable-basis⟨proof⟩*

end

tts-context
tts: (*?'a* **to** *U*)
rewriting *ctr-simps*

substituting *second-countable-topology-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*auto simp*: *countable-subset*)
begin

tts-lemma *countable-dense-exists*:
∃ *D*∈*Pow U*. *countable D* ∧ (∀ *X*∈*Pow U*. *τ X* ⟶ *X* ≠ {} ⟶ (∃ *d*∈*D*. *d* ∈ *X*))
is *Elementary-Topology.countable-dense-exists*⟨*proof*⟩

tts-lemma *countable-dense-setE*:
assumes ⋀*D*.
⟦
D ⊆ *U*;
countable D;
⋀*X*. ⟦*X* ⊆ *U*; *τ X*; *X* ≠ {}⟧ ⟹ ∃ *d*∈*D*. *d* ∈ *X*
⟧ ⟹ *thesis*
shows *thesis*
is *Elementary-Topology.countable-dense-setE*⟨*proof*⟩

tts-lemma *univ-second-countable*:
assumes ⋀*ℬ*.
⟦
ℬ ⊆ *Pow U*;
countable ℬ;
⋀*C*. ⟦*C* ⊆ *U*; *C* ∈ *ℬ*⟧ ⟹ *τ C*;
⋀*S*. ⟦*S* ⊆ *U*; *τ S*⟧ ⟹ ∃ *U*∈*Pow* (*Pow U*). *U* ⊆ *ℬ* ∧ *S* = ⋃ *U*
⟧ ⟹ *thesis*
shows *thesis*
is *Elementary-Topology.univ-second-countable*⟨*proof*⟩

tts-lemma *Lindelof*:
assumes *ℱ* ⊆ *Pow U*
and ⋀*S*. ⟦*S* ⊆ *U*; *S* ∈ *ℱ*⟧ ⟹ *τ S*
and ⋀*ℱ′*. ⟦*ℱ′* ⊆ *Pow U*; *ℱ′* ⊆ *ℱ*; *countable ℱ′*; ⋃ *ℱ′* = ⋃ *ℱ*⟧ ⟹ *thesis*
shows *thesis*
is *Elementary-Topology.Lindelof*⟨*proof*⟩

tts-lemma *countable-disjoint-open-subsets*:
assumes *ℱ* ⊆ *Pow U* **and** ⋀*S*. ⟦*S* ⊆ *U*; *S* ∈ *ℱ*⟧ ⟹ *τ S* **and** *disjoint ℱ*
shows *countable ℱ*
is *Elementary-Topology.countable-disjoint-open-subsets*⟨*proof*⟩

end

end

3.18 Relativization of the results about ordered topological spaces

3.18.1 Ordered topological space

Definitions and common properties

locale *order-topology-ow* =
  order-ow U le ls **for** *U* :: *'at set* **and** *le ls* +
  fixes τ :: *'at set* ⇒ *bool*
  assumes *open-generated-order*: $s \subseteq U \Longrightarrow$
    $\tau\ s =$
      (
        (
          in-topology-generated-by
            (
              $(\lambda a.\ (on\ U\ with\ (<_{ow}) : \{..< a\}))$ ‘ $U \cup$
              $(\lambda a.\ (on\ U\ with\ (<_{ow}) : \{a <..\}))$ ‘ U
)
          on U : «*open*» *s*
)
)
begin

sublocale *topological-space-ow*
⟨*proof*⟩

end

locale *ts-ot-ow* =
  ts: *topological-space-ow* $U_1\ \tau_1$ + *ot*: *order-topology-ow* $U_2\ le_2\ ls_2\ \tau_2$
  for U_1 :: *'at set* **and** τ_1 **and** U_2 :: *'bt set* **and** $le_2\ ls_2\ \tau_2$
begin

sublocale *topological-space-pair-ow* $U_1\ \tau_1\ U_2\ \tau_2$ ⟨*proof*⟩

end

locale *order-topology-pair-ow* =
  ot_1: *order-topology-ow* $U_1\ le_1\ ls_1\ \tau_1$ + ot_2: *order-topology-ow* $U_2\ le_2\ ls_2\ \tau_2$
  for U_1 :: *'at set* **and** $le_1\ ls_1\ \tau_1$ **and** U_2 :: *'bt set* **and** $le_2\ ls_2\ \tau_2$
begin

sublocale *ts-ot-ow* $U_1\ \tau_1\ U_2\ le_2\ ls_2\ \tau_2$ ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *order-topology-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (
      (*A* ===> *A* ===> (=)) ===>

```
    (A ===> A ===> (=)) ===>
    (rel-set A ===> (=)) ===>
    (=)
  ) (order-topology-ow (Collect (Domainp A))) class.order-topology
 ⟨proof⟩

end
```

Relativization

```
context order-topology-ow
begin

tts-context
  tts: (?'a to U)
  substituting order-topology-ow-axioms
  eliminating ‹?U ≠ {}› through simp
begin

tts-lemma open-greaterThan:
  assumes a ∈ U
  shows τ {a<_ow..}
    is order-topology-class.open-greaterThan⟨proof⟩

tts-lemma open-lessThan:
  assumes a ∈ U
  shows τ {..<_ow a}
  is order-topology-class.open-lessThan⟨proof⟩

tts-lemma open-greaterThanLessThan:
  assumes a ∈ U and b ∈ U
  shows τ {a<_ow..<_ow b}
    is order-topology-class.open-greaterThanLessThan⟨proof⟩

end

end
```

3.18.2 Linearly ordered topological space

Definitions and common properties

```
locale linorder-topology-ow =
  linorder-ow U le ls + order-topology-ow U le ls τ
  for U :: 'at set and le ls τ

locale ts-lt-ow =
  ts: topological-space-ow U_1 τ_1 + lt: linorder-topology-ow U_2 le_2 ls_2 τ_2
  for U_1 :: 'at set and τ_1 and U_2 :: 'bt set and le_2 ls_2 τ_2
begin

sublocale ts-ot-ow U_1 τ_1 U_2 le_2 ls_2 τ_2 ⟨proof⟩

end

locale ot-lt-ow =
  ot: order-topology-ow U_1 le_1 ls_1 τ_1 + lt: linorder-topology-ow U_2 le_2 ls_2 τ_2
  for U_1 :: 'at set and le_1 ls_1 τ_1 and U_2 :: 'bt set and le_2 ls_2 τ_2
begin
```

sublocale *ts-lt-ow* U_1 τ_1 U_2 le_2 ls_2 τ_2 ⟨*proof*⟩
sublocale *order-topology-pair-ow* U_1 le_1 ls_1 τ_1 U_2 le_2 ls_2 τ_2 ⟨*proof*⟩

end

locale *linorder-topology-pair-ow* =
  lt_1: *linorder-topology-ow* U_1 le_1 ls_1 τ_1 + lt_2: *linorder-topology-ow* U_2 le_2 ls_2 τ_2
  for U_1 :: *'at set* **and** le_1 ls_1 τ_1 **and** U_2 :: *'bt set* **and** le_2 ls_2 τ_2
begin

sublocale *ot-lt-ow* U_1 le_1 ls_1 τ_1 U_2 le_2 ls_2 τ_2 ⟨*proof*⟩

end

Transfer rules

context
  includes *lifting-syntax*
begin

lemma *linorder-topology-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (
      (*A* ===> *A* ===> (=)) ===>
      (*A* ===> *A* ===> (=)) ===>
      (*rel-set A* ===> (=)) ===>
      (=)
)
    (*linorder-topology-ow* (*Collect* (*Domainp A*))) *class.linorder-topology*
  ⟨*proof*⟩

end

Relativization

context *linorder-topology-ow*
begin

tts-context
  tts: (*?'a* **to** *U*)
  rewriting *ctr-simps*
  substituting *linorder-topology-ow-axioms*
  eliminating ‹*?U* ≠ {}› **through** *clarsimp*
begin

tts-lemma *open-right*:
  assumes $S \subseteq U$
    and $x \in U$
    and $y \in U$
    and $\tau\ S$
    and $x \in S$
    and $x <_{ow} y$
  shows $\exists z{\in}U.\ x <_{ow} z \wedge (on\ U\ with\ (\leq_{ow})\ (<_{ow}) : \{x..<z\}) \subseteq S$
    is *linorder-topology-class.open-right*⟨*proof*⟩

tts-lemma *open-left*:

assumes $S \subseteq U$
and $x \in U$
and $y \in U$
and $\tau\ S$
and $x \in S$
and $y <_{ow} x$
shows $\exists\, z{\in}U.\ z <_{ow} x \wedge \{z{<_{ow}}..x\} \subseteq S$
is *linorder-topology-class.open-left*⟨*proof*⟩

tts-lemma *connectedD-interval*:
assumes $U \subseteq U$
and $x \in U$
and $y \in U$
and $z \in U$
and *connected* U
and $x \in U$
and $y \in U$
and $x \leq_{ow} z$
and $z \leq_{ow} y$
shows $z \in U$
is *linorder-topology-class.connectedD-interval*⟨*proof*⟩

tts-lemma *connected-contains-Icc*:
assumes $A \subseteq U$
and $a \in U$
and $b \in U$
and *connected* A
and $a \in A$
and $b \in A$
shows $\{a{..}_{ow}b\} \subseteq A$
is *Topological-Spaces.connected-contains-Icc*⟨*proof*⟩

tts-lemma *connected-contains-Ioo*:
assumes $A \subseteq U$
and $a \in U$
and $b \in U$
and *connected* A
and $a \in A$
and $b \in A$
shows $\{a{<_{ow}}..{<_{ow}}b\} \subseteq A$
is *Topological-Spaces.connected-contains-Ioo*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *linorder-topology-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** *clarsimp*
begin

tts-lemma *not-in-connected-cases*:
assumes $S \subseteq U$
and $x \in U$
and *connected* S
and $x \notin S$
and $S \neq \{\}$
and $[\![\mathit{bdd\text{-}above}\ S;\ \bigwedge y.\ [\![y \in U;\ y \in S]\!] \Longrightarrow y \leq_{ow} x]\!] \Longrightarrow \mathit{thesis}$

and $[\![bdd\text{-}below\ S;\ \bigwedge y.\ [\![y \in U;\ y \in S]\!] \Longrightarrow x \leq_{ow} y]\!] \Longrightarrow thesis$
shows *thesis*
is *linorder-topology-class.not-in-connected-cases*⟨*proof*⟩

tts-lemma *compact-attains-sup*:
assumes $S \subseteq U$
and *compact S*
and $S \neq \{\}$
shows $\exists x{\in}S.\ \forall y{\in}S.\ y \leq_{ow} x$
is *linorder-topology-class.compact-attains-sup*⟨*proof*⟩

tts-lemma *compact-attains-inf*:
assumes $S \subseteq U$
and *compact S*
and $S \neq \{\}$
shows $\exists x{\in}S.\ Ball\ S\ ((\leq_{ow})\ x)$
is *linorder-topology-class.compact-attains-inf*⟨*proof*⟩

end

end

3.19 Relativization of the results about product topologies

3.19.1 Definitions and common properties

ud‹
open-prod-inst.open-prod::
(*'a*::*topological-space* × *'b*::*topological-space*) *set* ⇒ -
›
ud‹*open*::(*'a*::*topological-space* × *'b*::*topological-space*) *set* ⇒ *bool*›

ctr relativization
synthesis *ctr-simps*
assumes [*transfer-domain-rule*, *transfer-rule*]:
Domainp B = (λ*x*. *x* ∈ *U*$_1$) *Domainp A* = (λ*x*. *x* ∈ *U*$_2$)
and [*transfer-rule*]: *bi-unique A right-total A*
bi-unique B right-total B
trp (*?'a A*) **and** (*?'b B*)
in *open-ow*: *open-prod.with-def*
(‹*'*(/*on - - with - - : «open» -/'*)› [1000, 999, 1000, 999, 1000] 10)

locale *product-topology-ow* =
ts$_1$: *topological-space-ow U*$_1$ τ$_1$ + *ts*$_2$: *topological-space-ow U*$_2$ τ$_2$
for *U*$_1$ :: *'at set* **and** τ$_1$ :: *'at set* ⇒ *bool*
and *U*$_2$ :: *'bt set***and** τ$_2$ :: *'bt set* ⇒ *bool* +
fixes τ :: (*'at* × *'bt*) *set* ⇒ *bool*
assumes *open-prod*[*tts-implicit*]: τ = *open-ow U*$_1$ *U*$_2$ τ$_2$ τ$_1$
begin

sublocale *topological-space-ow* ‹*U*$_1$ × *U*$_2$› τ
⟨*proof*⟩

end

3.19.2 Transfer rules

lemma (**in** *product-topology-ow*) *open-with-oo-transfer*[*transfer-rule*]:
includes *lifting-syntax*
fixes *A* :: [*'at*, *'a*] ⇒ *bool*
and *B* :: [*'bt*, *'b*] ⇒ *bool*
assumes *tdr-U*$_1$[*transfer-domain-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*$_1$)
and [*transfer-rule*]: *bi-unique A right-total A*
and *tdr-U*$_2$[*transfer-domain-rule*]: *Domainp B* = (λ*x*. *x* ∈ *U*$_2$)
and [*transfer-rule*]: *bi-unique B right-total B*
and τ$_1$τ$_1$*'*[*transfer-rule*]: (*rel-set A* ===> (=)) τ$_1$ τ$_1$*'*
and τ$_2$τ$_2$*'*[*transfer-rule*]: (*rel-set B* ===> (=)) τ$_2$ τ$_2$*'*
shows (*rel-set* (*rel-prod A B*) ===> (=)) τ (*open-prod.with* τ$_2$*'* τ$_1$*'*)
⟨*proof*⟩

3.19.3 Relativization

context *product-topology-ow*
begin

tts-context
tts: (*?'a* **to** *U*$_1$) **and** (*?'b* **to** *U*$_2$)
rewriting *ctr-simps*
substituting *ts*$_1$.*topological-space-ow-axioms*
and *ts*$_2$.*topological-space-ow-axioms*
eliminating ‹*?U* ≠ {}› **through** (*fold tts-implicit*, *insert closed-empty*, *simp*)

applying [*folded tts-implicit*]
begin

tts-lemma *open-prod-intro*:
assumes $S \subseteq U_1 \times U_2$
and $\bigwedge x.\ [\![x \in U_1 \times U_2;\ x \in S]\!] \Longrightarrow$
$\exists A{\in}Pow\ U_1.\ \exists B{\in}Pow\ U_2.\ \tau_1\ A \wedge \tau_2\ B \wedge A \times B \subseteq S \wedge x \in A \times B$
shows $\tau\ S$
is *open-prod-intro*⟨*proof*⟩

tts-lemma *open-Times*:
assumes $S \subseteq U_1$ **and** $T \subseteq U_2$ **and** $\tau_1\ S$ **and** $\tau_2\ T$
shows $\tau\ (S \times T)$
is *open-Times*⟨*proof*⟩

tts-lemma *open-vimage-fst*:
assumes $S \subseteq U_1$ **and** $\tau_1\ S$
shows $\tau\ (fst\ -`\ S \cap U_1 \times U_2)$
is *open-vimage-fst*⟨*proof*⟩

tts-lemma *closed-vimage-fst*:
assumes $S \subseteq U_1$ **and** $ts_1.closed\ S$
shows $closed\ (fst\ -`\ S \cap U_1 \times U_2)$
is *closed-vimage-fst*⟨*proof*⟩

tts-lemma *closed-Times*:
assumes $S \subseteq U_1$ **and** $T \subseteq U_2$ **and** $ts_1.closed\ S$ **and** $ts_2.closed\ T$
shows $closed\ (S \times T)$
is *closed-Times*⟨*proof*⟩

tts-lemma *open-image-fst*:
assumes $S \subseteq U_1 \times U_2$ **and** $\tau\ S$
shows $\tau_1\ (fst\ `\ S)$
is *open-image-fst*⟨*proof*⟩

tts-lemma *open-image-snd*:
assumes $S \subseteq U_1 \times U_2$ **and** $\tau\ S$
shows $\tau_2\ (snd\ `\ S)$
is *open-image-snd*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U_1) **and** (*?'b* **to** U_2)
rewriting *ctr-simps*
substituting $ts_1.topological\text{-}space\text{-}ow\text{-}axioms$
and $ts_2.topological\text{-}space\text{-}ow\text{-}axioms$
eliminating ‹*?U* ≠ {}›
through (*fold tts-implicit*, *unfold connected-ow-def*, *simp*)
applying [*folded tts-implicit*]
begin

tts-lemma *connected-Times*:
assumes $S \subseteq U_1$ **and** $T \subseteq U_2$ **and** $ts_1.connected\ S$ **and** $ts_2.connected\ T$
shows $connected\ (S \times T)$
is *connected-Times*⟨*proof*⟩

tts-lemma *connected-Times-eq*:

assumes $S \subseteq U_1$ **and** $T \subseteq U_2$
shows
$connected\ (S \times T) = (S = \{\} \vee T = \{\} \vee ts_1.connected\ S \wedge ts_2.connected\ T)$
is *connected-Times-eq*⟨*proof*⟩

end

tts-context
tts: (?'b **to** U_1) **and** (?'a **to** U_2)
rewriting *ctr-simps*
substituting $ts_1.topological\text{-}space\text{-}ow\text{-}axioms$
and $ts_2.topological\text{-}space\text{-}ow\text{-}axioms$
eliminating ‹?$U \neq \{\}$› **through** (*fold tts-implicit*, *insert closed-empty*, *simp*)
applying [*folded tts-implicit*]
begin

tts-lemma *open-vimage-snd*:
assumes $S \subseteq U_2$ **and** $\tau_2\ S$
shows $\tau\ (snd -` S \cap U_1 \times U_2)$
is *open-vimage-snd*⟨*proof*⟩

tts-lemma *closed-vimage-snd*:
assumes $S \subseteq U_2$ **and** $ts_2.closed\ S$
shows $closed\ (snd -` S \cap U_1 \times U_2)$
is *closed-vimage-snd*⟨*proof*⟩

end

end

TTS Vector Spaces

4.1 Introduction

4.1.1 Background

The content of this chapter is an adoption of the applied relativization study presented in [14] to the ETTS. The content of this chapter incorporates many elements of the content of the aforementioned relativization study without an explicit reference. Nonetheless, no attempt was made to ensure that the theorems obtained as a result of this work are identical to the theorems obtained in [14].

4.1.2 Prerequisites

ctr parametricity
  in *bij-betw-ow*: *bij-betw-def*

lemma *bij-betw-parametric'*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes *bi-unique A*
  shows ((*A* ===> *A*) ===> *rel-set A* ===> *rel-set A* ===> (=))
    bij-betw bij-betw
  ⟨*proof*⟩

lemma *vimage-transfer*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: *bi-unique B right-total A*
  shows
    ((*A* ===> *B*) ===> (*rel-set B*) ===> *rel-set A*)
      (λ*f s*. (*vimage f s*) ∩ (*Collect* (*Domainp A*))) (−‘)
  ⟨*proof*⟩

lemma *Eps-unique-transfer-lemma*:
  includes *lifting-syntax*
  assumes [*transfer-rule*]:
    right-total A (*A* ===> (=)) *f g* (*A* ===> (=)) *f' g'*
    and *holds*: ∃ *x*. *Domainp A x* ∧ *f x*
    and *unique-g*: ⋀*x y*. ⟦ *g x*; *g y* ⟧ ⟹ *g' x* = *g' y*
  shows *f'* (*Eps* (λ*x*. *Domainp A x* ∧ *f x*)) = *g'* (*Eps g*)
⟨*proof*⟩

4.2 Groups

4.2.1 Definitions and elementary properties

locale *semigroup-add-ow* =
  fixes S :: $'a$ *set* **and** *pls* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹$\oplus_{ow}$› 65)
  assumes *add-assoc*:
    $[\![\ a \in S;\ b \in S;\ c \in S\ ]\!] \Longrightarrow (a \oplus_{ow} b) \oplus_{ow} c = a \oplus_{ow} (b \oplus_{ow} c)$
    and *add-closed*: $[\![\ a \in S;\ b \in S\ ]\!] \Longrightarrow a \oplus_{ow} b \in S$
begin

lemma *add-closed'*[*simp*]: $\forall x{\in}S.\ \forall y{\in}S.\ x \oplus_{ow} y \in S$ ⟨*proof*⟩

end

locale *ab-semigroup-add-ow* = *semigroup-add-ow* +
  assumes *add-commute*: $[\![\ a \in S;\ b \in S\ ]\!] \Longrightarrow a \oplus_{ow} b = b \oplus_{ow} a$

locale *comm-monoid-add-ow* = *ab-semigroup-add-ow* +
  fixes z
  assumes *add-zero*: $a \in S \Longrightarrow z \oplus_{ow} a = a$
    and *zero-closed*[*simp*]: $z \in S$
begin

lemma *carrier-ne*[*simp*]: $S \neq \{\}$ ⟨*proof*⟩

end

definition *sum-with pls z f S* =
  (
    if $\exists C.\ f\ `\ S \subseteq C \wedge$ *comm-monoid-add-ow C pls z*
    then Finite-Set.fold (*pls o f*) *z S*
    else z
)

lemma *sum-with-empty*[*simp*]: *sum-with pls z f* $\{\}$ = *z*
  ⟨*proof*⟩

lemma *sum-with-cases*[*case-names comm zero*]:
  assumes $\bigwedge C.\ [\![\ f\ `\ S \subseteq C;$ *comm-monoid-add-ow C pls z* $]\!] \Longrightarrow$
      P (*Finite-Set.fold* (*pls o f*) *z S*)
    and $(\bigwedge C.$ *comm-monoid-add-ow C pls z* $\Longrightarrow (\exists s{\in}S.\ f\ s \notin C)) \Longrightarrow P\ z$
  shows *P* (*sum-with pls z f S*)
  ⟨*proof*⟩

context *comm-monoid-add-ow*
begin

lemma *sum-with-infinite*: *infinite A* $\Longrightarrow$ *sum-with* ($\oplus_{ow}$) *z g A* = *z*
  ⟨*proof*⟩

context
begin

abbreviation *pls'* :: $'a \Rightarrow 'a \Rightarrow 'a$
  where *pls'* $\equiv \lambda x\ y.$ (*if x* $\in$ *S then x else z*) $\oplus_{ow}$ (*if y* $\in$ *S then y else z*)

lemma *fold-pls'-closed*: *Finite-Set.fold* (*pls'* $\circ$ *g*) *z A* $\in$ *S* **if** *g* ‘ *A* $\subseteq$ *S*

⟨*proof*⟩

lemma *fold-pls'-eq*:
assumes g ‘ $A \subseteq S$
shows *Finite-Set.fold* $(pls' \circ g)$ z A = *Finite-Set.fold* $(pls \circ g)$ z A
⟨*proof*⟩

lemma *sum-with-closed*:
assumes g ‘ $A \subseteq S$
shows *sum-with pls z g A* $\in S$
⟨*proof*⟩

lemma *sum-with-insert*:
assumes *g-into*: $g\ x \in S$ g ‘ $A \subseteq S$
and *A*: *finite A*
and *x*: $x \notin A$
shows *sum-with pls z g* (*insert x A*) = $(g\ x) \oplus_{ow}$ (*sum-with pls z g A*)
⟨*proof*⟩

end

end

locale *ab-group-add-ow* = *comm-monoid-add-ow* +
fixes *mns um*
assumes *ab-left-minus*: $a \in S \Longrightarrow (um\ a) \oplus_{ow} a = z$
and *ab-diff-conv-add-uminus*:
$[\![\ a \in S;\ b \in S\ ]\!] \Longrightarrow mns\ a\ b = a \oplus_{ow} (um\ b)$
and *uminus-closed*: $a \in S \Longrightarrow um\ a \in S$

4.2.2 Instances (by type class constraints)

lemma *semigroup-add-ow-Ball-def*:
semigroup-add-ow S pls ⟷
$(\forall a{\in}S.\ \forall b{\in}S.\ \forall c{\in}S.\ pls\ (pls\ a\ b)\ c =$
$pls\ a\ (pls\ b\ c)) \wedge (\forall a{\in}S.\ \forall b{\in}S.\ pls\ a\ b \in S)$
⟨*proof*⟩

lemma *ab-semigroup-add-ow-Ball-def*:
ab-semigroup-add-ow S pls ⟷
semigroup-add-ow S pls $\wedge$ $(\forall a{\in}S.\ \forall b{\in}S.\ pls\ a\ b = pls\ b\ a)$
⟨*proof*⟩

lemma *comm-monoid-add-ow-Ball-def*:
comm-monoid-add-ow S pls z ⟷
ab-semigroup-add-ow S pls $\wedge$ $(\forall a{\in}S.\ pls\ z\ a = a) \wedge z \in S$
⟨*proof*⟩

lemma *comm-monoid-add-ow*[*simp*]:
comm-monoid-add-ow UNIV (+) (0::'*a*::*comm-monoid-add*)
⟨*proof*⟩

lemma *ab-group-add-ow-Ball-def*:
ab-group-add-ow S pls z mns um ⟷
comm-monoid-add-ow S pls z $\wedge$
$(\forall a{\in}S.\ pls\ (um\ a)\ a = z) \wedge$
$(\forall a{\in}S.\ \forall b{\in}S.\ mns\ a\ b = pls\ a\ (um\ b)) \wedge$
$(\forall a{\in}S.\ um\ a \in S)$

⟨*proof*⟩

lemma *sum-with*[*ud-with*]: *sum* = *sum-with* (+) 0
⟨*proof*⟩

lemmas [*tts-implicit*] = *sum-with*[*symmetric*]

4.2.3 Transfer rules

context
includes *lifting-syntax*
begin

lemma *semigroup-add-on-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique A*
shows (*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
semigroup-add-ow semigroup-add-ow
⟨*proof*⟩

lemma *Domainp-applyI*:
includes *lifting-syntax*
shows (*A* ===> *B*) *f g* ⟹ *A x y* ⟹ *Domainp B* (*f x*)
⟨*proof*⟩

lemma *Domainp-apply2I*:
includes *lifting-syntax*
shows (*A* ===> *B* ===> *C*) *f g* ⟹ *A x y* ⟹ *B x′ y′* ⟹ *Domainp C* (*f x x′*)
⟨*proof*⟩

lemma *ab-semigroup-add-on-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
ab-semigroup-add-ow ab-semigroup-add-ow
⟨*proof*⟩

lemma *right-total-semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows ((*A* ===> *A* ===> *A*) ===> (=))
(*semigroup-add-ow* (*Collect* (*Domainp A*))) *class.semigroup-add*
⟨*proof*⟩

lemma *comm-monoid-add-on-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=))
comm-monoid-add-ow comm-monoid-add-ow
⟨*proof*⟩

lemma *right-total-ab-semigroup-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows
((*A* ===> *A* ===> *A*) ===> (=))
(*ab-semigroup-add-ow* (*Collect* (*Domainp A*))) *class.ab-semigroup-add*
⟨*proof*⟩

lemma *right-total-comm-monoid-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows ((*A* ===> *A* ===> *A*) ===> *A* ===> (=))
(*comm-monoid-add-ow* (*Collect* (*Domainp A*))) *class.comm-monoid-add*
⟨*proof*⟩

lemma *ab-group-add-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows
((*A* ===> *A* ===> *A*) ===> *A* ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A*)===> (=))
(*ab-group-add-ow* (*Collect* (*Domainp A*))) *class.ab-group-add*
⟨*proof*⟩

lemma *ex-comm-monoid-add-around-imageE*:
assumes *ex-comm*: ∃ *C*. *f* ‘ *S* ⊆ *C* ∧ *comm-monoid-add-ow C pls zero*
and *transfers*:
(*A* ===> *A* ===> *A*) *pls pls*′
A zero zero′
Domainp (*rel-set B*) *S*
and *in-dom*: ⋀*x*. *x* ∈ *S* ⟹ *Domainp A* (*f x*)
obtains *C* **where**
comm-monoid-add-ow C pls zero f ‘ *S* ⊆ *C Domainp* (*rel-set A*) *C*
⟨*proof*⟩

lemma *Domainp-sum-with*:
includes *lifting-syntax*
assumes ⋀*x*. *x* ∈ *t* ⟹ *Domainp A* (*r x*) *t* ⊆ *Collect* (*Domainp A*)
and *transfer-rules*[*transfer-rule*]: (*A* ===> *A* ===> *A*) *p p*′ *A z z*′
shows *DsI*: *Domainp A* (*sum-with p z r t*)
⟨*proof*⟩

lemma *sum-with-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique A bi-unique B*
shows ((*A* ===> *A* ===> *A*) ===> *A* ===> (*B* ===> *A*) ===> *rel-set B* ===> *A*)
sum-with sum-with
⟨*proof*⟩

end

4.2.4 Relativization.

context *ab-group-add-ow*
begin

tts-context
tts: (*?*′*a* **to** *S*)
rewriting *ctr-simps*
substituting *comm-monoid-add-ow-axioms*
eliminating ‹*S* ≠ {}› **through** *auto*
applying [*OF add-closed*′ *zero-closed*]
begin

tts-lemma *mono-neutral-cong-left*:
assumes *range h* ⊆ *S*
and *range g* ⊆ *S*
and *finite T*
and *Sa* ⊆ *T*

and $\forall x \in T - Sa.\ h\ x = z$
and $\bigwedge x.\ x \in Sa \Longrightarrow g\ x = h\ x$
shows *sum-with* $(\oplus_{ow})$ z g Sa = *sum-with* $(\oplus_{ow})$ z h T
is *sum.mono-neutral-cong-left*⟨*proof*⟩

end

end

4.3 Modules

4.3.1 *module-with*

locale *module-with* = *ab-group-add* $plus_M$ $zero_M$ $minus_M$ $uminus_M$
  for $plus_M$ :: ['m, 'm] ⇒ 'm (**infixl** ‹+$_M$› 65)
    and $zero_M$ (‹0$_M$›)
    and $minus_M$ (**infixl** ‹−$_M$› 65)
    and $uminus_M$ (‹−$_M$ -› [81] 80) +
  fixes *scale* :: ['cr1::*comm-ring*-1, 'm] ⇒ 'm (**infixr** ‹∗s_{with}› 75)
  assumes *scale-right-distrib*[*algebra-simps*]:
    $a *s_{with} (x +_M y) = a *s_{with} x +_M a *s_{with} y$
    and *scale-left-distrib*[*algebra-simps*]:
      $(a + b) *s_{with} x = a *s_{with} x +_M b *s_{with} x$
    and *scale-scale*[*simp*]: $a *s_{with} (b *s_{with} x) = (a * b) *s_{with} x$
    and *scale-one*[*simp*]: $1 *s_{with} x = x$

lemma *module-with-overloaded*[*ud-with*]: *module* = *module-with* (+) 0 (−) *uminus*
  ⟨*proof*⟩

locale *module-pair-with* =
  M_1: *module-with* $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$ +
  M_2: *module-with* $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
  for $plus_{M\text{-}1}$ :: ['m-1, 'm-1] ⇒ 'm-1 (**infixl** ‹+$_{M'\text{-}1}$› 65)
    and $zero_{M\text{-}1}$ (‹0$_{M'\text{-}1}$›)
    and $minus_{M\text{-}1}$ (**infixl** ‹−$_{M'\text{-}1}$› 65)
    and $uminus_{M\text{-}1}$ (‹−$_{M'\text{-}1}$ -› [81] 80)
    and $scale_1$ (**infixr** ‹∗$s_{with'\text{-}1}$› 75)
    and $plus_{M\text{-}2}$ :: ['m-2, 'm-2] ⇒ 'm-2 (**infixl** ‹+$_{M'\text{-}2}$› 65)
    and $zero_{M\text{-}2}$ (‹0$_{M'\text{-}2}$›)
    and $minus_{M\text{-}2}$ (**infixl** ‹−$_{M'\text{-}2}$› 65)
    and $uminus_{M\text{-}2}$ (‹−$_{M'\text{-}2}$ -› [81] 80)
    and $scale_2$ (**infixr** ‹∗$s_{with'\text{-}2}$› 75)

lemma *module-pair-with-overloaded*[*ud-with*]:
  module-pair =
    (
      λ$scale_1$ $scale_2$.
        module-pair-with (+) 0 (−) *uminus* $scale_1$ (+) 0 (−) *uminus* $scale_2$
)
  ⟨*proof*⟩

locale *module-hom-with* =
  M_1: *module-with* $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$ +
  M_2: *module-with* $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
  for $plus_{M\text{-}1}$ :: ['m-1, 'm-1] ⇒ 'm-1 (**infixl** ‹+$_{M'\text{-}1}$› 65)
    and $zero_{M\text{-}1}$ (‹0$_{M'\text{-}1}$›)
    and $minus_{M\text{-}1}$ (**infixl** ‹−$_{M'\text{-}1}$› 65)
    and $uminus_{M\text{-}1}$ (‹−$_{M'\text{-}1}$ -› [81] 80)
    and $scale_1$ (**infixr** ‹∗$s_{with'\text{-}1}$› 75)
    and $plus_{M\text{-}2}$ :: ['m-2, 'm-2] ⇒ 'm-2 (**infixl** ‹+$_{M'\text{-}2}$› 65)
    and $zero_{M\text{-}2}$ (‹0$_{M'\text{-}2}$›)
    and $minus_{M\text{-}2}$ (**infixl** ‹−$_{M'\text{-}2}$› 65)
    and $uminus_{M\text{-}2}$ (‹−$_{M'\text{-}2}$ -› [81] 80)
    and $scale_2$ (**infixr** ‹∗$s_{with'\text{-}2}$› 75) +
  fixes f :: 'm-1 ⇒ 'm-2
  assumes *add*: $f\ (b1 +_{M\text{-}1} b2) = f\ b1 +_{M\text{-}2} f\ b2$
    and *scale*: $f\ (r *s_{with\text{-}1} b) = r *s_{with\text{-}2} f\ b$

begin

sublocale *module-pair-with* ⟨*proof*⟩

end

lemma *module-hom-with-overloaded*[*ud-with*]:
module-hom =
(
λ*scale*$_1$ *scale*$_2$.
module-hom-with (+) 0 (−) *uminus scale*$_1$ (+) 0 (−) *uminus scale*$_2$
)
⟨*proof*⟩
ud ‹*module.subspace*› (‹(*with* - - - : «*subspace*» -)› [1000, 999, 998, 1000] 10)
ud ‹*module.span*› (‹(*with* - - - : «*span*» -)› [1000, 999, 998, 1000] 10)
ud ‹*module.dependent*›
(‹(*with* - - - - : «*dependent*» -)› [1000, 999, 998, 997, 1000] 10)
ud ‹*module.representation*›
(
‹(*with* - - - - : «*representation*» - -)›
[1000, 999, 998, 997, 1000, 999] 10
)

abbreviation *independent-with*
(‹(*with* - - - - : «*independent*» -)› [1000, 999, 998, 997, 1000] 10)
where
(*with zero*$_{CR1}$ *zero*$_M$ *scale*$_M$ *plus*$_M$: «*independent*» *s*) ≡
¬(*with zero*$_{CR1}$ *zero*$_M$ *scale*$_M$ *plus*$_M$: «*dependent*» *s*)

lemma *span-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows
(
A ===>
(*A* ===> *A* ===> *A*) ===>
((=) ===> *A* ===> *A*) ===>
rel-set A ===>
rel-set A
) *span.with span.with*
⟨*proof*⟩

lemma *dependent-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *right-total A bi-unique A*
shows
(
(=) ===>
A ===>
(*A* ===> *A* ===> *A*) ===>
((=) ===> *A* ===> *A*) ===>
rel-set A ===>
(=)
) *dependent.with dependent.with*
⟨*proof*⟩

ctr relativization
synthesis *ctr-simps*

assumes [*transfer-rule*]: *is-equality A bi-unique B*
trp (*?'a A*) **and** (*?'b B*)
in *subspace-with*: *subspace.with-def*

4.3.2 *module-ow*

Definitions and common properties

Single module.

locale *module-ow* = *ab-group-add-ow* U_M $plus_M$ $zero_M$ $minus_M$ $uminus_M$
for U_M :: *'m set*
and $plus_M$ (**infixl** ‹$+_M$› 65)
and $zero_M$ (‹0_M›)
and $minus_M$ (**infixl** ‹$-_M$› 65)
and $uminus_M$ (‹$-_M$ -› [81] 80) +
fixes *scale* :: ['*cr*1::*comm-ring*-1, *'m*] ⇒ *'m* (**infixr** ‹$*s_M$› 75)
assumes *scale-closed*[*simp*, *intro*]: $x \in U_M \Longrightarrow a *s_M x \in U_M$
and *scale-right-distrib*[*algebra-simps*]:
$[\![x \in U_M; y \in U_M]\!] \Longrightarrow a *s_M (x +_M y) = a *s_M x +_M a *s_M y$
and *scale-left-distrib*[*algebra-simps*]:
$x \in U_M \Longrightarrow (a + b) *s_M x = a *s_M x +_M b *s_M x$
and *scale-scale*[*simp*]:
$x \in U_M \Longrightarrow a *s_M (b *s_M x) = (a * b) *s_M x$
and *scale-one*[*simp*]: $x \in U_M \Longrightarrow 1 *s_M x = x$
begin

lemma *scale-closed'*[*simp*]: $\forall a.\ \forall x{\in}U_M.\ a *s_M x \in U_M$ ⟨*proof*⟩

lemma *minus-closed'*[*simp*]: $\forall x{\in}U_M.\ \forall y{\in}U_M.\ x -_M y \in U_M$
⟨*proof*⟩

lemma *uminus-closed'*[*simp*]: $\forall x{\in}U_M.\ -_M x \in U_M$ ⟨*proof*⟩

tts-register-sbts ‹$(*s_M)$› | U_M
⟨*proof*⟩

tts-register-sbts $plus_M$ | U_M
⟨*proof*⟩

tts-register-sbts $zero_M$ | U_M
⟨*proof*⟩

end

Pair of modules.

locale *module-pair-ow* =
M_1: *module-ow* $U_{M\text{-}1}$ $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$ +
M_2: *module-ow* $U_{M\text{-}2}$ $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$ :: *'m-1 set*
and $plus_{M\text{-}1}$ (**infixl** ‹$+_{M'\text{-}1}$› 65)
and $zero_{M\text{-}1}$ (‹$0_{M'\text{-}1}$›)
and $minus_{M\text{-}1}$ (**infixl** ‹$-_{M'\text{-}1}$› 65)
and $uminus_{M\text{-}1}$ (‹$-_{M'\text{-}1}$ -› [81] 80)
and $scale_1$ :: ['*cr*1::*comm-ring*-1, *'m*-1] ⇒ *'m*-1 (**infixr** ‹$*s_{M'\text{-}1}$› 75)
and $U_{M\text{-}2}$ :: *'m*-2 *set*
and $plus_{M\text{-}2}$ (**infixl** ‹$+_{M'\text{-}2}$› 65)
and $zero_{M\text{-}2}$ (‹$0_{M'\text{-}2}$›)
and $minus_{M\text{-}2}$ (**infixl** ‹$-_{M'\text{-}2}$› 65)

and $uminus_{M\text{-}2}$ ($\langle -_{M'\text{-}2}$ -$\rangle$ [81] 80)
and $scale_2$ :: [$'cr1$::*comm-ring*-1, $'m$-2] ⇒ $'m$-2 (**infixr** $\langle *s_{M'\text{-}2}\rangle$ 75)

Module homomorphisms.

locale *module-hom-ow* =
M_1: *module-ow* $U_{M\text{-}1}$ $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$ +
M_2: *module-ow* $U_{M\text{-}2}$ $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$ :: $'m$-1 *set*
and $plus_{M\text{-}1}$ (**infixl** $\langle +_{M'\text{-}1}\rangle$ 65)
and $zero_{M\text{-}1}$ ($\langle 0_{M'\text{-}1}\rangle$)
and $minus_{M\text{-}1}$ (**infixl** $\langle -_{M'\text{-}1}\rangle$ 65)
and $uminus_{M\text{-}1}$ ($\langle -_{M'\text{-}1}$ -$\rangle$ [81] 80)
and $scale_1$ :: [$'cr1$::*comm-ring*-1, $'m$-1] ⇒ $'m$-1 (**infixr** $\langle *s_{M'\text{-}1}\rangle$ 75)
and $U_{M\text{-}2}$ :: $'m$-2 *set*
and $plus_{M\text{-}2}$ (**infixl** $\langle +_{M'\text{-}2}\rangle$ 65)
and $zero_{M\text{-}2}$ ($\langle 0_{M'\text{-}2}\rangle$)
and $minus_{M\text{-}2}$ (**infixl** $\langle -_{M'\text{-}2}\rangle$ 65)
and $uminus_{M\text{-}2}$ ($\langle -_{M'\text{-}2}$ -$\rangle$ [81] 80)
and $scale_2$ :: [$'cr1$::*comm-ring*-1, $'m$-2] ⇒ $'m$-2 (**infixr** $\langle *s_{M'\text{-}2}\rangle$ 75) +
fixes f :: $'m$-1 ⇒ $'m$-2
assumes *f-closed*[*simp*]: f ‘ $U_{M\text{-}1} \subseteq U_{M\text{-}2}$
and *add*: ⟦ $b1 \in U_{M\text{-}1}$; $b2 \in U_{M\text{-}1}$ ⟧ ⟹ $f\ (b1 +_{M\text{-}1} b2) = f\ b1 +_{M\text{-}2} f\ b2$
and *scale*: ⟦ $r \in U_{CR1}$; $b \in U_{M\text{-}1}$ ⟧ ⟹ $f\ (r *s_{M\text{-}1} b) = r *s_{M\text{-}2} f\ b$
begin

tts-register-sbts f | $\langle U_{M\text{-}1}\rangle$ **and** $\langle U_{M\text{-}2}\rangle$ ⟨*proof*⟩

lemma *f-closed'*[*simp*]: ∀ $x \in U_{M\text{-}1}$. $f\ x \in U_{M\text{-}2}$ ⟨*proof*⟩

sublocale *module-pair-ow* ⟨*proof*⟩

end

Transfer.

lemma *module-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique B right-total B*
fixes *PP lhs*
defines
PP ≡
(
(*B* ===> *B* ===> *B*) ===>
B ===>
(*B* ===> *B* ===> *B*) ===>
(*B* ===> *B*) ===>
((=) ===> *B* ===> *B*) ===>
(=)
)
and
lhs ≡
(
λ $plus_M$ $zero_M$ $minus_M$ $uminus_M$ *scale*.
module-ow (*Collect* (*Domainp B*)) $plus_M$ $zero_M$ $minus_M$ $uminus_M$ *scale*
)
shows *PP lhs* (*module-with*)
⟨*proof*⟩

lemma *module-pair-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B_1 *right-total* B_1 *bi-unique* B_2 *right-total* B_2
fixes *PP lhs*
defines
$PP \equiv$
(
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$B_1 \Longrightarrow$
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_1 \Longrightarrow B_1) \Longrightarrow$
$((=) \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$B_2 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(B_2 \Longrightarrow B_2) \Longrightarrow$
$((=) \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(=)$
)
and
$lhs \equiv$
(
λ
$plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$
$plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$.
module-pair-ow
(*Collect* (*Domainp* B_1)) $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$
(*Collect* (*Domainp* B_2)) $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
)
shows *PP lhs module-pair-with*
⟨*proof*⟩

lemma *module-hom-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B_1 *right-total* B_1 *bi-unique* B_2 *right-total* B_2
fixes *PP lhs*
defines
$PP \equiv$
(
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$B_1 \Longrightarrow$
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_1 \Longrightarrow B_1) \Longrightarrow$
$((=) \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$B_2 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(B_2 \Longrightarrow B_2) \Longrightarrow$
$((=) \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(B_1 \Longrightarrow B_2) \Longrightarrow$
$(=)$
)
and
$lhs \equiv$
(
λ

$plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$
$plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
f.
module-hom-ow
(*Collect* (*Domainp* B_1)) $plus_{M\text{-}1}$ $zero_{M\text{-}1}$ $minus_{M\text{-}1}$ $uminus_{M\text{-}1}$ $scale_1$
(*Collect* (*Domainp* B_2)) $plus_{M\text{-}2}$ $zero_{M\text{-}2}$ $minus_{M\text{-}2}$ $uminus_{M\text{-}2}$ $scale_2$
f
)
shows *PP lhs module-hom-with*
⟨*proof*⟩

4.3.3 *module-on*

Definitions and common properties

locale *module-on* =
fixes U_M
and *scale* :: $'a$::*comm-ring*-1 ⇒ $'b$::*ab-group-add* ⇒ $'b$ (**infixr** ‹$*s$› 75)
assumes *scale-right-distrib-on*[*algebra-simps*]:
$[\![x \in U_M; y \in U_M]\!] \Longrightarrow a *s (x + y) = a *s x + a *s y$
and *scale-left-distrib-on*[*algebra-simps*]:
$x \in U_M \Longrightarrow (a + b) *s x = a *s x + b *s x$
and *scale-scale-on*[*simp*]: $x \in U_M \Longrightarrow a *s (b *s x) = (a * b) *s x$
and *scale-one-on*[*simp*]: $x \in U_M \Longrightarrow 1 *s x = x$
and *closed-add*: $[\![x \in U_M; y \in U_M]\!] \Longrightarrow x + y \in U_M$
and *closed-zero*: $0 \in U_M$
and *closed-scale*: $x \in U_M \Longrightarrow a *s x \in U_M$
begin

lemma *S-ne*: $U_M \neq \{\}$ ⟨*proof*⟩

lemma *scale-minus-left-on*:
assumes $x \in U_M$
shows *scale* $(-a)$ $x = -$ *scale* a x
⟨*proof*⟩

lemma *closed-uminus*:
assumes $x \in U_M$
shows $-x \in U_M$
⟨*proof*⟩

sublocale $implicit_M$: *ab-group-add-ow* U_M ‹(+)› 0 ‹(−)› *uminus*
⟨*proof*⟩

sublocale $implicit_M$: *module-ow* U_M ‹(+)› 0 ‹(−)› *uminus* ‹(*s)›
⟨*proof*⟩

definition *subspace* :: $'b$ *set* ⇒ *bool*
where *subspace-on-def*: *subspace* $T \longleftrightarrow$
$0 \in T \wedge (\forall x{\in}T.\ \forall y{\in}T.\ x + y \in T) \wedge (\forall c.\ \forall x{\in}T.\ c *s x \in T)$

definition *span* :: $'b$ *set* ⇒ $'b$ *set*
where *span-on-def*: *span* $b = \{sum\ (\lambda a.\ r\ a *s\ a)\ t \mid t\ r.\ finite\ t \wedge t \subseteq b\}$

definition *dependent* :: $'b$ *set* ⇒ *bool*
where *dependent-on-def*: *dependent* $s \longleftrightarrow$
$(\exists t\ u.\ finite\ t \wedge t \subseteq s \wedge (sum\ (\lambda v.\ u\ v *s v)\ t = 0 \wedge (\exists v{\in}t.\ u\ v \neq 0)))$

lemma *implicit-subspace-with*[*tts-implicit*]: *subspace.with* (+) 0 (∗*s*) = *subspace*
⟨*proof*⟩

lemma *implicit-dependent-with*[*tts-implicit*]:
dependent.with 0 0 (+) (∗*s*) = *dependent*
⟨*proof*⟩

lemma *implicit-span-with*[*tts-implicit*]: *span.with* 0 (+) (∗*s*) = *span*
⟨*proof*⟩

end

lemma *implicit-module-ow*[*tts-implicit*]:
module-ow U_M (+) 0 (−) *uminus* = *module-on* U_M
⟨*proof*⟩

locale *module-pair-on* =
M_1: *module-on* $U_{M\text{-}1}$ $scale_1$ + M_2: *module-on* $U_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$:: 'b::*ab-group-add set*
and $U_{M\text{-}2}$::'c::*ab-group-add set*
and $scale_1$::'a::*comm-ring*-1 ⇒ - ⇒ - (**infixr** ‹∗s_1› 75)
and $scale_2$::'a::*comm-ring*-1 ⇒ - ⇒ - (**infixr** ‹∗s_2› 75)
begin

sublocale $implicit_M$: *module-pair-ow*
$U_{M\text{-}1}$ ‹(+)› 0 ‹(−)› *uminus* $scale_1$ $U_{M\text{-}2}$ ‹(+)› 0 ‹(−)› *uminus* $scale_2$
⟨*proof*⟩

end

lemma *implicit-module-pair-ow*[*tts-implicit*]:
module-pair-ow $U_{M\text{-}1}$ (+) 0 (−) *uminus* $scale_1$ $U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ =
module-pair-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$
⟨*proof*⟩

locale *module-hom-on* = M_1: *module-on* $U_{M\text{-}1}$ $scale_1$ + M_2: *module-on* $U_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$ :: 'b::*ab-group-add set* **and** $U_{M\text{-}2}$ :: 'c::*ab-group-add set*
and $scale_1$ :: 'a::*comm-ring*-1 ⇒ 'b ⇒ 'b (**infixr** ‹∗s_1› 75)
and $scale_2$ :: 'a::*comm-ring*-1 ⇒ 'c ⇒ 'c (**infixr** ‹∗s_2› 75) +
fixes f :: 'b ⇒ 'c
assumes *hom-closed*: f ‘ $U_{M\text{-}1}$ ⊆ $U_{M\text{-}2}$
and *add*: ⋀b1 b2. ⟦ b1 ∈ $U_{M\text{-}1}$; b2 ∈ $U_{M\text{-}1}$ ⟧ ⟹ f (b1 + b2) = f b1 + f b2
and *scale*: ⋀b. b ∈ $U_{M\text{-}1}$ ⟹ f (r ∗s_1 b) = r ∗s_2 f b
begin

sublocale *module-pair-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ ⟨*proof*⟩

sublocale $implicit_M$: *module-hom-ow*
$U_{M\text{-}1}$ ‹(+)› 0 ‹(−)› *uminus* $scale_1$ $U_{M\text{-}2}$ ‹(+)› 0 ‹(−)› *uminus* $scale_2$
⟨*proof*⟩

end

lemma *implicit-module-hom-ow*[*tts-implicit*]:
module-hom-ow $U_{M\text{-}1}$ (+) 0 (−) *uminus* $scale_1$ $U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ =
module-hom-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$
⟨*proof*⟩

4.3.4 Relativization.

context *module-on*
begin

tts-context
tts: (*?'b* **to** ‹U_M::'*b set*›)
rewriting *ctr-simps*
substituting *implicit*$_M$.*module-ow-axioms*
and *implicit*$_M$.*ab-group-add-ow-axioms*
eliminating ‹*?a* ∈ U_M› **and** ‹*?B* ⊆ U_M› **through** *auto*
applying
[
OF
implicit$_M$.*carrier-ne*
implicit$_M$.*add-closed'*
implicit$_M$.*minus-closed'*
implicit$_M$.*uminus-closed'*
implicit$_M$.*scale-closed'*,
unfolded tts-implicit
]
begin

tts-lemma *scale-left-commute*:
assumes $x \in U_M$
shows $a *s b *s x = b *s a *s x$
is *module.scale-left-commute*⟨*proof*⟩

tts-lemma *scale-zero-left*:
assumes $x \in U_M$
shows $0 *s x = 0$
is *module.scale-zero-left*⟨*proof*⟩

tts-lemma *scale-minus-left*:
assumes $x \in U_M$
shows $- a *s x = - (a *s x)$
is *module.scale-minus-left*⟨*proof*⟩

tts-lemma *scale-left-diff-distrib*:
assumes $x \in U_M$
shows $(a - b) *s x = a *s x - b *s x$
is *module.scale-left-diff-distrib*⟨*proof*⟩

tts-lemma *scale-sum-left*:
assumes $x \in U_M$
shows *sum f A* $*s x = (\sum a{\in}A.\ f\ a *s x)$
is *module.scale-sum-left*⟨*proof*⟩

tts-lemma *scale-zero-right*: $a *s 0 = 0$
is *module.scale-zero-right*⟨*proof*⟩

tts-lemma *scale-minus-right*:
assumes $x \in U_M$
shows $a *s - x = - (a *s x)$
is *module.scale-minus-right*⟨*proof*⟩

tts-lemma *scale-right-diff-distrib*:
assumes $x \in U_M$ **and** $y \in U_M$

shows $a *s (x - y) = a *s x - a *s y$
is *module.scale-right-diff-distrib*⟨*proof*⟩

tts-lemma *scale-sum-right*:
assumes *range f* ⊆ U_M
shows $a *s$ *sum f A* = $(\sum x{\in}A.\ a *s f x)$
is *module.scale-sum-right*⟨*proof*⟩

tts-lemma *sum-constant-scale*:
assumes $y \in U_M$
shows $(\sum x{\in}A.\ y)$ = *of-nat* (*card A*) $*s$ y
is *module.sum-constant-scale*⟨*proof*⟩

tts-lemma *subspace-def*:
assumes $S \subseteq U_M$
shows *subspace S* =
$(0 \in S \wedge (\forall x.\ \forall y{\in}S.\ x *s y \in S) \wedge (\forall x{\in}S.\ \forall y{\in}S.\ x + y \in S))$
is *module.subspace-def*⟨*proof*⟩

tts-lemma *subspaceI*:
assumes $S \subseteq U_M$
and $0 \in S$
and $\bigwedge x\ y.\ [\![x \in U_M;\ y \in U_M;\ x \in S;\ y \in S]\!] \Longrightarrow x + y \in S$
and $\bigwedge c\ x.\ [\![x \in U_M;\ x \in S]\!] \Longrightarrow c *s x \in S$
shows *subspace S*
is *module.subspaceI*⟨*proof*⟩

tts-lemma *subspace-single-0*: *subspace* {0}
is *module.subspace-single-0*⟨*proof*⟩

tts-lemma *subspace-0*:
assumes $S \subseteq U_M$ **and** *subspace S*
shows $0 \in S$
is *module.subspace-0*⟨*proof*⟩

tts-lemma *subspace-add*:
assumes $S \subseteq U_M$ **and** *subspace S* **and** $x \in S$ **and** $y \in S$
shows $x + y \in S$
is *module.subspace-add*⟨*proof*⟩

tts-lemma *subspace-scale*:
assumes $S \subseteq U_M$ **and** *subspace S* **and** $x \in S$
shows $c *s x \in S$
is *module.subspace-scale*⟨*proof*⟩

tts-lemma *subspace-neg*:
assumes $S \subseteq U_M$ **and** *subspace S* **and** $x \in S$
shows $- x \in S$
is *module.subspace-neg*⟨*proof*⟩

tts-lemma *subspace-diff*:
assumes $S \subseteq U_M$ **and** *subspace S* **and** $x \in S$ **and** $y \in S$
shows $x - y \in S$
is *module.subspace-diff*⟨*proof*⟩

tts-lemma *subspace-sum*:
assumes $A \subseteq U_M$
and *range f* ⊆ U_M

and *subspace A*
and $\bigwedge x.\ x \in B \Longrightarrow f\ x \in A$
shows *sum f B* $\in A$
is *module.subspace-sum*⟨*proof*⟩

tts-lemma *subspace-inter*:
assumes $A \subseteq U_M$ **and** $B \subseteq U_M$ **and** *subspace A* **and** *subspace B*
shows *subspace* $(A \cap B)$
is *module.subspace-inter*⟨*proof*⟩

tts-lemma *span-explicit'*:
assumes $b \subseteq U_M$
shows *span b* =
{
$x \in U_M.\ \exists f.$
$x = (\sum v{\in}\{x \in U_M.\ f\ x \neq 0\}.\ f\ v *s\ v) \wedge$
finite $\{x \in U_M.\ f\ x \neq 0\} \wedge$
$(\forall\, x{\in}U_M.\ f\ x \neq 0 \longrightarrow x \in b)$
}
is *module.span-explicit'*⟨*proof*⟩

tts-lemma *span-finite*:
assumes $S \subseteq U_M$ **and** *finite S*
shows *span S* = *range* $(\lambda u.\ \sum v{\in}S.\ u\ v *s\ v)$
is *module.span-finite*⟨*proof*⟩

tts-lemma *span-induct-alt*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $x \in$ *span S*
and *h* 0
and $\bigwedge c\ x\ y.\ [\![x \in U_M;\ y \in U_M;\ x \in S;\ h\ y]\!] \Longrightarrow h\ (c *s\ x + y)$
shows *h x*
is *module.span-induct-alt*⟨*proof*⟩

tts-lemma *span-mono*:
assumes $B \subseteq U_M$ **and** $A \subseteq B$
shows *span A* $\subseteq$ *span B*
is *module.span-mono*⟨*proof*⟩

tts-lemma *span-base*:
assumes $S \subseteq U_M$ **and** $a \in S$
shows $a \in$ *span S*
is *module.span-base*⟨*proof*⟩

tts-lemma *span-superset*:
assumes $S \subseteq U_M$
shows $S \subseteq$ *span S*
is *module.span-superset*⟨*proof*⟩

tts-lemma *span-zero*:
assumes $S \subseteq U_M$
shows 0 $\in$ *span S*
is *module.span-zero*⟨*proof*⟩

tts-lemma *span-add*:
assumes $x \in U_M$
and $S \subseteq U_M$

and $y \in U_M$
and $x \in span\ S$
and $y \in span\ S$
shows $x + y \in span\ S$
is *module.span-add*⟨*proof*⟩

tts-lemma *span-scale*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $x \in span\ S$
shows $c *s x \in span\ S$
is *module.span-scale*⟨*proof*⟩

tts-lemma *subspace-span*:
assumes $S \subseteq U_M$
shows *subspace* (*span S*)
is *module.subspace-span*⟨*proof*⟩

tts-lemma *span-neg*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $x \in span\ S$
shows $- x \in span\ S$
is *module.span-neg*⟨*proof*⟩

tts-lemma *span-diff*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $y \in U_M$
and $x \in span\ S$
and $y \in span\ S$
shows $x - y \in span\ S$
is *module.span-diff*⟨*proof*⟩

tts-lemma *span-sum*:
assumes *range* $f \subseteq U_M$ **and** $S \subseteq U_M$ **and** $\bigwedge x.\ x \in A \Longrightarrow f\ x \in span\ S$
shows *sum* $f\ A \in span\ S$
is *module.span-sum*⟨*proof*⟩

tts-lemma *span-minimal*:
assumes $T \subseteq U_M$ **and** $S \subseteq T$ **and** *subspace* T
shows *span* $S \subseteq T$
is *module.span-minimal*⟨*proof*⟩

tts-lemma *span-subspace-induct*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $P \subseteq U_M$
and $x \in span\ S$
and *subspace* P
and $\bigwedge x.\ x \in S \Longrightarrow x \in P$
shows $x \in P$
given *module.span-subspace-induct*
⟨*proof*⟩

tts-lemma *span-induct*:
assumes $x \in U_M$
and $S \subseteq U_M$
and $x \in span\ S$
and *subspace* $\{x.\ P\ x \wedge x \in U_M\}$
and $\bigwedge x.\ x \in S \Longrightarrow P\ x$
shows $P\ x$

given *module.span-induct* ⟨*proof*⟩

tts-lemma *span-empty*: *span* {} = {0}
is *module.span-empty*⟨*proof*⟩

tts-lemma *span-subspace*:
assumes $B \subseteq U_M$ **and** $A \subseteq B$ **and** $B \subseteq$ *span* A **and** *subspace* B
shows *span* A = B
is *module.span-subspace*⟨*proof*⟩

tts-lemma *span-span*:
assumes $A \subseteq U_M$
shows *span* (*span* A) = *span* A
is *module.span-span*⟨*proof*⟩

tts-lemma *span-add-eq*:
assumes $x \in U_M$ **and** $S \subseteq U_M$ **and** $y \in U_M$ **and** $x \in$ *span* S
shows ($x + y \in$ *span* S) = ($y \in$ *span* S)
is *module.span-add-eq*⟨*proof*⟩

tts-lemma *span-add-eq*2:
assumes $y \in U_M$ **and** $S \subseteq U_M$ **and** $x \in U_M$ **and** $y \in$ *span* S
shows ($x + y \in$ *span* S) = ($x \in$ *span* S)
is *module.span-add-eq*2⟨*proof*⟩

tts-lemma *span-singleton*:
assumes $x \in U_M$
shows *span* $\{x\}$ = *range* ($\lambda k.\ k *s x$)
is *module.span-singleton*⟨*proof*⟩

tts-lemma *span-Un*:
assumes $S \subseteq U_M$ **and** $T \subseteq U_M$
shows *span* ($S \cup T$) =
$\{x \in U_M.\ \exists a{\in}U_M.\ \exists b{\in}U_M.\ x = a + b \wedge a \in$ *span* $S \wedge b \in$ *span* $T\}$
is *module.span-Un*⟨*proof*⟩

tts-lemma *span-insert*:
assumes $a \in U_M$ **and** $S \subseteq U_M$
shows *span* (*insert* a S) = $\{x \in U_M.\ \exists y.\ x - y *s a \in$ *span* $S\}$
is *module.span-insert*⟨*proof*⟩

tts-lemma *span-breakdown*:
assumes $S \subseteq U_M$ **and** $a \in U_M$ **and** $b \in S$ **and** $a \in$ *span* S
shows $\exists x.\ a - x *s b \in$ *span* ($S - \{b\}$)
is *module.span-breakdown*⟨*proof*⟩

tts-lemma *span-breakdown-eq*:
assumes $x \in U_M$ **and** $a \in U_M$ **and** $S \subseteq U_M$
shows ($x \in$ *span* (*insert* a S)) = ($\exists y.\ x - y *s a \in$ *span* S)
is *module.span-breakdown-eq*⟨*proof*⟩

tts-lemma *span-clauses*:
$[\![S \subseteq U_M;\ a \in S]\!] \Longrightarrow a \in$ *span* S
$S \subseteq U_M \Longrightarrow 0 \in$ *span* S
$[\![x \in U_M;\ S \subseteq U_M;\ y \in U_M;\ x \in$ *span* $S;\ y \in$ *span* $S]\!] \Longrightarrow x + y \in$ *span* S
$[\![x \in U_M;\ S \subseteq U_M;\ x \in$ *span* $S]\!] \Longrightarrow c *s x \in$ *span* S
is *module.span-clauses*⟨*proof*⟩

tts-lemma *span-eq-iff*:
assumes $s \subseteq U_M$
shows $(span\ s = s) = subspace\ s$
is *module.span-eq-iff*⟨*proof*⟩

tts-lemma *span-eq*:
assumes $S \subseteq U_M$ **and** $T \subseteq U_M$
shows $(span\ S = span\ T) = (S \subseteq span\ T \wedge T \subseteq span\ S)$
is *module.span-eq*⟨*proof*⟩

tts-lemma *eq-span-insert-eq*:
assumes $x \in U_M$ **and** $y \in U_M$ **and** $S \subseteq U_M$ **and** $x - y \in span\ S$
shows $span\ (insert\ x\ S) = span\ (insert\ y\ S)$
is *module.eq-span-insert-eq*⟨*proof*⟩

tts-lemma *dependent-mono*:
assumes $A \subseteq U_M$ **and** *dependent* B **and** $B \subseteq A$
shows *dependent* A
is *module.dependent-mono*⟨*proof*⟩

tts-lemma *independent-mono*:
assumes $A \subseteq U_M$ **and** $\neg$ *dependent* A **and** $B \subseteq A$
shows $\neg$ *dependent* B
is *module.independent-mono*⟨*proof*⟩

tts-lemma *dependent-zero*:
assumes $A \subseteq U_M$ **and** $0 \in A$
shows *dependent* A
is *module.dependent-zero*⟨*proof*⟩

tts-lemma *independent-empty*: $\neg$ *dependent* $\{\}$
is *module.independent-empty*⟨*proof*⟩

tts-lemma *independentD*:
assumes $s \subseteq U_M$
and $\neg$ *dependent* s
and *finite* t
and $t \subseteq s$
and $(\sum v{\in}t.\ u\ v *_s v) = 0$
and $v \in t$
shows $u\ v = 0$
is *module.independentD*⟨*proof*⟩

tts-lemma *independent-Union-directed*:
assumes $C \subseteq Pow\ U_M$
and $\bigwedge c\ d.\ [\![c \subseteq U_M;\ d \subseteq U_M;\ c \in C;\ d \in C]\!] \Longrightarrow c \subseteq d \vee d \subseteq c$
and $\bigwedge c.\ [\![c \subseteq U_M;\ c \in C]\!] \Longrightarrow \neg$ *dependent* c
shows $\neg$ *dependent* $(\bigcup\ C)$
is *module.independent-Union-directed*⟨*proof*⟩

tts-lemma *dependent-finite*:
assumes $S \subseteq U_M$ **and** *finite* S
shows *dependent* $S = (\exists x.\ (\exists y{\in}S.\ x\ y \neq 0) \wedge (\sum v{\in}S.\ x\ v *_s v) = 0)$
is *module.dependent-finite*⟨*proof*⟩

tts-lemma *independentD-alt*:
assumes $B \subseteq U_M$
and $x \in U_M$

and ¬ *dependent B*
and *finite* $\{x \in U_M.\ X\ x \neq 0\}$
and $\{x \in U_M.\ X\ x \neq 0\} \subseteq B$
and $(\sum x \mid x \in U_M \wedge X\ x \neq 0.\ X\ x *s\ x) = 0$
shows $X\ x = 0$
is *module.independentD-alt*⟨*proof*⟩

tts-lemma *spanning-subset-independent*:
assumes $A \subseteq U_M$ **and** $B \subseteq A$ **and** ¬ *dependent A* **and** $A \subseteq$ *span B*
shows $A = B$
is *module.spanning-subset-independent*⟨*proof*⟩

tts-lemma *unique-representation*:
assumes *basis* $\subseteq U_M$
and ¬ *dependent basis*
and $\bigwedge v.\ [\![v \in U_M;\ f\ v \neq 0]\!] \Longrightarrow v \in$ *basis*
and $\bigwedge v.\ [\![v \in U_M;\ g\ v \neq 0]\!] \Longrightarrow v \in$ *basis*
and *finite* $\{x \in U_M.\ f\ x \neq 0\}$
and *finite* $\{x \in U_M.\ g\ x \neq 0\}$
and
$(\sum v{\in}\{x \in U_M.\ f\ x \neq 0\}.\ f\ v *s\ v) = (\sum v{\in}\{x \in U_M.\ g\ x \neq 0\}.\ g\ v *s\ v)$
shows $\forall x{\in}U_M.\ f\ x = g\ x$
is *module.unique-representation*[*unfolded fun-eq-iff*]⟨*proof*⟩

tts-lemma *independentD-unique*:
assumes $B \subseteq U_M$
and ¬ *dependent B*
and *finite* $\{x \in U_M.\ X\ x \neq 0\}$
and $\{x \in U_M.\ X\ x \neq 0\} \subseteq B$
and *finite* $\{x \in U_M.\ Y\ x \neq 0\}$
and $\{x \in U_M.\ Y\ x \neq 0\} \subseteq B$
and $(\sum x \mid x \in U_M \wedge X\ x \neq 0.\ X\ x *s\ x) =$
$(\sum x \mid x \in U_M \wedge Y\ x \neq 0.\ Y\ x *s\ x)$
shows $\forall x{\in}U_M.\ X\ x = Y\ x$
is *module.independentD-unique*[*unfolded fun-eq-iff*]⟨*proof*⟩

tts-lemma *subspace-UNIV*: *subspace* U_M
is *module.subspace-UNIV*⟨*proof*⟩

tts-lemma *span-UNIV*: *span* $U_M = U_M$
is *module.span-UNIV*⟨*proof*⟩

tts-lemma *span-alt*:
assumes $B \subseteq U_M$
shows
span B =
{
$x \in U_M.\ \exists f.$
$x = (\sum x \mid x \in U_M \wedge f\ x \neq 0.\ f\ x *s\ x) \wedge$
finite $\{x \in U_M.\ f\ x \neq 0\} \wedge$
$\{x \in U_M.\ f\ x \neq 0\} \subseteq B$
}
is *module.span-alt*⟨*proof*⟩

tts-lemma *dependent-alt*:
assumes $B \subseteq U_M$
shows *dependent B* =
(

$\exists f.$
$\quad$ *finite* $\{v \in U_M.\ f\ v \neq 0\} \wedge$
$\quad \{v \in U_M.\ f\ v \neq 0\} \subseteq B \wedge$
$\quad (\exists\, v{\in}U_M.\ f\ v \neq 0) \wedge$
$\quad (\sum x \mid x \in U_M \wedge f\ x \neq 0.\ f\ x *s\ x) = 0$
)
is *module.dependent-alt*⟨*proof*⟩

tts-lemma *independent-alt*:
assumes $B \subseteq U_M$
shows
$(\neg$ *dependent* $B) =$
(
$\forall f.$
$\quad$ *finite* $\{x \in U_M.\ f\ x \neq 0\} \longrightarrow$
$\quad \{x \in U_M.\ f\ x \neq 0\} \subseteq B \longrightarrow$
$\quad (\sum x \mid x \in U_M \wedge f\ x \neq 0.\ f\ x *s\ x) = 0 \longrightarrow$
$\quad (\forall\, x{\in}U_M.\ f\ x = 0)$
)
is *module.independent-alt*⟨*proof*⟩

tts-lemma *subspace-Int*:
assumes *range* $s \subseteq$ *Pow* U_M **and** $\bigwedge i.\ i \in I \Longrightarrow$ *subspace* $(s\ i)$
shows *subspace* $(\bigcap\ (s\ `\ I) \cap U_M)$
is *module.subspace-Int*⟨*proof*⟩

tts-lemma *subspace-Inter*:
assumes $f \subseteq$ *Pow* U_M **and** *Ball f subspace*
shows *subspace* $(\bigcap\ f \cap U_M)$
is *module.subspace-Inter*⟨*proof*⟩

tts-lemma *module-hom-scale-self*: *module-hom-on* $U_M\ U_M\ (*s)\ (*s)\ ((*s)\ c)$
is *module.module-hom-scale-self*⟨*proof*⟩

tts-lemma *module-hom-scale-left*:
assumes $x \in U_M$
shows *module-hom-on UNIV* $U_M\ (*)\ (*s)\ (\lambda r.\ r *s\ x)$
is *module.module-hom-scale-left*⟨*proof*⟩

tts-lemma *module-hom-id*: *module-hom-on* $U_M\ U_M\ (*s)\ (*s)$ *id*
is *module.module-hom-id*⟨*proof*⟩

tts-lemma *module-hom-ident*: *module-hom-on* $U_M\ U_M\ (*s)\ (*s)\ (\lambda x.\ x)$
is *module.module-hom-ident*⟨*proof*⟩

tts-lemma *module-hom-uminus*: *module-hom-on* $U_M\ U_M\ (*s)\ (*s)$ *uminus*
is *module.module-hom-uminus*⟨*proof*⟩

end

tts-context
tts: (*?'b* **to** ‹U_M::*'b set*›)
rewriting *ctr-simps*
substituting *implicit*$_M$*.module-ow-axioms*
and *implicit*$_M$*.ab-group-add-ow-axioms*
eliminating ‹*?a* ∈ U_M› **and** ‹*?B* ⊆ U_M› **through** *clarsimp*
applying
[

```
    OF
      implicit_M.carrier-ne
      implicit_M.add-closed'
      implicit_M.minus-closed'
      implicit_M.uminus-closed'
      implicit_M.scale-closed',
    unfolded tts-implicit
  ]
begin

tts-lemma span-explicit:
  assumes b ⊆ U_M
  shows span b =
    {x ∈ U_M. ∃ y⊆U_M. ∃ f. (finite y ∧ y ⊆ b) ∧ x = (∑ a∈y. f a *s a)}
  given module.span-explicit ⟨proof⟩

tts-lemma span-unique:
  assumes S ⊆ U_M
    and T ⊆ U_M
    and S ⊆ T
    and subspace T
    and ⋀T'. ⟦T' ⊆ U_M; S ⊆ T'; subspace T'⟧ ⟹ T ⊆ T'
  shows span S = T
    is module.span-unique⟨proof⟩

tts-lemma dependent-explicit:
  assumes V ⊆ U_M
  shows dependent V =
    (∃ U⊆U_M. ∃ f. finite U ∧ U ⊆ V ∧ (∃ v∈U. f v ≠ 0) ∧ (∑ v∈U. f v *s v) = 0)
    given module.dependent-explicit ⟨proof⟩

tts-lemma independent-explicit-module:
  assumes V ⊆ U_M
  shows (¬ dependent V) =
    (
      ∀ U⊆U_M. ∀ f. ∀ v∈U_M.
        finite U ⟶
        U ⊆ V ⟶
        (∑ u∈U. f u *s u) = 0 ⟶
        v ∈ U ⟶
        f v = 0
    )
    given module.independent-explicit-module ⟨proof⟩

end

end

context module-pair-on
begin

tts-context
  tts: (?'b to ‹U_M-1::'b set›) and (?'c to ‹U_M-2::'c set›)
  rewriting ctr-simps
  substituting M_1.implicit_M.module-ow-axioms
    and M_2.implicit_M.module-ow-axioms
    and M_1.implicit_M.ab-group-add-ow-axioms
    and M_2.implicit_M.ab-group-add-ow-axioms
```

and *implicit*$_M$*.module-pair-ow-axioms*
eliminating through *auto*
applying [*unfolded tts-implicit*]
begin

tts-lemma *module-hom-zero*: *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ 0)$
is *module-pair.module-hom-zero*⟨*proof*⟩

tts-lemma *module-hom-add*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ g
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ f\ x + g\ x)$
is *module-pair.module-hom-add*⟨*proof*⟩

tts-lemma *module-hom-sub*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ g
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ f\ x - g\ x)$
is *module-pair.module-hom-sub*⟨*proof*⟩

tts-lemma *module-hom-neg*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$ **and** *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ -\ f\ x)$
is *module-pair.module-hom-neg*⟨*proof*⟩

tts-lemma *module-hom-scale*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$ **and** *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ c\ *_{s_2}\ f\ x)$
is *module-pair.module-hom-scale*⟨*proof*⟩

tts-lemma *module-hom-compose-scale*:
assumes $c \in U_{M\text{-}2}$ **and** *module-hom-on* $U_{M\text{-}1}$ *UNIV* $(*s_1)$ $(*)$ f
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ f\ x\ *_{s_2}\ c)$
is *module-pair.module-hom-compose-scale*⟨*proof*⟩

tts-lemma *module-hom-sum*:
assumes $\forall\, u.\ \forall\, v{\in}U_{M\text{-}1}.\ f\ u\ v \in U_{M\text{-}2}$
and $\bigwedge i.\ i \in I \Longrightarrow$ *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(f\ i)$
and $I = \{\} \Longrightarrow$ *module-on* $U_{M\text{-}1}$ $(*s_1)$ $\wedge$ *module-on* $U_{M\text{-}2}$ $(*s_2)$
shows *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ \sum i{\in}I.\ f\ i\ x)$
is *module-pair.module-hom-sum*⟨*proof*⟩

tts-lemma *module-hom-eq-on-span*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and $x \in U_{M\text{-}1}$
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and *module-hom-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ g
and $\bigwedge x.\ [\![x \in U_{M\text{-}1};\ x \in B]\!] \Longrightarrow f\ x = g\ x$
and $x \in M_1.span\ B$
shows $f\ x = g\ x$
is *module-pair.module-hom-eq-on-span*⟨*proof*⟩

end

end

4.4 Vector spaces

4.4.1 *vector-space-with*

locale *vector-space-with* = *ab-group-add* $plus_{VS}$ $zero_{VS}$ $minus_{VS}$ $uminus_{VS}$
  for $plus_{VS}$ :: ['*vs*, '*vs*] ⇒ '*vs* (**infixl** ‹+$_{VS}$› 65)
    and $zero_{VS}$ (‹0$_{VS}$›)
    and $minus_{VS}$ (**infixl** ‹−$_{VS}$› 65)
    and $uminus_{VS}$ (‹−$_{VS}$ -› [81] 80) +
  fixes *scale* :: ['*f*::*field*, '*vs*] ⇒ '*vs* (**infixr** ‹∗s_{with}› 75)
  assumes *scale-right-distrib*[*algebra-simps*]:
    $a *s_{with} (x +_{VS} y) = a *s_{with} x +_{VS} a *s_{with} y$
    and *scale-left-distrib*[*algebra-simps*]:
      $(a + b) *s_{with} x = a *s_{with} x +_{VS} b *s_{with} x$
    and *scale-scale*[*simp*]: $a *s_{with} (b *s_{with} x) = (a * b) *s_{with} x$
    and *scale-one*[*simp*]: $1 *s_{with} x = x$
begin

notation $plus_{VS}$ (**infixl** ‹+$_{VS}$› 65)
  and $zero_{VS}$ (‹0$_{VS}$›)
  and $minus_{VS}$ (**infixl** ‹−$_{VS}$› 65)
  and $uminus_{VS}$ (‹−$_{VS}$ -› [81] 80)
  and *scale* (**infixr** ‹∗s_{with}› 75)

end

lemma *vector-space-with-overloaded*[*ud-with*]:
  vector-space = *vector-space-with* (+) 0 (−) *uminus*
  ⟨*proof*⟩

locale *vector-space-pair-with* =
  VS_1: *vector-space-with* $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ +
  VS_2: *vector-space-with* $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$
  for $plus_{VS\text{-}1}$ :: ['*vs*-1, '*vs*-1] ⇒ '*vs*-1 (**infixl** ‹+$_{VS'\text{-}1}$› 65)
    and $zero_{VS\text{-}1}$ (‹0$_{VS'\text{-}1}$›)
    and $minus_{VS\text{-}1}$ (**infixl** ‹−$_{VS'\text{-}1}$› 65)
    and $uminus_{VS\text{-}1}$ (‹−$_{VS'\text{-}1}$ -› [81] 80)
    and $scale_1$ :: ['*f*::*field*, '*vs*-1] ⇒ '*vs*-1 (**infixr** ‹∗$s_{with'\text{-}1}$› 75)
    and $plus_{VS\text{-}2}$ :: ['*vs*-2, '*vs*-2] ⇒ '*vs*-2 (**infixl** ‹+$_{VS'\text{-}2}$› 65)
    and $zero_{VS\text{-}2}$ (‹0$_{VS'\text{-}2}$›)
    and $minus_{VS\text{-}2}$ (**infixl** ‹−$_{VS'\text{-}2}$› 65)
    and $uminus_{VS\text{-}2}$ (‹−$_{VS'\text{-}2}$ -› [81] 80)
    and $scale_2$ :: ['*f*::*field*, '*vs*-2] ⇒ '*vs*-2 (**infixr** ‹∗$s_{with'\text{-}2}$› 75)

lemma *vector-space-pair-with-overloaded*[*ud-with*]:
  vector-space-pair =
    (
      λ$scale_1$ $scale_2$.
        vector-space-pair-with (+) 0 (−) *uminus* $scale_1$ (+) 0 (−) *uminus* $scale_2$
)
  ⟨*proof*⟩

locale *linear-with* =
  VS_1: *vector-space-with* $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ +
  VS_2: *vector-space-with* $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$ +
  module-hom-with
    $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$
    $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$

f
for $plus_{VS\text{-}1}$ :: ['*vs*-1, '*vs*-1] ⇒ '*vs*-1 (**infixl** ‹$+_{VS'\text{-}1}$› 65)
and $zero_{VS\text{-}1}$ (‹$0_{VS'\text{-}1}$›)
and $minus_{VS\text{-}1}$ (**infixl** ‹$-_{VS'\text{-}1}$› 65)
and $uminus_{VS\text{-}1}$ (‹$-_{VS'\text{-}1}$ -› [81] 80)
and $scale_1$ :: ['*f*::*field*, '*vs*-1] ⇒ '*vs*-1 (**infixr** ‹$*s_{with'\text{-}1}$› 75)
and $plus_{VS\text{-}2}$ :: ['*vs*-2, '*vs*-2] ⇒ '*vs*-2 (**infixl** ‹$+_{VS'\text{-}2}$› 65)
and $zero_{VS\text{-}2}$ (‹$0_{VS'\text{-}2}$›)
and $minus_{VS\text{-}2}$ (**infixl** ‹$-_{VS'\text{-}2}$› 65)
and $uminus_{VS\text{-}2}$ (‹$-_{VS'\text{-}2}$ -› [81] 80)
and $scale_2$ :: ['*f*::*field*, '*vs*-2] ⇒ '*vs*-2 (**infixr** ‹$*s_{with'\text{-}2}$› 75)
and *f* :: '*vs*-1 ⇒ '*vs*-2

lemma *linear-with-overloaded*[*ud-with*]:
Vector-Spaces.linear =
(
$\lambda scale_1\ scale_2$.
linear-with (+) 0 (−) *uminus* $scale_1$ (+) 0 (−) *uminus* $scale_2$
)
⟨*proof*⟩

locale *finite-dimensional-vector-space-with* =
vector-space-with $plus_{VS}$ $zero_{VS}$ $minus_{VS}$ $uminus_{VS}$ *scale*
for $plus_{VS}$ :: ['*vs*, '*vs*] ⇒ '*vs*
and $zero_{VS}$
and $minus_{VS}$
and $uminus_{VS}$
and *scale* :: ['*f*::*field*, '*vs*] ⇒ '*vs* +
fixes *basis* :: '*vs set*
assumes *finite-basis*: *finite basis*
and *independent-basis*: *independent-with* 0 0_{VS} $(+_{VS})$ $(*s_{with})$ *basis*
and *span-basis*: *span.with* 0_{VS} $(+_{VS})$ $(*s_{with})$ *basis* = *UNIV*

lemma *finite-dimensional-vector-space-with-overloaded*[*ud-with*]:
finite-dimensional-vector-space =
finite-dimensional-vector-space-with (+) 0 (−) *uminus*
⟨*proof*⟩

locale *finite-dimensional-vector-space-pair*-1-*with* =
VS_1: *finite-dimensional-vector-space-with*
$plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$ +
VS_2: *vector-space-with*
$plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$
for $plus_{VS\text{-}1}$ :: ['*vs*-1, '*vs*-1] ⇒ '*vs*-1 (**infixl** ‹$+_{VS'\text{-}1}$› 65)
and $zero_{VS\text{-}1}$ (‹$0_{VS'\text{-}1}$›)
and $minus_{VS\text{-}1}$ (**infixl** ‹$-_{VS'\text{-}1}$› 65)
and $uminus_{VS\text{-}1}$ (‹$-_{VS'\text{-}1}$ -› [81] 80)
and $scale_1$ :: ['*f*::*field*, '*vs*-1] ⇒ '*vs*-1 (**infixr** ‹$*s_{with'\text{-}1}$› 75)
and $basis_1$
and $plus_{VS\text{-}2}$ :: ['*vs*-2, '*vs*-2] ⇒ '*vs*-2 (**infixl** ‹$+_{VS'\text{-}2}$› 65)
and $zero_{VS\text{-}2}$ (‹$0_{VS'\text{-}2}$›)
and $minus_{VS\text{-}2}$ (**infixl** ‹$-_{VS'\text{-}2}$› 65)
and $uminus_{VS\text{-}2}$ (‹$-_{VS'\text{-}2}$ -› [81] 80)
and $scale_2$ :: ['*f*::*field*, '*vs*-2] ⇒ '*vs*-2 (**infixr** ‹$*s_{with'\text{-}2}$› 75)

lemma *finite-dimensional-vector-space-pair*-1-*with-overloaded*[*ud-with*]:
finite-dimensional-vector-space-pair-1 =
(

$\lambda scale_1\ basis_1\ scale_2.$
finite-dimensional-vector-space-pair-1-with
$(+)\ 0\ (-)\ uminus\ scale_1\ basis_1\ (+)\ 0\ (-)\ uminus\ scale_2$
)
⟨*proof*⟩

locale *finite-dimensional-vector-space-pair-with* =
VS_1: *finite-dimensional-vector-space-with*
$plus_{VS\text{-}1}\ zero_{VS\text{-}1}\ minus_{VS\text{-}1}\ uminus_{VS\text{-}1}\ scale_1\ basis_1$ +
VS_2: *finite-dimensional-vector-space-with*
$plus_{VS\text{-}2}\ zero_{VS\text{-}2}\ minus_{VS\text{-}2}\ uminus_{VS\text{-}2}\ scale_2\ basis_2$
for $plus_{VS\text{-}1}$ :: ['*vs*-1, '*vs*-1] ⇒ '*vs*-1 (**infixl** ‹$+_{VS'\text{-}1}$› 65)
and $zero_{VS\text{-}1}$ (‹$0_{VS'\text{-}1}$›)
and $minus_{VS\text{-}1}$ (**infixl** ‹$-_{VS'\text{-}1}$› 65)
and $uminus_{VS\text{-}1}$ (‹$-_{VS'\text{-}1}$ -› [81] 80)
and $scale_1$ :: ['*f*::*field*, '*vs*-1] ⇒ '*vs*-1 (**infixr** ‹$*s_{with'\text{-}1}$› 75)
and $basis_1$
and $plus_{VS\text{-}2}$ :: ['*vs*-2, '*vs*-2] ⇒ '*vs*-2 (**infixl** ‹$+_{VS'\text{-}2}$› 65)
and $zero_{VS\text{-}2}$ (‹$0_{VS'\text{-}2}$›)
and $minus_{VS\text{-}2}$ (**infixl** ‹$-_{VS'\text{-}2}$› 65)
and $uminus_{VS\text{-}2}$ (‹$-_{VS'\text{-}2}$ -› [81] 80)
and $scale_2$ :: ['*f*::*field*, '*vs*-2] ⇒ '*vs*-2 (**infixr** ‹$*s_{with'\text{-}2}$› 75)
and $basis_2$

lemma *finite-dimensional-vector-space-pair-with-overloaded*[*ud-with*]:
finite-dimensional-vector-space-pair =
(
$\lambda scale_1\ basis_1\ scale_2\ basis_2.$
finite-dimensional-vector-space-pair-with
$(+)\ 0\ (-)\ uminus\ scale_1\ basis_1\ (+)\ 0\ (-)\ uminus\ scale_2\ basis_2$
)
⟨*proof*⟩

4.4.2 *vector-space-ow*

Definitions and common properties

Single vector space.

locale *vector-space-ow* = *ab-group-add-ow* $U_{VS}\ plus_{VS}\ zero_{VS}\ minus_{VS}\ uminus_{VS}$
for U_{VS} :: '*vs set*
and $plus_{VS}$ (**infixl** ‹$+_{VS}$› 65)
and $zero_{VS}$ (‹0_{VS}›)
and $minus_{VS}$ (**infixl** ‹$-_{VS}$› 65)
and $uminus_{VS}$ (‹$-_{VS}$ -› [81] 80) +
fixes *scale* :: ['*f*::*field*, '*vs*] ⇒ '*vs* (**infixr** ‹$*s_{ow}$› 75)
assumes *scale-closed*[*simp*, *intro*]: $x \in U_{VS} \Longrightarrow a *s_{ow} x \in U_{VS}$
and *scale-right-distrib*[*algebra-simps*]:
$[\![x \in U_{VS}; y \in U_{VS}]\!] \Longrightarrow a *s_{ow} (x +_{VS} y) = a *s_{ow} x +_{VS} a *s_{ow} y$
and *scale-left-distrib*[*algebra-simps*]:
$x \in U_{VS} \Longrightarrow (a + b) *s_{ow} x = a *s_{ow} x +_{VS} b *s_{ow} x$
and *scale-scale*[*simp*]:
$x \in U_{VS} \Longrightarrow a *s_{ow} (b *s_{ow} x) = (a * b) *s_{ow} x$
and *scale-one*[*simp*]: $x \in U_{VS} \Longrightarrow 1 *s_{ow} x = x$
begin

lemma *scale-closed'*[*simp*]: $\forall a.\ \forall x{\in}U_{VS}.\ a *s_{ow} x \in U_{VS}$ ⟨*proof*⟩

lemma *minus-closed'*[*simp*]: $\forall x{\in}U_{VS}.\ \forall y{\in}U_{VS}.\ x -_{VS} y \in U_{VS}$

⟨*proof*⟩

lemma *uminus-closed'*[*simp*]: ∀ x∈U_{VS}. $-_{VS}$ x ∈ U_{VS} ⟨*proof*⟩

tts-register-sbts ‹($*s_{ow}$)› | U_{VS}
⟨*proof*⟩

sublocale *implicit*$_{VS}$: *module-ow* U_{VS} *plus*$_{VS}$ *zero*$_{VS}$ *minus*$_{VS}$ *uminus*$_{VS}$ *scale*
⟨*proof*⟩

end

ud ‹*vector-space.dim*›
ud *dim'* ‹*dim*›

Pair of vector spaces.

locale *vector-space-pair-ow* =
VS$_1$: *vector-space-ow* $U_{VS\text{-}1}$ *plus*$_{VS\text{-}1}$ *zero*$_{VS\text{-}1}$ *minus*$_{VS\text{-}1}$ *uminus*$_{VS\text{-}1}$ *scale*$_1$ +
VS$_2$: *vector-space-ow* $U_{VS\text{-}2}$ *plus*$_{VS\text{-}2}$ *zero*$_{VS\text{-}2}$ *minus*$_{VS\text{-}2}$ *uminus*$_{VS\text{-}2}$ *scale*$_2$
for $U_{VS\text{-}1}$ :: *'vs-1 set*
and *plus*$_{VS\text{-}1}$ (**infixl** ‹+$_{VS'\text{-}1}$› 65)
and *zero*$_{VS\text{-}1}$ (‹0$_{VS'\text{-}1}$›)
and *minus*$_{VS\text{-}1}$ (**infixl** ‹−$_{VS'\text{-}1}$› 65)
and *uminus*$_{VS\text{-}1}$ (‹−$_{VS'\text{-}1}$ -› [81] 80)
and *scale*$_1$ :: [*'f*::*field*, *'vs-1*] ⇒ *'vs-1* (**infixr** ‹$*s_{ow'\text{-}1}$› 75)
and $U_{VS\text{-}2}$ :: *'vs-2 set*
and *plus*$_{VS\text{-}2}$ (**infixl** ‹+$_{VS'\text{-}2}$› 65)
and *zero*$_{VS\text{-}2}$ (‹0$_{VS'\text{-}2}$›)
and *minus*$_{VS\text{-}2}$ (**infixl** ‹−$_{VS'\text{-}2}$› 65)
and *uminus*$_{VS\text{-}2}$ (‹−$_{VS'\text{-}2}$ -› [81] 80)
and *scale*$_2$ :: [*'f*::*field*, *'vs-2*] ⇒ *'vs-2* (**infixr** ‹$*s_{ow'\text{-}2}$› 75)
begin

sublocale *implicit*$_{VS}$: *module-pair-ow*
$U_{VS\text{-}1}$ *plus*$_{VS\text{-}1}$ *zero*$_{VS\text{-}1}$ *minus*$_{VS\text{-}1}$ *uminus*$_{VS\text{-}1}$ *scale*$_1$
$U_{VS\text{-}2}$ *plus*$_{VS\text{-}2}$ *zero*$_{VS\text{-}2}$ *minus*$_{VS\text{-}2}$ *uminus*$_{VS\text{-}2}$ *scale*$_2$
⟨*proof*⟩

end

Linear map.

locale *linear-ow* =
VS$_1$: *vector-space-ow* $U_{VS\text{-}1}$ *plus*$_{VS\text{-}1}$ *zero*$_{VS\text{-}1}$ *minus*$_{VS\text{-}1}$ *uminus*$_{VS\text{-}1}$ *scale*$_1$ +
VS$_2$: *vector-space-ow* $U_{VS\text{-}2}$ *plus*$_{VS\text{-}2}$ *zero*$_{VS\text{-}2}$ *minus*$_{VS\text{-}2}$ *uminus*$_{VS\text{-}2}$ *scale*$_2$ +
module-hom-ow
$U_{VS\text{-}1}$ *plus*$_{VS\text{-}1}$ *zero*$_{VS\text{-}1}$ *minus*$_{VS\text{-}1}$ *uminus*$_{VS\text{-}1}$ *scale*$_1$
$U_{VS\text{-}2}$ *plus*$_{VS\text{-}2}$ *zero*$_{VS\text{-}2}$ *minus*$_{VS\text{-}2}$ *uminus*$_{VS\text{-}2}$ *scale*$_2$
f
for $U_{VS\text{-}1}$ :: *'vs-1 set*
and *plus*$_{VS\text{-}1}$ (**infixl** ‹+$_{VS'\text{-}1}$› 65)
and *zero*$_{VS\text{-}1}$ (‹0$_{VS'\text{-}1}$›)
and *minus*$_{VS\text{-}1}$ (**infixl** ‹−$_{VS'\text{-}1}$› 65)
and *uminus*$_{VS\text{-}1}$ (‹−$_{VS'\text{-}1}$ -› [81] 80)
and *scale*$_1$ :: [*'f*::*field*, *'vs-1*] ⇒ *'vs-1* (**infixr** ‹$*s_{ow'\text{-}1}$› 75)
and $U_{VS\text{-}2}$ :: *'vs-2 set*
and *plus*$_{VS\text{-}2}$ (**infixl** ‹+$_{VS'\text{-}2}$› 65)
and *zero*$_{VS\text{-}2}$ (‹0$_{VS'\text{-}2}$›)
and *minus*$_{VS\text{-}2}$ (**infixl** ‹−$_{VS'\text{-}2}$› 65)

and $uminus_{VS\text{-}2}$ ($\langle -_{VS\prime\text{-}2}$ -$\rangle$ [81] 80)
and $scale_2$ :: ['f::*field*, 'vs-2] ⇒ 'vs-2 (**infixr** $\langle *s_{ow\prime\text{-}2}\rangle$ 75)
and f :: 'vs-1 ⇒ 'vs-2
begin

sublocale $implicit_{VS}$: *vector-space-pair-ow*
$U_{VS\text{-}1}$ $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$
$U_{VS\text{-}2}$ $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$
⟨*proof*⟩

end

Single finite dimensional vector space.

locale *finite-dimensional-vector-space-ow* =
vector-space-ow U_{VS} $plus_{VS}$ $zero_{VS}$ $minus_{VS}$ $uminus_{VS}$ *scale*
for U_{VS} :: 'vs *set*
and $plus_{VS}$ (**infixl** $\langle +_{VS}\rangle$ 65)
and $zero_{VS}$ ($\langle 0_{VS}\rangle$)
and $minus_{VS}$ (**infixl** $\langle -_{VS}\rangle$ 65)
and $uminus_{VS}$ ($\langle -_{VS}$ -$\rangle$ [81] 80)
and *scale* :: ['f::*field*, 'vs] ⇒ 'vs (**infixr** $\langle *s_{ow}\rangle$ 75) +
fixes *basis* :: 'vs *set*
assumes *basis-closed*: *basis* ⊆ U_{VS}
and *finite-basis*: *finite basis*
and *independent-basis*: *independent-with* 0 $zero_{VS}$ $plus_{VS}$ *scale basis*
and *span-basis*: *span.with* $zero_{VS}$ $plus_{VS}$ *scale basis* = U_{VS}

Pair of finite dimensional vector spaces.

locale *finite-dimensional-vector-space-pair*-1-*ow* =
VS_1: *finite-dimensional-vector-space-ow*
$U_{VS\text{-}1}$ $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$ +
VS_2: *vector-space-ow*
$U_{VS\text{-}2}$ $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$
for $U_{VS\text{-}1}$ :: 'vs-1 *set*
and $plus_{VS\text{-}1}$ (**infixl** $\langle +_{VS\prime\text{-}1}\rangle$ 65)
and $zero_{VS\text{-}1}$ ($\langle 0_{VS\prime\text{-}1}\rangle$)
and $minus_{VS\text{-}1}$ (**infixl** $\langle -_{VS\prime\text{-}1}\rangle$ 65)
and $uminus_{VS\text{-}1}$ ($\langle -_{VS\prime\text{-}1}$ -$\rangle$ [81] 80)
and $scale_1$ :: ['f::*field*, 'vs-1] ⇒ 'vs-1 (**infixr** $\langle *s_{ow\prime\text{-}1}\rangle$ 75)
and $basis_1$
and $U_{VS\text{-}2}$ :: 'vs-2 *set*
and $plus_{VS\text{-}2}$ (**infixl** $\langle +_{VS\prime\text{-}2}\rangle$ 65)
and $zero_{VS\text{-}2}$ ($\langle 0_{VS\prime\text{-}2}\rangle$)
and $minus_{VS\text{-}2}$ (**infixl** $\langle -_{VS\prime\text{-}2}\rangle$ 65)
and $uminus_{VS\text{-}2}$ ($\langle -_{VS\prime\text{-}2}$ -$\rangle$ [81] 80)
and $scale_2$ :: ['f::*field*, 'vs-2] ⇒ 'vs-2 (**infixr** $\langle *s_{ow\prime\text{-}2}\rangle$ 75)

locale *finite-dimensional-vector-space-pair-ow* =
VS_1: *finite-dimensional-vector-space-ow*
$U_{VS\text{-}1}$ $plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$ +
VS_2: *finite-dimensional-vector-space-ow*
$U_{VS\text{-}2}$ $plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$ $basis_2$
for $U_{VS\text{-}1}$ :: 'vs-1 *set*
and $plus_{VS\text{-}1}$ (**infixl** $\langle +_{VS\prime\text{-}1}\rangle$ 65)
and $zero_{VS\text{-}1}$ ($\langle 0_{VS\prime\text{-}1}\rangle$)
and $minus_{VS\text{-}1}$ (**infixl** $\langle -_{VS\prime\text{-}1}\rangle$ 65)
and $uminus_{VS\text{-}1}$ ($\langle -_{VS\prime\text{-}1}$ -$\rangle$ [81] 80)
and $scale_1$ :: ['f::*field*, 'vs-1] ⇒ 'vs-1 (**infixr** $\langle *s_{ow\prime\text{-}1}\rangle$ 75)

and $basis_1$
and $U_{VS\text{-}2}$ :: *'vs-2 set*
and $plus_{VS\text{-}2}$ (**infixl** ‹$+_{VS'\text{-}2}$› 65)
and $zero_{VS\text{-}2}$ (‹$0_{VS'\text{-}2}$›)
and $minus_{VS\text{-}2}$ (**infixl** ‹$-_{VS'\text{-}2}$› 65)
and $uminus_{VS\text{-}2}$ (‹$-_{VS'\text{-}2}$ -› [81] 80)
and $scale_2$ :: [*'f*::*field*, *'vs-2*] ⇒ *'vs-2* (**infixr** ‹$*s_{ow'\text{-}2}$› 75)
and $basis_2$

Transfer.

lemma *vector-space-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique B right-total B*
fixes *PP lhs*
defines
PP ≡
(
(*B* ===> *B* ===> *B*) ===>
B ===>
(*B* ===> *B* ===> *B*) ===>
(*B* ===> *B*) ===>
((=) ===> *B* ===> *B*) ===>
(=)
)
and
lhs ≡
(
λ$plus_{VS}$ $zero_{VS}$ $minus_{VS}$ $uminus_{VS}$ *scale*.
vector-space-ow
(*Collect* (*Domainp B*)) $plus_{VS}$ $zero_{VS}$ $minus_{VS}$ $uminus_{VS}$ *scale*
)
shows *PP lhs vector-space-with*
⟨*proof*⟩

lemma *vector-space-pair-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B_1 *right-total* B_1 *bi-unique* B_2 *right-total* B_2
fixes *PP lhs*
defines
PP ≡
(
(B_1 ===> B_1 ===> B_1) ===>
B_1 ===>
(B_1 ===> B_1 ===> B_1) ===>
(B_1 ===> B_1) ===>
((=) ===> B_1 ===> B_1) ===>
(B_2 ===> B_2 ===> B_2) ===>
B_2 ===>
(B_2 ===> B_2 ===> B_2) ===>
(B_2 ===> B_2) ===>
((=) ===> B_2 ===> B_2) ===>
(=)
)
and
lhs ≡
(

```
      λ
        plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
        plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2.
      vector-space-pair-ow
        (Collect (Domainp B_1)) plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
        (Collect (Domainp B_2)) plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2
    )
  shows PP lhs vector-space-pair-with
⟨proof⟩

lemma linear-with-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]:
    bi-unique B_1 right-total B_1 bi-unique B_2 right-total B_2
  fixes PP lhs
  defines
    PP ≡
      (
        (B_1 ===> B_1 ===> B_1) ===>
        B_1 ===>
        (B_1 ===> B_1 ===> B_1) ===>
        (B_1 ===> B_1) ===>
        ((=) ===> B_1 ===> B_1) ===>
        (B_2 ===> B_2 ===> B_2) ===>
        B_2 ===>
        (B_2 ===> B_2 ===> B_2) ===>
        (B_2 ===> B_2) ===>
        ((=) ===> B_2 ===> B_2) ===>
        (B_1 ===> B_2) ===>
        (=)
    )
    and
      lhs ≡
        (
          λ
            plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
            plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2
            f.
          linear-ow
            (Collect (Domainp B_1)) plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
            (Collect (Domainp B_2)) plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2
            f
        )

    shows PP lhs linear-with
⟨proof⟩

lemma linear-with-transfer'[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: bi-unique B right-total B
  fixes PP lhs
  defines
    PP ≡
      (
        (B ===> B ===> B) ===>
        B ===>
        (B ===> B ===> B) ===>
        (B ===> B) ===>
```

```
      ((=) ===> B ===> B) ===>
      (B ===> B ===> B) ===>
      B ===>
      (B ===> B ===> B) ===>
      (B ===> B) ===>
      ((=) ===> B ===> B) ===>
      (B ===> B) ===>
      (=)
  )
  and
    lhs ≡
      (
        λ
          plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
          plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2
          f.
        linear-ow
          (Collect (Domainp B)) plus_VS-1 zero_VS-1 minus_VS-1 uminus_VS-1 scale_1
          (Collect (Domainp B)) plus_VS-2 zero_VS-2 minus_VS-2 uminus_VS-2 scale_2
          f
      )

  shows PP lhs linear-with
⟨proof⟩

lemma finite-dimensional-vector-space-with-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: bi-unique B right-total B
  fixes PP lhs
  defines
    PP ≡
      (
        (B ===> B ===> B) ===>
        B ===>
        (B ===> B ===> B) ===>
        (B ===> B) ===>
        ((=) ===> B ===> B) ===>
        rel-set B ===>
        (=)
      )
    and
      lhs ≡
        (
          λplus_VS zero_VS minus_VS uminus_VS scale basis.
            finite-dimensional-vector-space-ow
              (Collect (Domainp B)) plus_VS zero_VS minus_VS uminus_VS scale basis
        )
    shows PP lhs finite-dimensional-vector-space-with
⟨proof⟩

lemma finite-dimensional-vector-space-pair-1-with-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]:
    bi-unique B_1 right-total B_1 bi-unique B_2 right-total B_2
  fixes PP lhs
  defines
    PP ≡
      (
```

$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$B_1 \Longrightarrow$
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_1 \Longrightarrow B_1) \Longrightarrow$
$((=) \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
rel-set $B_1 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$B_2 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(B_2 \Longrightarrow B_2) \Longrightarrow$
$((=) \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(=)$
)
and
lhs ≡
(
λ
$plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$
$plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$.
finite-dimensional-vector-space-pair-1-*ow*
(*Collect* (*Domainp* B_1))
$plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$
(*Collect* (*Domainp* B_2))
$plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$
)
shows *PP lhs finite-dimensional-vector-space-pair*-1-*with*
⟨*proof*⟩

lemma *finite-dimensional-vector-space-pair-with-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]:
bi-unique B_1 *right-total* B_1 *bi-unique* B_2 *right-total* B_2
fixes *PP lhs*
defines
PP ≡
(
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$B_1 \Longrightarrow$
$(B_1 \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
$(B_1 \Longrightarrow B_1) \Longrightarrow$
$((=) \Longrightarrow B_1 \Longrightarrow B_1) \Longrightarrow$
rel-set $B_1 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$B_2 \Longrightarrow$
$(B_2 \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
$(B_2 \Longrightarrow B_2) \Longrightarrow$
$((=) \Longrightarrow B_2 \Longrightarrow B_2) \Longrightarrow$
rel-set $B_2 \Longrightarrow$
$(=)$
)
and
lhs ≡
(
λ
$plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$
$plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$ $basis_2$.
finite-dimensional-vector-space-pair-ow
(*Collect* (*Domainp* B_1))

$plus_{VS\text{-}1}$ $zero_{VS\text{-}1}$ $minus_{VS\text{-}1}$ $uminus_{VS\text{-}1}$ $scale_1$ $basis_1$
($Collect$ ($Domainp$ B_2))
$plus_{VS\text{-}2}$ $zero_{VS\text{-}2}$ $minus_{VS\text{-}2}$ $uminus_{VS\text{-}2}$ $scale_2$ $basis_2$
)
shows *PP lhs finite-dimensional-vector-space-pair-with*
⟨*proof*⟩

4.4.3 *vector-space-on*

locale *vector-space-on* = *module-on* U_{VS} *scale*
for U_{VS} **and** *scale* :: *'a*::*field* ⇒ *'b*::*ab-group-add* ⇒ *'b* (**infixr** ‹∗s› 75)
begin

notation *scale* (**infixr** ‹∗s› 75)

sublocale $implicit_{VS}$: *vector-space-ow* U_{VS} ‹(+)› 0 ‹(−)› *uminus scale*
⟨*proof*⟩

lemmas *ab-group-add-ow-axioms* = $implicit_M$.*ab-group-add-ow-axioms*
lemmas *vector-space-ow-axioms* = $implicit_{VS}$.*vector-space-ow-axioms*

definition *dim* :: *'b set* ⇒ *nat*
where *dim V* = (*if* ∃ *b*⊆U_{VS}. ¬ *dependent b* ∧ *span b* = *span V*
then card (*SOME b*. *b* ⊆ U_{VS} ∧ ¬ *dependent b* ∧ *span b* = *span V*)
else 0)

end

lemma *vector-space-on-alt-def*: *vector-space-on* U_{VS} = *module-on* U_{VS}
⟨*proof*⟩

lemma *implicit-vector-space-ow*[*tts-implicit*]:
vector-space-ow U_{VS} (+) 0 (−) *uminus* = *vector-space-on* U_{VS}
⟨*proof*⟩

locale *linear-on* =
VS_1: *vector-space-on* $U_{M\text{-}1}$ $scale_1$ +
VS_2: *vector-space-on* $U_{M\text{-}2}$ $scale_2$ +
module-hom-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ *f*
for $U_{M\text{-}1}$ $U_{M\text{-}2}$ **and** $scale_1$::*'a*::*field* ⇒ *'b* ⇒ *'b*::*ab-group-add*
and $scale_2$::*'a*::*field* ⇒ *'c* ⇒ *'c*::*ab-group-add*
and *f*

lemma *implicit-linear-on*[*tts-implicit*]:
linear-ow $U_{M\text{-}1}$ (+) 0 *minus uminus* $scale_1$ $U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ =
linear-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$
⟨*proof*⟩

locale *finite-dimensional-vector-space-on* =
vector-space-on U_{VS} *scale*
for U_{VS} :: *'b*::*ab-group-add set*
and *scale* :: *'a*::*field* ⇒ *'b* ⇒ *'b* +
fixes *basis* :: *'b set*
assumes *finite-basis*: *finite basis*
and *independent-basis*: ¬ *dependent basis*
and *span-basis*: *span basis* = U_{VS}
and *basis-subset*: *basis* ⊆ U_{VS}
begin

sublocale *implicit*$_{VS}$:
finite-dimensional-vector-space-ow U_{VS} ‹(+)› 0 ‹(−)› *uminus scale basis*
⟨*proof*⟩

end

lemma *implicit-finite-dimensional-vector-space-on*[*tts-implicit*]:
finite-dimensional-vector-space-ow U_{VS} (+) 0 *minus uminus scale basis* =
finite-dimensional-vector-space-on U_{VS} *scale basis*
⟨*proof*⟩

locale *vector-space-pair-on* =
VS_1: *vector-space-on* $U_{M\text{-}1}$ $scale_1$ +
VS_2: *vector-space-on* $U_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$:: *'b*::*ab-group-add set* **and** $U_{M\text{-}2}$::*'c*::*ab-group-add set*
and $scale_1$::*'a*::*field* ⇒ - ⇒ - (**infixr** ‹∗s_1› 75)
and $scale_2$::*'a* ⇒ - ⇒ - (**infixr** ‹∗s_2› 75)
begin

notation $scale_1$ (**infixr** ‹∗s_1› 75)
notation $scale_2$ (**infixr** ‹∗s_2› 75)

sublocale *module-pair-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ ⟨*proof*⟩

sublocale *implicit*$_{VS}$:
vector-space-pair-ow
$U_{M\text{-}1}$ ‹(+)› 0 ‹(−)› *uminus* $scale_1$
$U_{M\text{-}2}$ ‹(+)› 0 ‹(−)› *uminus* $scale_2$
⟨*proof*⟩

end

lemma *implicit-vector-space-pair-on*[*tts-implicit*]:
vector-space-pair-ow
$U_{M\text{-}1}$ (+) 0 (−) *uminus* $scale_1$
$U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ =
vector-space-pair-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$
⟨*proof*⟩

locale *finite-dimensional-vector-space-pair*-1-*on* =
VS_1: *finite-dimensional-vector-space-on* $U_{M\text{-}1}$ $scale_1$ *basis*1 +
VS_2: *vector-space-on* $U_{M\text{-}2}$ $scale_2$
for $U_{M\text{-}1}$ $U_{M\text{-}2}$
and $scale_1$::*'a*::*field* ⇒ *'b*::*ab-group-add* ⇒ *'b*
and $scale_2$::*'a*::*field* ⇒ *'c*::*ab-group-add* ⇒ *'c*
and *basis*1
begin

sublocale *vector-space-pair-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ ⟨*proof*⟩

sublocale *implicit*$_{VS}$:
finite-dimensional-vector-space-pair-1-*ow*
$U_{M\text{-}1}$ ‹(+)› 0 ‹(−)› *uminus* $scale_1$ *basis*1 $U_{M\text{-}2}$ ‹(+)› 0 ‹(−)› *uminus* $scale_2$
⟨*proof*⟩

end

lemma *implicit-finite-dimensional-vector-space-pair-1-on[tts-implicit]*:
finite-dimensional-vector-space-pair-1-ow
$U_{M\text{-}1}$ (+) 0 *minus uminus* $scale_1$ *basis*1 $U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ =
finite-dimensional-vector-space-pair-1-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ *basis*1
⟨*proof*⟩

locale *finite-dimensional-vector-space-pair-on* =
VS_1: *finite-dimensional-vector-space-on* $U_{M\text{-}1}$ $scale_1$ *basis*1 +
VS_2: *finite-dimensional-vector-space-on* $U_{M\text{-}2}$ $scale_2$ *basis*2
for $U_{M\text{-}1}$ $U_{M\text{-}2}$
and $scale_1$::'a::*field* ⇒ 'b::*ab-group-add* ⇒ 'b
and $scale_2$::'a::*field* ⇒ 'c::*ab-group-add* ⇒ 'c
and *basis*1 *basis*2
begin

sublocale *finite-dimensional-vector-space-pair-1-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$
⟨*proof*⟩

sublocale $implicit_{VS}$:
finite-dimensional-vector-space-pair-ow
$U_{M\text{-}1}$ ‹(+)› 0 ‹(−)› *uminus* $scale_1$ *basis*1
$U_{M\text{-}2}$ ‹(+)› 0 ‹(−)› *uminus* $scale_2$ *basis*2
⟨*proof*⟩

end

lemma *implicit-finite-dimensional-vector-space-pair-on[tts-implicit]*:
finite-dimensional-vector-space-pair-ow
$U_{M\text{-}1}$ (+) 0 *minus uminus* $scale_1$ *basis*1
$U_{M\text{-}2}$ (+) 0 (−) *uminus* $scale_2$ *basis*2 =
finite-dimensional-vector-space-pair-on
$U_{M\text{-}1}$ $U_{M\text{-}2}$ $scale_1$ $scale_2$ *basis*1 *basis*2
⟨*proof*⟩

4.4.4 Relativization : part I

context *vector-space-on*
begin

tts-context
tts: (?'b **to** ‹U_{VS}::'b *set*›)
rewriting *ctr-simps*
substituting *ab-group-add-ow-axioms*
and *vector-space-ow-axioms*
and $implicit_M$.*module-ow-axioms*
applying
[
OF
$implicit_M$.*carrier-ne*
$implicit_M$.*add-closed'*
$implicit_M$.*zero-closed*
$implicit_{VS}$.*minus-closed'*
$implicit_{VS}$.*uminus-closed'*
$implicit_{VS}$.*scale-closed'*,
unfolded tts-implicit
]
begin

tts-lemma *linear-id*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ *id*
is *vector-space.linear-id*⟨*proof*⟩

tts-lemma *linear-ident*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ $(\lambda x.\ x)$
is *vector-space.linear-ident*⟨*proof*⟩

tts-lemma *linear-scale-self*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ $((*s)\ c)$
is *vector-space.linear-scale-self*⟨*proof*⟩

tts-lemma *linear-scale-left*:
assumes $x \in U_{VS}$
shows *linear-on UNIV* U_{VS} $(*)$ $(*s)$ $(\lambda r.\ r\ *s\ x)$
is *vector-space.linear-scale-left*⟨*proof*⟩

tts-lemma *linear-uminus*: *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ *uminus*
is *vector-space.linear-uminus*⟨*proof*⟩

tts-lemma *linear-imp-scale*[*consumes* − 1, *case-names* 1]:
assumes *range* $D \subseteq U_{VS}$
and *linear-on UNIV* U_{VS} $(*)$ $(*s)$ D
and $\bigwedge d.\ [\![d \in U_{VS};\ D = (\lambda x.\ x\ *s\ d)]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *vector-space.linear-imp-scale*⟨*proof*⟩

tts-lemma *scale-eq*-0-*iff*:
assumes $x \in U_{VS}$
shows $(a\ *s\ x = 0) = (a = 0 \vee x = 0)$
is *vector-space.scale-eq*-0-*iff*⟨*proof*⟩

tts-lemma *scale-left-imp-eq*:
assumes $x \in U_{VS}$ **and** $y \in U_{VS}$ **and** $a \neq 0$ **and** $a\ *s\ x = a\ *s\ y$
shows $x = y$
is *vector-space.scale-left-imp-eq*⟨*proof*⟩

tts-lemma *scale-right-imp-eq*:
assumes $x \in U_{VS}$ **and** $x \neq 0$ **and** $a\ *s\ x = b\ *s\ x$
shows $a = b$
is *vector-space.scale-right-imp-eq*⟨*proof*⟩

tts-lemma *scale-cancel-left*:
assumes $x \in U_{VS}$ **and** $y \in U_{VS}$
shows $(a\ *s\ x = a\ *s\ y) = (x = y \vee a = 0)$
is *vector-space.scale-cancel-left*⟨*proof*⟩

tts-lemma *scale-cancel-right*:
assumes $x \in U_{VS}$
shows $(a\ *s\ x = b\ *s\ x) = (a = b \vee x = 0)$
is *vector-space.scale-cancel-right*⟨*proof*⟩

tts-lemma *injective-scale*:
assumes $c \neq 0$
shows *inj-on* $((*s)\ c)$ U_{VS}
is *vector-space.injective-scale*⟨*proof*⟩

tts-lemma *dependent-def*:
assumes $P \subseteq U_{VS}$
shows *dependent* $P = (\exists x{\in}P.\ x \in span\ (P - \{x\}))$
is *vector-space.dependent-def*⟨*proof*⟩

tts-lemma *dependent-single*:
assumes $x \in U_{VS}$
shows *dependent* $\{x\} = (x = 0)$
is *vector-space.dependent-single*⟨*proof*⟩

tts-lemma *in-span-insert*:
assumes $a \in U_{VS}$
and $b \in U_{VS}$
and $S \subseteq U_{VS}$
and $a \in$ *span* (*insert* b S)
and $a \notin$ *span* S
shows $b \in$ *span* (*insert* a S)
is *vector-space.in-span-insert*⟨*proof*⟩

tts-lemma *dependent-insertD*:
assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $a \notin$ *span* S **and** *dependent* (*insert* a S)
shows *dependent* S
is *vector-space.dependent-insertD*⟨*proof*⟩

tts-lemma *independent-insertI*:
assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $a \notin$ *span* S **and** $\neg$ *dependent* S
shows $\neg$ *dependent* (*insert* a S)
is *vector-space.independent-insertI*⟨*proof*⟩

tts-lemma *independent-insert*:
assumes $a \in U_{VS}$ **and** $S \subseteq U_{VS}$
shows ($\neg$ *dependent* (*insert* a S)) =
(*if* $a \in S$ *then* $\neg$ *dependent* S *else* $\neg$ *dependent* $S \wedge a \notin$ *span* S)
is *vector-space.independent-insert*⟨*proof*⟩

tts-lemma *maximal-independent-subset-extend*[*consumes* − 1, *case-names* 1]:
assumes $S \subseteq U_{VS}$
and $V \subseteq U_{VS}$
and $S \subseteq V$
and $\neg$ *dependent* S
and $\bigwedge B.\ [\![B \subseteq U_{VS};\ S \subseteq B;\ B \subseteq V;\ \neg$ *dependent* $B;\ V \subseteq$ *span* $B]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *vector-space.maximal-independent-subset-extend*⟨*proof*⟩

tts-lemma *maximal-independent-subset*[*consumes* − 1, *case-names* 1]:
assumes $V \subseteq U_{VS}$
and $\bigwedge B.\ [\![B \subseteq U_{VS};\ B \subseteq V;\ \neg$ *dependent* $B;\ V \subseteq$ *span* $B]\!] \Longrightarrow$ *thesis*
shows *thesis*
is *vector-space.maximal-independent-subset*⟨*proof*⟩

tts-lemma *in-span-delete*:
assumes $a \in U_{VS}$
and $S \subseteq U_{VS}$
and $b \in U_{VS}$
and $a \in$ *span* S
and $a \notin$ *span* ($S - \{b\}$)
shows $b \in$ *span* (*insert* a ($S - \{b\}$))
is *vector-space.in-span-delete*⟨*proof*⟩

tts-lemma *span-redundant*:
assumes $x \in U_{VS}$ **and** $S \subseteq U_{VS}$ **and** $x \in$ *span* S
shows *span* (*insert* x S) = *span* S

is *vector-space.span-redundant*⟨*proof*⟩

tts-lemma *span-trans*:
assumes $x \in U_{VS}$
and $S \subseteq U_{VS}$
and $y \in U_{VS}$
and $x \in span\ S$
and $y \in span\ (insert\ x\ S)$
shows $y \in span\ S$
is *vector-space.span-trans*⟨*proof*⟩

tts-lemma *span-insert-0*:
assumes $S \subseteq U_{VS}$
shows $span\ (insert\ 0\ S) = span\ S$
is *vector-space.span-insert-0*⟨*proof*⟩

tts-lemma *span-delete-0*:
assumes $S \subseteq U_{VS}$
shows $span\ (S - \{0\}) = span\ S$
is *vector-space.span-delete-0*⟨*proof*⟩

tts-lemma *span-image-scale*:
assumes $S \subseteq U_{VS}$ **and** *finite* S **and** $\bigwedge x.\ [\![x \in U_{VS};\ x \in S]\!] \Longrightarrow c\ x \neq 0$
shows $span\ ((\lambda x.\ c\ x\ *s\ x)\ `\ S) = span\ S$
is *vector-space.span-image-scale*⟨*proof*⟩

tts-lemma *exchange-lemma*:
assumes $T \subseteq U_{VS}$
and $S \subseteq U_{VS}$
and *finite* T
and $\neg dependent\ S$
and $S \subseteq span\ T$
shows $\exists t' \in Pow\ U_{VS}.$
$card\ t' = card\ T \wedge finite\ t' \wedge S \subseteq t' \wedge t' \subseteq S \cup T \wedge S \subseteq span\ t'$
is *vector-space.exchange-lemma*⟨*proof*⟩

tts-lemma *independent-span-bound*:
assumes $T \subseteq U_{VS}$
and $S \subseteq U_{VS}$
and *finite* T
and $\neg\ dependent\ S$
and $S \subseteq span\ T$
shows $finite\ S \wedge card\ S \leq card\ T$
is *vector-space.independent-span-bound*⟨*proof*⟩

tts-lemma *independent-explicit-finite-subsets*:
assumes $A \subseteq U_{VS}$
shows $(\neg\ dependent\ A) =$
(
$\forall x \subseteq U_{VS}.$
$x \subseteq A \longrightarrow$
$finite\ x \longrightarrow$
$(\forall f.\ (\sum v \in x.\ f\ v\ *s\ v) = 0 \longrightarrow (\forall x \in x.\ f\ x = 0))$
)
given *vector-space.independent-explicit-finite-subsets* ⟨*proof*⟩

tts-lemma *independent-if-scalars-zero*:
assumes $A \subseteq U_{VS}$

and *finite A*
and $\bigwedge f\ x.\ [\![x \in U_{VS};\ (\sum x{\in}A.\ f\ x\ *s\ x) = 0;\ x \in A]\!] \Longrightarrow f\ x = 0$
shows $\neg$ *dependent A*
is *vector-space.independent-if-scalars-zero*⟨*proof*⟩

tts-lemma *subspace-sums*:
assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** *subspace S* **and** *subspace T*
shows *subspace* $\{x \in U_{VS}.\ \exists\, a{\in}U_{VS}.\ \exists\, b{\in}U_{VS}.\ x = a + b \wedge a \in S \wedge b \in T\}$
is *vector-space.subspace-sums*⟨*proof*⟩

tts-lemma *bij-if-span-eq-span-bases*:
assumes $B \subseteq U_{VS}$
and $C \subseteq U_{VS}$
and $\neg$*dependent B*
and $\neg$*dependent C*
and *span B* = *span C*
shows $\exists\, x.$ *bij-betw x B C* $\wedge$ $(\forall\, a{\in}U_{VS}.\ x\ a \in U_{VS})$
given *vector-space.bij-if-span-eq-span-bases* ⟨*proof*⟩

end

end

4.4.5 Transfer: *dim*

context *vector-space-on*
begin

lemma *dim-eq-card*:
assumes $B \subseteq U_{VS}$
and $V \subseteq U_{VS}$
and *BV*: *span B* = *span V*
and *B*: $\neg$*dependent B*
shows *dim V* = *card B*
⟨*proof*⟩

lemma *dim-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-domain-rule*]: *Domainp A* = $(\lambda x.\ x \in U_{VS})$
and [*transfer-rule*]: *right-total A bi-unique A*
and [*transfer-rule*]: (*A* ===> *A* ===> *A*) *plus plus′*
and [*transfer-rule*]: ((=) ===> *A* ===> *A*) *scale scale′*
and [*transfer-rule*]: *A* 0 *zero′*
shows (*rel-set A* ===> (=)) *dim* (*dim.with plus′ zero′* 0 *scale′*)
⟨*proof*⟩

end

4.4.6 Relativization: part II

context *vector-space-on*
begin

tts-context
tts: (*?′b* **to** ‹U_{VS}::*′b set*›)
sbterms: (‹(+)::*?′b*::*ab-group-add*⇒*?′b*⇒*?′b*› **to** ‹(+)::*′b*⇒*′b*⇒*′b*›)
and
(

```
      ‹?scale::?'a::field ⇒?'b::ab-group-add⇒?'b::ab-group-add› to
      ‹(*s)::'a⇒'b⇒'b›
    )
  and (‹0::?'b::ab-group-add› to ‹0::'b›)
 rewriting ctr-simps
 substituting ab-group-add-ow-axioms
  and vector-space-ow-axioms
  and implicit_M.module-ow-axioms
 eliminating ‹?a ∈ ?A› and ‹?B ⊆ ?C› through auto
 applying
  [
   OF
    implicit_M.carrier-ne
    implicit_VS.minus-closed'
    implicit_VS.uminus-closed',
   unfolded tts-implicit
  ]
begin

tts-lemma dim-unique:
 assumes V ⊆ U_VS
  and B ⊆ V
  and V ⊆ span B
  and ¬ dependent B
  and card B = n
 shows dim V = n
  is vector-space.dim-unique⟨proof⟩

tts-lemma basis-card-eq-dim:
 assumes V ⊆ U_VS and B ⊆ V and V ⊆ span B and ¬ dependent B
 shows card B = dim V
  is vector-space.basis-card-eq-dim⟨proof⟩

tts-lemma dim-eq-card-independent:
 assumes B ⊆ U_VS and ¬ dependent B
 shows dim B = card B
  is vector-space.dim-eq-card-independent⟨proof⟩

tts-lemma dim-span:
 assumes S ⊆ U_VS
 shows dim (span S) = dim S
  is vector-space.dim-span⟨proof⟩

tts-lemma dim-span-eq-card-independent:
 assumes B ⊆ U_VS and ¬ dependent B
 shows dim (span B) = card B
  is vector-space.dim-span-eq-card-independent⟨proof⟩

tts-lemma dim-le-card:
 assumes V ⊆ U_VS and W ⊆ U_VS and V ⊆ span W and finite W
 shows dim V ≤ card W
  is vector-space.dim-le-card⟨proof⟩

tts-lemma span-eq-dim:
 assumes S ⊆ U_VS and T ⊆ U_VS and span S = span T
 shows dim S = dim T
  is vector-space.span-eq-dim⟨proof⟩
```

tts-lemma *dim-le-card′*:
assumes $s \subseteq U_{VS}$ **and** *finite s*
shows *dim s* $\leq$ *card s*
is *vector-space.dim-le-card′⟨proof⟩*

tts-lemma *span-card-ge-dim*:
assumes $V \subseteq U_{VS}$ **and** $B \subseteq V$ **and** $V \subseteq$ *span B* **and** *finite B*
shows *dim V* $\leq$ *card B*
is *vector-space.span-card-ge-dim⟨proof⟩*

end

tts-context
tts: (*?′b* **to** ‹U_{VS}::*′b set*›)
sbterms: (‹(+)::*?′b*::*ab-group-add*⇒*?′b*⇒*?′b*› **to** ‹(+)::*′b*⇒*′b*⇒*′b*›)
and (‹*?scale*::*?′a*::*field* ⇒*?′b*::*ab-group-add*⇒*?′b*› **to** ‹(∗*s*)::*′a*⇒*′b*⇒*′b*›)
and (‹0::*?′b*::*ab-group-add*› **to** ‹0::*′b*›)
rewriting *ctr-simps*
substituting *ab-group-add-ow-axioms*
and *vector-space-ow-axioms*
and *implicit*$_M$*.module-ow-axioms*
applying
[
OF
implicit$_M$*.carrier-ne*
implicit$_{VS}$*.minus-closed′*
implicit$_{VS}$*.uminus-closed′*,
unfolded tts-implicit
]
begin

tts-lemma *basis-exists*:
assumes $V \subseteq U_{VS}$
and $\bigwedge B.$
⟦
$B \subseteq U_{VS}$;
$B \subseteq V$;
¬ *dependent B*;
$V \subseteq$ *span B*;
card B = *dim V*
⟧ ⟹ *thesis*
shows *thesis*
is *vector-space.basis-exists⟨proof⟩*

end

end

context *finite-dimensional-vector-space-on*
begin

tts-context
tts: (*?′b* **to** ‹U_{VS}::*′b set*›)
sbterms: (‹(+)::*?′b*::*ab-group-add*⇒*?′b*⇒*?′b*› **to** ‹(+)::*′b*⇒*′b*⇒*′b*›)
and (‹*?scale*::*?′a*::*field* ⇒*?′b*::*ab-group-add*⇒*?′b*› **to** ‹(∗*s*)::*′a*⇒*′b*⇒*′b*›)
and (‹0::*?′b*::*ab-group-add*› **to** ‹0::*′b*›)
rewriting *ctr-simps*
substituting *ab-group-add-ow-axioms*

```
    and vector-space-ow-axioms
    and implicit_VS.finite-dimensional-vector-space-ow-axioms
    and implicit_M.module-ow-axioms
  eliminating ‹?a ∈ ?A› and ‹?B ⊆ ?C› through auto
  applying
    [
      OF
        implicit_M.carrier-ne
        implicit_VS.minus-closed'
        implicit_VS.uminus-closed'
        basis-subset,
      unfolded tts-implicit
    ]
begin

tts-lemma finiteI-independent:
  assumes B ⊆ U_VS and ¬ dependent B
  shows finite B
  is finite-dimensional-vector-space.finiteI-independent⟨proof⟩

tts-lemma dim-empty: dim {} = 0
  is finite-dimensional-vector-space.dim-empty⟨proof⟩

tts-lemma dim-insert:
  assumes x ∈ U_VS and S ⊆ U_VS
  shows dim (insert x S) = (if x ∈ span S then dim S else dim S + 1)
    is finite-dimensional-vector-space.dim-insert⟨proof⟩

tts-lemma dim-singleton:
  assumes x ∈ U_VS
  shows dim {x} = (if x = 0 then 0 else 1)
    is finite-dimensional-vector-space.dim-singleton⟨proof⟩

tts-lemma choose-subspace-of-subspace[consumes − 1, case-names 1]:
  assumes S ⊆ U_VS
    and n ≤ dim S
    and ⋀T. [[T ⊆ U_VS; subspace T; T ⊆ span S; dim T = n]] ⟹ thesis
  shows thesis
    is finite-dimensional-vector-space.choose-subspace-of-subspace⟨proof⟩

tts-lemma basis-subspace-exists[consumes − 1, case-names 1]:
  assumes S ⊆ U_VS
    and subspace S
    and ⋀B.
      [[
        B ⊆ U_VS;
        finite B;
        B ⊆ S;
        ¬ dependent B;
        span B = S;
        card B = dim S
      ]] ⟹ thesis
  shows thesis
    is finite-dimensional-vector-space.basis-subspace-exists⟨proof⟩

tts-lemma dim-mono:
  assumes V ⊆ U_VS and W ⊆ U_VS and V ⊆ span W
  shows dim V ≤ dim W
```

is *finite-dimensional-vector-space.dim-mono⟨proof⟩*

tts-lemma *dim-subset*:
assumes $T \subseteq U_{VS}$ **and** $S \subseteq T$
shows *dim* $S \leq$ *dim* T
is *finite-dimensional-vector-space.dim-subset⟨proof⟩*

tts-lemma *dim-eq-0*:
assumes $S \subseteq U_{VS}$
shows (*dim* $S = 0$) = ($S \subseteq \{0\}$)
is *finite-dimensional-vector-space.dim-eq-0⟨proof⟩*

tts-lemma *dim-UNIV*: *dim* U_{VS} = *card basis*
is *finite-dimensional-vector-space.dim-UNIV⟨proof⟩*

tts-lemma *independent-card-le-dim*:
assumes $V \subseteq U_{VS}$ **and** $B \subseteq V$ **and** ¬ *dependent B*
shows *card* $B \leq$ *dim* V
is *finite-dimensional-vector-space.independent-card-le-dim⟨proof⟩*

tts-lemma *card-ge-dim-independent*:
assumes $V \subseteq U_{VS}$ **and** $B \subseteq V$ **and** ¬ *dependent B* **and** *dim* $V \leq$ *card* B
shows $V \subseteq$ *span B*
is *finite-dimensional-vector-space.card-ge-dim-independent⟨proof⟩*

tts-lemma *card-le-dim-spanning*:
assumes $V \subseteq U_{VS}$
and $B \subseteq V$
and $V \subseteq$ *span B*
and *finite B*
and *card* $B \leq$ *dim* V
shows ¬ *dependent B*
is *finite-dimensional-vector-space.card-le-dim-spanning⟨proof⟩*

tts-lemma *card-eq-dim*:
assumes $V \subseteq U_{VS}$ **and** $B \subseteq V$ **and** *card* B = *dim* V **and** *finite B*
shows (¬ *dependent B*) = ($V \subseteq$ *span B*)
is *finite-dimensional-vector-space.card-eq-dim⟨proof⟩*

tts-lemma *subspace-dim-equal*:
assumes $T \subseteq U_{VS}$
and *subspace S*
and *subspace T*
and $S \subseteq T$
and *dim* $T \leq$ *dim* S
shows $S = T$
is *finite-dimensional-vector-space.subspace-dim-equal⟨proof⟩*

tts-lemma *dim-eq-span*:
assumes $T \subseteq U_{VS}$ **and** $S \subseteq T$ **and** *dim* $T \leq$ *dim* S
shows *span* S = *span* T
is *finite-dimensional-vector-space.dim-eq-span⟨proof⟩*

tts-lemma *dim-psubset*:
assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** *span* $S \subset$ *span* T
shows *dim* $S <$ *dim* T
is *finite-dimensional-vector-space.dim-psubset⟨proof⟩*

tts-lemma *indep-card-eq-dim-span*:
assumes $B \subseteq U_{VS}$ **and** $\neg$ *dependent* B
shows *finite* $B \wedge$ *card* $B =$ *dim* (*span* B)
is *finite-dimensional-vector-space.indep-card-eq-dim-span*⟨*proof*⟩

tts-lemma *independent-bound-general*:
assumes $S \subseteq U_{VS}$ **and** $\neg$ *dependent* S
shows *finite* $S \wedge$ *card* $S \leq$ *dim* S
is *finite-dimensional-vector-space.independent-bound-general*⟨*proof*⟩

tts-lemma *independent-explicit*:
assumes $B \subseteq U_{VS}$
shows ($\neg$ *dependent* B) =
(*finite* $B \wedge (\forall x.\ (\sum v{\in}B.\ x\ v *s\ v) = 0 \longrightarrow (\forall a{\in}B.\ x\ a = 0))$)
is *finite-dimensional-vector-space.independent-explicit*⟨*proof*⟩

tts-lemma *dim-sums-Int*:
assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** *subspace* S **and** *subspace* T
shows
dim $\{x \in U_{VS}.\ \exists y{\in}U_{VS}.\ \exists z{\in}U_{VS}.\ x = y + z \wedge y \in S \wedge z \in T\}$ + *dim* $(S \cap T)$ =
dim S + *dim* T
is *finite-dimensional-vector-space.dim-sums-Int*⟨*proof*⟩

tts-lemma *dependent-biggerset-general*:
assumes $S \subseteq U_{VS}$ **and** *finite* $S \Longrightarrow$ *dim* $S <$ *card* S
shows *dependent* S
is *finite-dimensional-vector-space.dependent-biggerset-general*⟨*proof*⟩

tts-lemma *subset-le-dim*:
assumes $S \subseteq U_{VS}$ **and** $T \subseteq U_{VS}$ **and** $S \subseteq$ *span* T
shows *dim* $S \leq$ *dim* T
is *finite-dimensional-vector-space.subset-le-dim*⟨*proof*⟩

tts-lemma *linear-inj-imp-surj*:
assumes $\forall x{\in}U_{VS}.\ f\ x \in U_{VS}$
and *linear-on* $U_{VS}\ U_{VS}\ (*s)\ (*s)\ f$
and *inj-on* $f\ U_{VS}$
shows f ‘ $U_{VS} = U_{VS}$
is *finite-dimensional-vector-space.linear-inj-imp-surj*⟨*proof*⟩

tts-lemma *linear-surj-imp-inj*:
assumes $\forall x{\in}U_{VS}.\ f\ x \in U_{VS}$
and *linear-on* $U_{VS}\ U_{VS}\ (*s)\ (*s)\ f$
and f ‘ $U_{VS} = U_{VS}$
shows *inj-on* $f\ U_{VS}$
is *finite-dimensional-vector-space.linear-surj-imp-inj*⟨*proof*⟩

tts-lemma *linear-inverse-left*:
assumes $\forall x{\in}U_{VS}.\ f\ x \in U_{VS}$
and $\forall x{\in}U_{VS}.\ f'\ x \in U_{VS}$
and *linear-on* $U_{VS}\ U_{VS}\ (*s)\ (*s)\ f$
and *linear-on* $U_{VS}\ U_{VS}\ (*s)\ (*s)\ f'$
shows $(\forall x{\in}U_{VS}.\ (f \circ f')\ x = id\ x) = (\forall x{\in}U_{VS}.\ (f' \circ f)\ x = id\ x)$
is *finite-dimensional-vector-space.linear-inverse-left*[*unfolded fun-eq-iff*]⟨*proof*⟩

tts-lemma *left-inverse-linear*:
assumes $\forall x{\in}U_{VS}.\ f\ x \in U_{VS}$
and $\forall x{\in}U_{VS}.\ g\ x \in U_{VS}$

and *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ f
and $\forall\, x{\in}U_{VS}.\ (g \circ f)\ x = id\ x$
shows *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ g
is *finite-dimensional-vector-space.left-inverse-linear*[*unfolded fun-eq-iff*]⟨*proof*⟩

tts-lemma *right-inverse-linear*:
assumes $\forall\, x{\in}U_{VS}.\ f\ x \in U_{VS}$
and $\forall\, x{\in}U_{VS}.\ g\ x \in U_{VS}$
and *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ f
and $\forall\, x{\in}U_{VS}.\ (f \circ g)\ x = id\ x$
shows *linear-on* U_{VS} U_{VS} $(*s)$ $(*s)$ g
is *finite-dimensional-vector-space.right-inverse-linear*[*unfolded fun-eq-iff*]⟨*proof*⟩

end

end

context *vector-space-pair-on*
begin

tts-context
tts: (*?'b* **to** ‹$U_{M\text{-}1}$::*'b set*›) **and** (*?'c* **to** ‹$U_{M\text{-}2}$::*'c set*›)
sbterms: (‹(+)::*?'b*::*ab-group-add*⇒*?'b*⇒*?'b*› **to** ‹(+)::*'b*⇒*'b*⇒*'b*›)
and (‹*?s*1.0::*?'a*::*field* ⇒*?'b*::*ab-group-add*⇒*?'b*› **to** ‹$(*s_1)$::*'a*⇒*'b*⇒*'b*›)
and (‹0::*?'b*::*ab-group-add*› **to** ‹0::*'b*›)
and (‹(+)::*?'c*::*ab-group-add*⇒*?'c*⇒*?'c*› **to** ‹(+)::*'c*⇒*'c*⇒*'c*›)
and (‹*?s*2.0::*?'a*::*field* ⇒*?'c*::*ab-group-add*⇒*?'c*› **to** ‹$(*s_2)$::*'a*⇒*'c*⇒*'c*›)
and (‹0::*?'c*::*ab-group-add*› **to** ‹0::*'c*›)
rewriting *ctr-simps*
substituting VS_1.*ab-group-add-ow-axioms*
and VS_1.*vector-space-ow-axioms*
and VS_2.*ab-group-add-ow-axioms*
and VS_2.*vector-space-ow-axioms*
and $implicit_{VS}$.*vector-space-pair-ow-axioms*
and VS_1.$implicit_M$.*module-ow-axioms*
and VS_2.$implicit_M$.*module-ow-axioms*
eliminating ‹*?a* ∈ $U_{M\text{-}1}$› **and** ‹*?B* ⊆ $U_{M\text{-}1}$› **and** ‹*?a* ∈ $U_{M\text{-}2}$› **and** ‹*?B* ⊆ $U_{M\text{-}2}$›
through *auto*
applying
[
OF
VS_1.$implicit_M$.*carrier-ne* VS_2.$implicit_M$.*carrier-ne*
VS_1.$implicit_M$.*minus-closed'* VS_1.$implicit_M$.*uminus-closed'*
VS_2.$implicit_{VS}$.*minus-closed'* VS_2.$implicit_{VS}$.*uminus-closed'*,
unfolded tts-implicit
]
begin

tts-lemma *linear-add*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $b1 \in U_{M\text{-}1}$
and $b2 \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $f\ (b1 + b2) = f\ b1 + f\ b2$
is *vector-space-pair.linear-add*⟨*proof*⟩

tts-lemma *linear-scale*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$

and $b \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $f\ (r *s_1 b) = r *s_2 f\ b$
is *vector-space-pair.linear-scale*⟨*proof*⟩

tts-lemma *linear-neg*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $x \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $f\ (-x) = -f\ x$
is *vector-space-pair.linear-neg*⟨*proof*⟩

tts-lemma *linear-diff*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $x \in U_{M\text{-}1}$
and $y \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $f\ (x - y) = f\ x - f\ y$
is *vector-space-pair.linear-diff*⟨*proof*⟩

tts-lemma *linear-sum*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and *range* $g \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $f\ (sum\ g\ S) = (\sum a{\in}S.\ f\ (g\ a))$
is *vector-space-pair.linear-sum*⟨*proof*⟩

tts-lemma *linear-inj-on-iff-eq*-0:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $s \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $VS_1.subspace\ s$
shows *inj-on* $f\ s = (\forall x{\in}s.\ f\ x = 0 \longrightarrow x = 0)$
is *vector-space-pair.linear-inj-on-iff-eq*-0⟨*proof*⟩

tts-lemma *linear-inj-iff-eq*-0:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows *inj-on* $f\ U_{M\text{-}1} = (\forall x{\in}U_{M\text{-}1}.\ f\ x = 0 \longrightarrow x = 0)$
is *vector-space-pair.linear-inj-iff-eq*-0⟨*proof*⟩

tts-lemma *linear-subspace-image*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $VS_1.subspace\ S$
shows $VS_2.subspace\ (f\ `\ S)$
is *vector-space-pair.linear-subspace-image*⟨*proof*⟩

tts-lemma *linear-subspace-kernel*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $VS_1.subspace\ \{x \in U_{M\text{-}1}.\ f\ x = 0\}$
is *vector-space-pair.linear-subspace-kernel*⟨*proof*⟩

tts-lemma *linear-span-image*:
assumes $\forall x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$

and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
shows $VS_2.span\ (f\ `\ S) = f\ `\ VS_1.span\ S$
is *vector-space-pair.linear-span-image*⟨*proof*⟩

tts-lemma *linear-dependent-inj-imageD*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $s \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $VS_2.dependent\ (f\ `\ s)$
and *inj-on* $f\ (VS_1.span\ s)$
shows $VS_1.dependent\ s$
is *vector-space-pair.linear-dependent-inj-imageD*⟨*proof*⟩

tts-lemma *linear-eq*-0-*on-span*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $b \subseteq U_{M\text{-}1}$
and $x \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $\bigwedge x.\ [\![x \in U_{M\text{-}1};\ x \in b]\!] \Longrightarrow f\ x = 0$
and $x \in VS_1.span\ b$
shows $f\ x = 0$
is *vector-space-pair.linear-eq*-0-*on-span*⟨*proof*⟩

tts-lemma *linear-independent-injective-image*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $s \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $\neg\ VS_1.dependent\ s$
and *inj-on* $f\ (VS_1.span\ s)$
shows $\neg\ VS_2.dependent\ (f\ `\ s)$
is *vector-space-pair.linear-independent-injective-image*⟨*proof*⟩

tts-lemma *linear-inj-on-span-independent-image*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $\neg\ VS_2.dependent\ (f\ `\ B)$
and *inj-on* $f\ B$
shows *inj-on* $f\ (VS_1.span\ B)$
is *vector-space-pair.linear-inj-on-span-independent-image*⟨*proof*⟩

tts-lemma *linear-inj-on-span-iff-independent-image*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $\neg\ VS_2.dependent\ (f\ `\ B)$
shows *inj-on* $f\ (VS_1.span\ B) =$ *inj-on* $f\ B$
is *vector-space-pair.linear-inj-on-span-iff-independent-image*⟨*proof*⟩

tts-lemma *linear-subspace-linear-preimage*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and $VS_2.subspace\ S$
shows $VS_1.subspace\ \{x \in U_{M\text{-}1}.\ f\ x \in S\}$
is *vector-space-pair.linear-subspace-linear-preimage*⟨*proof*⟩

tts-lemma *linear-spans-image*:

assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $V \subseteq U_{M\text{-}1}$
and $B \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $V \subseteq VS_1.span\ B$
shows $f\ `\ V \subseteq VS_2.span\ (f\ `\ B)$
is *vector-space-pair.linear-spans-image*⟨*proof*⟩

tts-lemma *linear-spanning-surjective-image*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $U_{M\text{-}1} \subseteq VS_1.span\ S$
and $f\ `\ U_{M\text{-}1} = U_{M\text{-}2}$
shows $U_{M\text{-}2} \subseteq VS_2.span\ (f\ `\ S)$
is *vector-space-pair.linear-spanning-surjective-image*⟨*proof*⟩

tts-lemma *linear-eq-on-span*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and $x \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ g$
and $\bigwedge x.\ [\![x \in U_{M\text{-}1};\ x \in B]\!] \Longrightarrow f\ x = g\ x$
and $x \in VS_1.span\ B$
shows $f\ x = g\ x$
is *vector-space-pair.linear-eq-on-span*⟨*proof*⟩

tts-lemma *linear-compose-scale-right*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
shows *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ (\lambda x.\ c\ *s_2\ f\ x)$
is *vector-space-pair.linear-compose-scale-right*⟨*proof*⟩

tts-lemma *linear-compose-add*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ g$
shows *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ (\lambda x.\ f\ x + g\ x)$
is *vector-space-pair.linear-compose-add*⟨*proof*⟩

tts-lemma *linear-zero*:
shows *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ (\lambda x.\ 0)$
is *vector-space-pair.linear-zero*⟨*proof*⟩

tts-lemma *linear-compose-sub*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ g$
shows *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ (\lambda x.\ f\ x - g\ x)$
is *vector-space-pair.linear-compose-sub*⟨*proof*⟩

tts-lemma *linear-compose-neg*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$

shows *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ -\ f\ x)$
is *vector-space-pair.linear-compose-neg*⟨*proof*⟩

tts-lemma *linear-compose-scale*:
assumes $c \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ *UNIV* $(*s_1)$ $(*)$ f
shows *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ f\ x\ *s_2\ c)$
is *vector-space-pair.linear-compose-scale*⟨*proof*⟩

tts-lemma *linear-indep-image-lemma*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and $x \in U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and *finite* B
and $\neg$ VS_2.*dependent* $(f\ `\ B)$
and *inj-on* f B
and $x \in VS_1$.*span* B
and $f\ x = 0$
shows $x = 0$
is *vector-space-pair.linear-indep-image-lemma*⟨*proof*⟩

tts-lemma *linear-eq-on*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\forall\, x{\in}U_{M\text{-}1}.\ g\ x \in U_{M\text{-}2}$
and $x \in U_{M\text{-}1}$
and $B \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ g
and $x \in VS_1$.*span* B
and $\bigwedge b.\ [\![b \in U_{M\text{-}1};\ b \in B]\!] \Longrightarrow f\ b = g\ b$
shows $f\ x = g\ x$
is *vector-space-pair.linear-eq-on*⟨*proof*⟩

tts-lemma *linear-compose-sum*:
assumes $\forall\, x.\ \forall\, y{\in}U_{M\text{-}1}.\ f\ x\ y \in U_{M\text{-}2}$
and $\forall\, x{\in}S.$ *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(f\ x)$
shows *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $(\lambda x.\ \sum a{\in}S.\ f\ a\ x)$
is *vector-space-pair.linear-compose-sum*⟨*proof*⟩

tts-lemma *linear-independent-extend-subspace*:
assumes $B \subseteq U_{M\text{-}1}$
and $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\neg$ VS_1.*dependent* B
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}1}.\ x\ a \in U_{M\text{-}2}) \wedge$
linear-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $x \wedge$
$(\forall\, a{\in}B.\ x\ a = f\ a) \wedge$
$x\ `\ U_{M\text{-}1} = VS_2$.*span* $(f\ `\ B)$
given *vector-space-pair.linear-independent-extend-subspace* ⟨*proof*⟩

tts-lemma *linear-independent-extend*:
assumes $B \subseteq U_{M\text{-}1}$
and $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $\neg$ VS_1.*dependent* B
shows
$\exists\, x.$

$(\forall\, a{\in}U_{M\text{-}1}.\ x\ a\ \in\ U_{M\text{-}2})\ \wedge$
$(\forall\, a{\in}B.\ x\ a = f\ a)\ \wedge$
linear-on $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ x$
given *vector-space-pair.linear-independent-extend* ⟨*proof*⟩

tts-lemma *linear-exists-left-inverse-on*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x\ \in\ U_{M\text{-}2}$
and $V \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $VS_1.subspace\ V$
and *inj-on* $f\ V$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a\ \in\ U_{M\text{-}1})\ \wedge$
$x\ `\ U_{M\text{-}2} \subseteq V\ \wedge$
$(\forall\, a{\in}V.\ x\ (f\ a) = a)\ \wedge$
linear-on $U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-exists-left-inverse-on* ⟨*proof*⟩

tts-lemma *linear-exists-right-inverse-on*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x\ \in\ U_{M\text{-}2}$
and $V \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $VS_1.subspace\ V$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a\ \in\ U_{M\text{-}1})\ \wedge$
$x\ `\ U_{M\text{-}2} \subseteq V\ \wedge$
$(\forall\, a{\in}f\ `\ V.\ f\ (x\ a) = a)\ \wedge$
linear-on $U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-exists-right-inverse-on* ⟨*proof*⟩

tts-lemma *linear-inj-on-left-inverse*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x\ \in\ U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and *inj-on* $f\ (VS_1.span\ S)$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a\ \in\ U_{M\text{-}1})\ \wedge$
$x\ `\ U_{M\text{-}2} \subseteq VS_1.span\ S\ \wedge$
$(\forall\, a{\in}VS_1.span\ S.\ x\ (f\ a) = a)\ \wedge$
linear-on $U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-inj-on-left-inverse* ⟨*proof*⟩

tts-lemma *linear-surj-right-inverse*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x\ \in\ U_{M\text{-}2}$
and $T \subseteq U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $VS_2.span\ T \subseteq f\ `\ VS_1.span\ S$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a\ \in\ U_{M\text{-}1})\ \wedge$
$x\ `\ U_{M\text{-}2} \subseteq VS_1.span\ S\ \wedge$
$(\forall\, a{\in}VS_2.span\ T.\ f\ (x\ a) = a)\ \wedge$
linear-on $U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-surj-right-inverse* ⟨*proof*⟩

tts-lemma *finite-basis-to-basis-subspace-isomorphism*:
assumes $S \subseteq U_{M\text{-}1}$
and $T \subseteq U_{M\text{-}2}$
and $VS_1.subspace\ S$
and $VS_2.subspace\ T$
and $VS_1.dim\ S = VS_2.dim\ T$
and *finite* B
and $B \subseteq S$
and $\neg\ VS_1.dependent\ B$
and $S \subseteq VS_1.span\ B$
and $card\ B = VS_1.dim\ S$
and *finite* C
and $C \subseteq T$
and $\neg\ VS_2.dependent\ C$
and $T \subseteq VS_2.span\ C$
and $card\ C = VS_2.dim\ T$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}1}.\ x\ a \in U_{M\text{-}2}) \wedge$
$linear\text{-}on\ U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ x \wedge$
$x\ `\ B = C \wedge$
$inj\text{-}on\ x\ S \wedge x\ `\ S = T$
given *vector-space-pair.finite-basis-to-basis-subspace-isomorphism* ⟨*proof*⟩

tts-lemma *linear-subspace-vimage*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}2}$
and $linear\text{-}on\ U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $VS_2.subspace\ S$
shows $VS_1.subspace\ (f\ -`\ S \cap U_{M\text{-}1})$
is *vector-space-pair.linear-subspace-vimage*⟨*proof*⟩

tts-lemma *linear-injective-left-inverse*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $linear\text{-}on\ U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $inj\text{-}on\ f\ U_{M\text{-}1}$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a \in U_{M\text{-}1}) \wedge$
$(\forall\, a{\in}U_{M\text{-}1}.\ (x \circ f)\ a = id\ a) \wedge$
$linear\text{-}on\ U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-injective-left-inverse*[*unfolded fun-eq-iff*]
⟨*proof*⟩

tts-lemma *linear-surjective-right-inverse*:
assumes $\forall\, x{\in}U_{M\text{-}1}.\ f\ x \in U_{M\text{-}2}$
and $linear\text{-}on\ U_{M\text{-}1}\ U_{M\text{-}2}\ (*s_1)\ (*s_2)\ f$
and $f\ `\ U_{M\text{-}1} = U_{M\text{-}2}$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a \in U_{M\text{-}1}) \wedge$
$(\forall\, a{\in}U_{M\text{-}2}.\ (f \circ x)\ a = id\ a) \wedge$
$linear\text{-}on\ U_{M\text{-}2}\ U_{M\text{-}1}\ (*s_2)\ (*s_1)\ x$
given *vector-space-pair.linear-surjective-right-inverse*[*unfolded fun-eq-iff*]
⟨*proof*⟩

end

end

context *finite-dimensional-vector-space-pair-1-on*
begin

tts-context
tts: (*?'b* **to** ‹$U_{M\text{-}1}$::*'b set*›) **and** (*?'c* **to** ‹$U_{M\text{-}2}$::*'c set*›)
sbterms: (‹(+)::*?'b*::*ab-group-add*⇒*?'b*⇒*?'b*› **to** ‹(+)::*'b*⇒*'b*⇒*'b*›)
and (‹*?s*1.0::*?'a*::*field* ⇒*?'b*::*ab-group-add*⇒*?'b*› **to** ‹($*s_1$)::*'a*⇒*'b*⇒*'b*›)
and (‹0::*?'b*::*ab-group-add*› **to** ‹0::*'b*›)
and (‹(+)::*?'c*::*ab-group-add*⇒*?'c*⇒*?'c*› **to** ‹(+)::*'c*⇒*'c*⇒*'c*›)
and (‹*?s*2.0::*?'a*::*field* ⇒*?'c*::*ab-group-add*⇒*?'c*› **to** ‹($*s_2$)::*'a*⇒*'c*⇒*'c*›)
and (‹0::*?'c*::*ab-group-add*› **to** ‹0::*'c*›)
rewriting *ctr-simps*
substituting VS_1.*ab-group-add-ow-axioms*
and VS_1.*vector-space-ow-axioms*
and VS_2.*ab-group-add-ow-axioms*
and VS_2.*vector-space-ow-axioms*
and $implicit_{VS}$.*vector-space-pair-ow-axioms*
and $VS_1.implicit_M$.*module-ow-axioms*
and $VS_2.implicit_M$.*module-ow-axioms*
and $implicit_{VS}$.*finite-dimensional-vector-space-pair-1-ow-axioms*
applying
[
OF
$VS_1.implicit_M$.*carrier-ne* $VS_2.implicit_M$.*carrier-ne*
$VS_1.implicit_{VS}$.*minus-closed'* $VS_1.implicit_{VS}$.*uminus-closed'*
$VS_2.implicit_{VS}$.*minus-closed'* $VS_2.implicit_{VS}$.*uminus-closed'*
VS_1.*basis-subset*,
unfolded tts-implicit
]
begin

tts-lemma *lt-dim-image-eq*:
assumes ∀ *x*∈$U_{M\text{-}1}$. *f x* ∈ $U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ ($*s_1$) ($*s_2$) *f*
and *inj-on f* (*VS*$_1$.*span S*)
shows $VS_2.dim\ (f\ `\ S) = VS_1.dim\ S$
is *finite-dimensional-vector-space-pair-1.dim-image-eq*⟨*proof*⟩

tts-lemma *lt-dim-image-le*:
assumes ∀ *x*∈$U_{M\text{-}1}$. *f x* ∈ $U_{M\text{-}2}$
and $S \subseteq U_{M\text{-}1}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ ($*s_1$) ($*s_2$) *f*
shows $VS_2.dim\ (f\ `\ S) \leq VS_1.dim\ S$
is *finite-dimensional-vector-space-pair-1.dim-image-le*⟨*proof*⟩

end

end

context *finite-dimensional-vector-space-pair-on*
begin

tts-context
tts: (*?'b* **to** ‹$U_{M\text{-}1}$::*'b set*›) **and** (*?'c* **to** ‹$U_{M\text{-}2}$::*'c set*›)

sbterms: (‹(+)::?'b::*ab-group-add*⇒?'b⇒?'b› **to** ‹(+)::'b⇒'b⇒'b›)
and (‹?s1.0::?'a::*field* ⇒?'b::*ab-group-add*⇒?'b› **to** ‹(∗s_1)::'a⇒'b⇒'b›)
and (‹0::?'b::*ab-group-add*› **to** ‹0::'b›)
and (‹(+)::?'c::*ab-group-add*⇒?'c⇒?'c› **to** ‹(+)::'c⇒'c⇒'c›)
and (‹?s2.0::?'a::*field* ⇒?'c::*ab-group-add*⇒?'c› **to** ‹(∗s_2)::'a⇒'c⇒'c›)
and (‹0::?'c::*ab-group-add*› **to** ‹0::'c›)
rewriting *ctr-simps*
substituting VS_1.*ab-group-add-ow-axioms*
and VS_1.*vector-space-ow-axioms*
and VS_2.*ab-group-add-ow-axioms*
and VS_2.*vector-space-ow-axioms*
and *implicit*$_{VS}$.*vector-space-pair-ow-axioms*
and VS_1.*implicit*$_M$.*module-ow-axioms*
and VS_2.*implicit*$_M$.*module-ow-axioms*
and *implicit*$_{VS}$.*finite-dimensional-vector-space-pair-ow-axioms*
applying
[
OF
VS_1.*implicit*$_M$.*carrier-ne* VS_2.*implicit*$_M$.*carrier-ne*
VS_1.*implicit*$_{VS}$.*minus-closed'* VS_1.*implicit*$_{VS}$.*uminus-closed'*
VS_2.*implicit*$_{VS}$.*minus-closed'* VS_2.*implicit*$_{VS}$.*uminus-closed'*
VS_1.*basis-subset* VS_2.*basis-subset*,
unfolded tts-implicit
]
begin

tts-lemma *linear-surjective-imp-injective*:
assumes ∀ x∈$U_{M\text{-}1}$. $f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ (∗s_1) (∗s_2) f
and f ‘ $U_{M\text{-}1} = U_{M\text{-}2}$
and VS_2.*dim* $U_{M\text{-}2} = VS_1$.*dim* $U_{M\text{-}1}$
shows *inj-on* f $U_{M\text{-}1}$
is *finite-dimensional-vector-space-pair.linear-surjective-imp-injective*⟨*proof*⟩

tts-lemma *linear-injective-imp-surjective*:
assumes ∀ x∈$U_{M\text{-}1}$. $f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ (∗s_1) (∗s_2) f
and *inj-on* f $U_{M\text{-}1}$
and VS_2.*dim* $U_{M\text{-}2} = VS_1$.*dim* $U_{M\text{-}1}$
shows f ‘ $U_{M\text{-}1} = U_{M\text{-}2}$
is *finite-dimensional-vector-space-pair.linear-injective-imp-surjective*⟨*proof*⟩

tts-lemma *linear-injective-isomorphism*:
assumes ∀ x∈$U_{M\text{-}1}$. $f\ x \in U_{M\text{-}2}$
and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ (∗s_1) (∗s_2) f
and *inj-on* f $U_{M\text{-}1}$
and VS_2.*dim* $U_{M\text{-}2} = VS_1$.*dim* $U_{M\text{-}1}$
shows
∃ x.
(∀ a∈$U_{M\text{-}2}$. $x\ a \in U_{M\text{-}1}$) ∧
linear-on $U_{M\text{-}2}$ $U_{M\text{-}1}$ (∗s_2) (∗s_1) x ∧
(∀ a∈$U_{M\text{-}1}$. $x\ (f\ a) = a$) ∧
(∀ a∈$U_{M\text{-}2}$. $f\ (x\ a) = a$)
given *finite-dimensional-vector-space-pair.linear-injective-isomorphism*
⟨*proof*⟩

tts-lemma *linear-surjective-isomorphism*:
assumes ∀ x∈$U_{M\text{-}1}$. $f\ x \in U_{M\text{-}2}$

and *linear-on* $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ f
and f ‘ $U_{M\text{-}1} = U_{M\text{-}2}$
and $VS_2.dim\ U_{M\text{-}2} = VS_1.dim\ U_{M\text{-}1}$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}2}.\ x\ a \in U_{M\text{-}1}) \wedge$
linear-on $U_{M\text{-}2}$ $U_{M\text{-}1}$ $(*s_2)$ $(*s_1)$ $x \wedge$
$(\forall\, a{\in}U_{M\text{-}1}.\ x\ (f\ a) = a) \wedge$
$(\forall\, a{\in}U_{M\text{-}2}.\ f\ (x\ a) = a)$
given *finite-dimensional-vector-space-pair.linear-surjective-isomorphism*
⟨*proof*⟩

tts-lemma *basis-to-basis-subspace-isomorphism*:
assumes $S \subseteq U_{M\text{-}1}$
and $T \subseteq U_{M\text{-}2}$
and $B \subseteq U_{M\text{-}1}$
and $C \subseteq U_{M\text{-}2}$
and $VS_1.subspace\ S$
and $VS_2.subspace\ T$
and $VS_1.dim\ S = VS_2.dim\ T$
and $B \subseteq S$
and $\neg\ VS_1.dependent\ B$
and $S \subseteq VS_1.span\ B$
and $card\ B = VS_1.dim\ S$
and $C \subseteq T$
and $\neg\ VS_2.dependent\ C$
and $T \subseteq VS_2.span\ C$
and $card\ C = VS_2.dim\ T$
shows
$\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}1}.\ x\ a \in U_{M\text{-}2}) \wedge$
linear-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ $x \wedge$
x ‘ $B = C \wedge$
inj-on $x\ S \wedge$
x ‘ $S = T$
given *finite-dimensional-vector-space-pair.basis-to-basis-subspace-isomorphism*
⟨*proof*⟩

tts-lemma *subspace-isomorphism*:
assumes $S \subseteq U_{M\text{-}1}$
and $T \subseteq U_{M\text{-}2}$
and $VS_1.subspace\ S$
and $VS_2.subspace\ T$
and $VS_1.dim\ S = VS_2.dim\ T$
shows $\exists\, x.$
$(\forall\, a{\in}U_{M\text{-}1}.\ x\ a \in U_{M\text{-}2}) \wedge$
$(inj\text{-}on\ x\ S \wedge x$ ‘ $S = T) \wedge$
linear-on $U_{M\text{-}1}$ $U_{M\text{-}2}$ $(*s_1)$ $(*s_2)$ x
given *finite-dimensional-vector-space-pair.subspace-isomorphism* ⟨*proof*⟩

end

end

TTS Foundations

5.1 Extension of the theory *Set*

lemma *Ex*1-*transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows ((*A* ===> (=)) ===> (=)) (λ*P*. (∃!*x*∈(*Collect* (*Domainp A*)). *P x*)) *Ex*1
⟨*proof*⟩

5.2 Definite description operator

5.2.1 Definition and common properties

definition *The-on*
where *The-on U P* =
(*if* ∃!*x*. *x* ∈ *U* ∧ *P x* *then Some* (*THE x*. *x* ∈ *U* ∧ *P x*) *else None*)

syntax
-The-on :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *'a option*
(‹(*THE* - *on* -./ -)› [0, 0, 10] 10)
syntax-consts
-The-on ⇌ *The-on*
translations *THE x on U*. *P* ⇌ *CONST The-on U* (λ*x*. *P*)

print_translation ‹
```
[
  (
    const_syntax‹The_on›,
    fn ctxt => fn [Ut, Abs abs] =>
      let val (x, t) = Syntax_Trans.atomic_abs_tr' ctxt abs
      in Syntax.const syntax_const‹_The_on› $ x $ Ut $ t end
  )
]
```
›⟨*ML*⟩

lemma *The-on-UNIV-eq-The*:
assumes ∃!*x*. *P x*
obtains *x* **where** (*THE x on UNIV*. *P x*) = *Some x* **and** (*THE x*. *P x*) = *x*
⟨*proof*⟩

lemma *The-on-UNIV-None*:
assumes ¬(∃!*x*. *P x*)
shows (*THE x on UNIV*. *P x*) = *None*
⟨*proof*⟩

lemma *The-on-eq-The*:
assumes ∃!*x*. *x* ∈ *U* ∧ *P x*
obtains *x* **where** (*THE x on U*. *P x*) = *Some x* **and** (*THE x*. *x* ∈ *U* ∧ *P x*) = *x*
⟨*proof*⟩

lemma *The-on-None*:
assumes ¬(∃!*x*. *x* ∈ *U* ∧ *P x*)
shows (*THE x on U*. *P x*) = *None*
⟨*proof*⟩

lemma *The-on-Some-equality*[*intro*]:
assumes *a* ∈ *U* **and** *P a* **and** ⋀*x*. *x* ∈ *U* ⟹ *P x* ⟹ *x* = *a*
shows (*THE x on U*. *P x*) = *Some a*
⟨*proof*⟩

lemma *The-on-equality*[*intro*]:
assumes *a* ∈ *U* **and** *P a* **and** ⋀*x*. *x* ∈ *U* ⟹ *P x* ⟹ *x* = *a*
shows *the* (*THE x on U*. *P x*) = *a*
⟨*proof*⟩

lemma *The-on-SomeI*:
assumes *a* ∈ *U* **and** *P a* **and** ⋀*x*. *x* ∈ *U* ⟹ *P x* ⟹ *x* = *a*

obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $P\ x$
⟨*proof*⟩

lemma *The-onI*:
assumes $a \in U$ **and** $P\ a$ **and** $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow x = a$
shows $P\ (the\ (THE\ x\ on\ U.\ P\ x))$
⟨*proof*⟩

lemma *The-on-SomeI′*:
assumes $\exists!x.\ x \in U \wedge P\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $P\ x$
⟨*proof*⟩

lemma *The-onI′*:
assumes $\exists!x.\ x \in U \wedge P\ x$
shows $P\ (the\ (THE\ x\ on\ U.\ P\ x))$
⟨*proof*⟩

lemma *The-on-SomeI*2:
assumes $a \in U$
and $P\ a$
and $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow x = a$
and $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow Q\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $Q\ x$
⟨*proof*⟩

lemma *The-on-I*2:
assumes $a \in U$
and $P\ a$
and $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow x = a$
and $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow Q\ x$
shows $Q\ (the\ (THE\ x\ on\ U.\ P\ x))$
⟨*proof*⟩

lemma *The-on-Some*1*I*2:
assumes $\exists!x.\ x \in U \wedge P\ x$ **and** $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow Q\ x$
obtains x **where** $(THE\ x\ on\ U.\ P\ x) = Some\ x$ **and** $Q\ x$
⟨*proof*⟩

lemma *The-on*1*I*2:
assumes $\exists!x.\ x \in U \wedge P\ x$ **and** $\bigwedge x.\ x \in U \Longrightarrow P\ x \Longrightarrow Q\ x$
shows $Q\ (the\ (THE\ x\ on\ U.\ P\ x))$
⟨*proof*⟩

lemma *The-on*1*-equality* [*elim?*]:
assumes $\exists!x.\ P\ x$ **and** $a \in U$ **and** $P\ a$
shows $(THE\ x\ on\ U.\ P\ x) = Some\ a$
⟨*proof*⟩

lemma *the-sym-eq-trivial*:
assumes $x \in U$
shows $(THE\ y\ on\ U.\ x = y) = Some\ x$
⟨*proof*⟩

5.2.2 Transfer rules

lemma *The-on-transfer*[*transfer-rule*]:
includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique A right-total A*
shows (*rel-set A* ===> (*A* ===> (=)) ===> *rel-option A*) *The-on The-on*
⟨*proof*⟩

5.3 Auxiliary

5.3.1 Methods

method *ow-locale-transfer* **uses** *locale-defs* =
(*unfold locale-defs*,
((*simp only*: *all-simps*(6) *all-comm*, *fold Ball-def*)
| (*fold Ball-def*)
| *tactic*‹*all-tac*›),
transfer-prover-start,
transfer-step+,
rule refl)

5.4 Abstract orders on types

5.4.1 Background

The results presented in this section were ported (with amendments and additions) from the theories *Orderings* and *Set-Interval* in the main library of Isabelle/HOL.

5.4.2 Order operations

Abstract order operations.

locale *ord* =
  fixes *le ls* :: [′*a*, ′*a*] ⇒ *bool*

locale *ord-syntax* = *ord le ls* **for** *le ls* :: [′*a*, ′*a*] ⇒ *bool*
begin

notation
  le (‹′(≤$_a$′)›) **and**
  le (**infix** ‹≤$_a$› 50) **and**
  ls (‹′(<$_a$′)›) **and**
  ls (**infix** ‹<$_a$› 50)

abbreviation (*input*) *ge* (**infix** ‹≥$_a$› 50)
  where *x* ≥$_a$ *y* ≡ *y* ≤$_a$ *x*
abbreviation (*input*) *gt* (**infix** ‹>$_a$› 50)
  where *x* >$_a$ *y* ≡ *y* <$_a$ *x*

notation
  ge (‹′(≥$_a$′)›) **and**
  ge (**infix** ‹≥$_a$› 50) **and**
  gt (‹′(>$_a$′)›) **and**
  gt (**infix** ‹>$_a$› 50)

end

locale *ord-dual* = *ord le ls* **for** *le ls* :: [′*a*, ′*a*] ⇒ *bool*
begin

interpretation *ord-syntax* ⟨*proof*⟩
sublocale *dual*: *ord ge gt* ⟨*proof*⟩

end

Pairs.

locale *ord-pair* = *ord*$_a$: *ord le*$_a$ *ls*$_a$ + *ord*$_b$: *ord le*$_b$ *ls*$_b$
  for *le*$_a$ *ls*$_a$ :: [′*a*, ′*a*] ⇒ *bool* **and** *le*$_b$ *ls*$_b$ :: [′*b*, ′*b*] ⇒ *bool*
begin

sublocale *rev*: *ord-pair le*$_b$ *ls*$_b$ *le*$_a$ *ls*$_a$ ⟨*proof*⟩

end

locale *ord-pair-syntax* = *ord-pair le*$_a$ *ls*$_a$ *le*$_b$ *ls*$_b$
  for *le*$_a$ *ls*$_a$ :: [′*a*, ′*a*] ⇒ *bool* **and** *le*$_b$ *ls*$_b$ :: [′*b*, ′*b*] ⇒ *bool*
begin

sublocale *ord*$_a$: *ord-syntax le*$_a$ *ls*$_a$ + *ord*$_b$: *ord-syntax le*$_b$ *ls*$_b$ ⟨*proof*⟩

notation le_a ($\langle$'($\leq_a$')$\rangle$)
  and le_a (**infix** $\langle\leq_a\rangle$ 50)
  and ls_a ($\langle$'($<_a$')$\rangle$)
  and ls_a (**infix** $\langle<_a\rangle$ 50)
  and le_b ($\langle$'($\leq_b$')$\rangle$)
  and le_b (**infix** $\langle\leq_b\rangle$ 50)
  and ls_b ($\langle$'($<_b$')$\rangle$)
  and ls_b (**infix** $\langle<_b\rangle$ 50)

notation $ord_a.ge$ ($\langle$'($\geq_a$')$\rangle$)
  and $ord_a.ge$ (**infix** $\langle\geq_a\rangle$ 50)
  and $ord_a.gt$ ($\langle$'($>_a$')$\rangle$)
  and $ord_a.gt$ (**infix** $\langle>_a\rangle$ 50)
  and $ord_b.ge$ ($\langle$'($\geq_b$')$\rangle$)
  and $ord_b.ge$ (**infix** $\langle\geq_b\rangle$ 50)
  and $ord_b.gt$ ($\langle$'($>_b$')$\rangle$)
  and $ord_b.gt$ (**infix** $\langle>_b\rangle$ 50)

end

locale *ord-pair-dual* = *ord-pair* le_a ls_a le_b ls_b
  for le_a ls_a :: ['a, 'a] $\Rightarrow$ *bool* **and** le_b ls_b :: ['b, 'b] $\Rightarrow$ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-dual*: *ord-pair* $\langle(\leq_a)\rangle$ $\langle(<_a)\rangle$ $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ ⟨*proof*⟩
sublocale *dual-ord*: *ord-pair* $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ $\langle(\leq_b)\rangle$ $\langle(<_b)\rangle$ ⟨*proof*⟩
sublocale *dual-dual*: *ord-pair* $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ ⟨*proof*⟩

end

5.4.3 Preorders

Definitions

Abstract preorders.

locale *preorder* = *ord le ls* **for** *le ls* :: ['a, 'a] $\Rightarrow$ *bool* +
  assumes *less-le-not-le*: *ls x y* $\longleftrightarrow$ *le x y* $\wedge$ $\neg$ (*le y x*)
    and *order-refl*[*iff*]: *le x x*
    and *order-trans*: *le x y* $\Longrightarrow$ *le y z* $\Longrightarrow$ *le x z*

locale *preorder-dual* = *preorder le ls* **for** *le ls* :: ['a, 'a] $\Rightarrow$ *bool*
begin

interpretation *ord-syntax* ⟨*proof*⟩

sublocale *ord-dual* ⟨*proof*⟩

sublocale *dual*: *preorder ge gt*
  ⟨*proof*⟩

end

Pairs.

locale *ord-preorder* = *ord-pair* le_a ls_a le_b ls_b + ord_b: *preorder* le_b ls_b
  for le_a ls_a :: ['a, 'a] $\Rightarrow$ *bool* **and** le_b ls_b :: ['b, 'b] $\Rightarrow$ *bool*

locale *ord-preorder-dual* = *ord-preorder* le_a ls_a le_b ls_b
  for le_a ls_a :: $['a, 'a] \Rightarrow$ *bool* **and** le_b ls_b :: $['b, 'b] \Rightarrow$ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-pair-dual* ⟨*proof*⟩
sublocale *ord-dual*: *ord-preorder* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$›
  ⟨*proof*⟩
sublocale *dual-ord*: *ord-preorder* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *ord-preorder* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$›
  ⟨*proof*⟩

end

locale *preorder-pair* = *ord-preorder* le_a ls_a le_b ls_b + ord_a: *preorder* le_a ls_a
  for le_a ls_a :: $['a, 'a] \Rightarrow$ *bool* **and** le_b ls_b :: $['b, 'b] \Rightarrow$ *bool*
begin

sublocale *rev*: *preorder-pair* le_b ls_b le_a ls_a ⟨*proof*⟩

end

locale *preorder-pair-dual* = *preorder-pair* le_a ls_a le_b ls_b
  for le_a ls_a :: $['a, 'a] \Rightarrow$ *bool* **and** le_b ls_b :: $['b, 'b] \Rightarrow$ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-preorder-dual* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-pair* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *preorder-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *preorder-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

Results

context *preorder*
begin

interpretation *ord-syntax* ⟨*proof*⟩

Reflexivity.

lemma *eq-refl*:
  assumes $x = y$
  shows $x \leq_a y$
  ⟨*proof*⟩

lemma *less-irrefl*[*iff*]: $\neg\ x <_a x$ ⟨*proof*⟩

lemma *less-imp-le*:
  assumes $x <_a y$
  shows $x \leq_a y$
  ⟨*proof*⟩

lemma *strict-implies-not-eq*:
assumes $a <_a b$
shows $a \neq b$
⟨*proof*⟩

Asymmetry.

lemma *less-not-sym*:
assumes $x <_a y$
shows $\neg\ (y <_a x)$
⟨*proof*⟩

lemma *asym*:
assumes $a <_a b$ **and** $b <_a a$
shows *False*
⟨*proof*⟩

lemma *less-asym*:
assumes $x <_a y$ **and** $(\neg\ P \Longrightarrow y <_a x)$
shows P
⟨*proof*⟩

Transitivity.

lemma *less-trans*:
assumes $x <_a y$ **and** $y <_a z$
shows $x <_a z$
⟨*proof*⟩

lemma *le-less-trans*:
assumes $x \leq_a y$ **and** $y <_a z$
shows $x <_a z$
⟨*proof*⟩

lemma *less-le-trans*:
assumes $x <_a y$ **and** $y \leq_a z$
shows $x <_a z$
⟨*proof*⟩

lemma *less-imp-not-less*:
assumes $x <_a y$
shows $(\neg\ y <_a x) \longleftrightarrow$ *True*
⟨*proof*⟩

lemma *less-imp-triv*:
assumes $x <_a y$
shows $(y <_a x \longrightarrow P) \longleftrightarrow$ *True*
⟨*proof*⟩

lemma *less-asym'*:
assumes $a <_a b$ **and** $b <_a a$
shows P
⟨*proof*⟩

end

5.4.4 Partial orders

Definitions

Abstract partial orders.

locale *order* = *preorder le ls* **for** *le ls* :: $['a, 'a] \Rightarrow bool$ +
  assumes *antisym*: $le\ x\ y \Longrightarrow le\ y\ x \Longrightarrow x = y$

locale *order-dual* = *order le ls* **for** *le ls* :: $['a, 'a] \Rightarrow bool$
begin

interpretation *ord-syntax* ⟨*proof*⟩

sublocale *preorder-dual* ⟨*proof*⟩

sublocale *dual*: *order ge gt*
  ⟨*proof*⟩

end

Pairs.

locale *ord-order* = *ord-preorder* $le_a\ ls_a\ le_b\ ls_b$ + ord_b: *order* $le_b\ ls_b$
  for $le_a\ ls_a$ :: $'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b$ :: $'b \Rightarrow 'b \Rightarrow bool$

locale *ord-order-dual* = *ord-order* $le_a\ ls_a\ le_b\ ls_b$
  for $le_a\ ls_a$ :: $'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b$ :: $'b \Rightarrow 'b \Rightarrow bool$
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-preorder-dual* ⟨*proof*⟩
sublocale *ord-dual*: *ord-order* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$›
  ⟨*proof*⟩
sublocale *dual-ord*: *ord-order* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *ord-order* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$›
  ⟨*proof*⟩

end

locale *preorder-order* = *ord-order* $le_a\ ls_a\ le_b\ ls_b$ + ord_a: *preorder* $le_a\ ls_a$
  for $le_a\ ls_a$ :: $['a, 'a] \Rightarrow bool$ **and** $le_b\ ls_b$ :: $['b, 'b] \Rightarrow bool$
begin

sublocale *preorder-pair* ⟨*proof*⟩

end

locale *preorder-order-dual* = *preorder-order* $le_a\ ls_a\ le_b\ ls_b$
  for $le_a\ ls_a$ :: $['a, 'a] \Rightarrow bool$ **and** $le_b\ ls_b$ :: $['b, 'b] \Rightarrow bool$
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-order-dual* ⟨*proof*⟩
sublocale *preorder-pair-dual* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-order* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *preorder-order* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$› ⟨*proof*⟩

sublocale *dual-dual*: *preorder-order* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

locale *order-pair* = *preorder-order* le_a ls_a le_b ls_b + ord_a: *order* le_a ls_a
  for le_a ls_a :: $['a, 'a] \Rightarrow bool$ **and** le_b ls_b :: $['b, 'b] \Rightarrow bool$
begin

sublocale *rev*: *order-pair* le_b ls_b le_a ls_a ⟨*proof*⟩

end

locale *order-pair-dual* = *order-pair* le_a ls_a le_b ls_b
  for le_a ls_a :: $['a, 'a] \Rightarrow bool$ **and** le_b ls_b :: $['b, 'b] \Rightarrow bool$
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *preorder-order-dual* ⟨*proof*⟩
sublocale *ord-dual*: *order-pair* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *order-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *order-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

Results

context *order*
begin

interpretation *ord-syntax* ⟨*proof*⟩

Reflexivity.

lemma *less-le*: $x <_a y \longleftrightarrow x \leq_a y \wedge x \neq y$
  ⟨*proof*⟩

lemma *le-less*: $x \leq_a y \longleftrightarrow x <_a y \vee x = y$ ⟨*proof*⟩

lemma *le-imp-less-or-eq*:
  assumes $x \leq_a y$
  shows $x <_a y \vee x = y$
  ⟨*proof*⟩

lemma *less-imp-not-eq*:
  assumes $x <_a y$
  shows $(x = y) \longleftrightarrow False$
  ⟨*proof*⟩

lemma *less-imp-not-eq2*:
  assumes $x <_a y$
  shows $(y = x) \longleftrightarrow False$
  ⟨*proof*⟩

Transitivity.

lemma *neq-le-trans*:
  assumes $a \neq b$ **and** $a \leq_a b$
  shows $a <_a b$

⟨*proof*⟩

lemma *le-neq-trans*:
assumes $a \leq_a b$ **and** $a \neq b$
shows $a <_a b$
⟨*proof*⟩

Asymmetry.

lemma *eq-iff*: $x = y \longleftrightarrow x \leq_a y \wedge y \leq_a x$ ⟨*proof*⟩

lemma *antisym-conv*:
assumes $y \leq_a x$
shows $x \leq_a y \longleftrightarrow x = y$
⟨*proof*⟩

Other results.

lemma *antisym-conv*1:
assumes $\neg\ x <_a y$
shows $x \leq_a y \longleftrightarrow x = y$
⟨*proof*⟩

lemma *antisym-conv*2:
assumes $x \leq_a y$
shows $\neg\ x <_a y \longleftrightarrow x = y$
⟨*proof*⟩

lemma *leD*:
assumes $y \leq_a x$
shows $\neg\ x <_a y$
⟨*proof*⟩

end

5.4.5 Dense orders

Abstract dense orders.

locale *dense-order* = *order le ls* **for** *le ls* :: $['a, 'a] \Rightarrow bool$ +
assumes *dense*: $ls\ x\ y \Longrightarrow (\exists z.\ ls\ x\ z \wedge ls\ z\ y)$

locale *dense-order-dual* = *dense-order le ls* **for** *le ls* :: $['a, 'a] \Rightarrow bool$
begin

interpretation *ord-syntax* ⟨*proof*⟩

sublocale *order-dual* ⟨*proof*⟩

sublocale *dual*: *dense-order ge gt*
⟨*proof*⟩

end

Pairs.

locale *ord-dense-order* = *ord-order* $le_a\ ls_a\ le_b\ ls_b$ + ord_b: *dense-order* $le_b\ ls_b$
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

locale *ord-dense-order-dual* = *ord-dense-order* $le_a\ ls_a\ le_b\ ls_b$
for $le_a\ ls_a :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $le_b\ ls_b :: 'b \Rightarrow 'b \Rightarrow bool$

begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-order-dual* ⟨*proof*⟩
sublocale *ord-dual*: *ord-dense-order* ‹(≤$_a$)› ‹(<$_a$)› ‹(≥$_b$)› ‹(>$_b$)›
⟨*proof*⟩
sublocale *dual-ord*: *ord-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≤$_b$)› ‹(<$_b$)›
⟨*proof*⟩
sublocale *dual-dual*: *ord-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≥$_b$)› ‹(>$_b$)›
⟨*proof*⟩

end

locale *preorder-dense-order* =
ord-dense-order le_a ls_a le_b ls_b + ord_a: *preorder* le_a ls_a
for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow bool$ **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow bool$
begin

sublocale *preorder-order* ⟨*proof*⟩

end

locale *preorder-dense-order-dual* = *preorder-dense-order* le_a ls_a le_b ls_b
for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow bool$ **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow bool$
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *ord-dense-order-dual* ⟨*proof*⟩
sublocale *preorder-order-dual* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-dense-order* ‹(≤$_a$)› ‹(<$_a$)› ‹(≥$_b$)› ‹(>$_b$)› ⟨*proof*⟩
sublocale *dual-ord*: *preorder-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≤$_b$)› ‹(<$_b$)› ⟨*proof*⟩
sublocale *dual-dual*: *preorder-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≥$_b$)› ‹(>$_b$)› ⟨*proof*⟩

end

locale *order-dense-order* =
preorder-dense-order le_a ls_a le_b ls_b + ord_a: *order* le_a ls_a
for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow bool$ **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow bool$
begin

sublocale *order-pair* ⟨*proof*⟩

end

locale *order-dense-order-dual* = *order-dense-order* le_a ls_a le_b ls_b
for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow bool$ **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow bool$
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *preorder-dense-order-dual* ⟨*proof*⟩
sublocale *order-pair-dual* ⟨*proof*⟩
sublocale *ord-dual*: *order-dense-order* ‹(≤$_a$)› ‹(<$_a$)› ‹(≥$_b$)› ‹(>$_b$)› ⟨*proof*⟩
sublocale *dual-ord*: *order-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≤$_b$)› ‹(<$_b$)› ⟨*proof*⟩
sublocale *dual-dual*: *order-dense-order* ‹(≥$_a$)› ‹(>$_a$)› ‹(≥$_b$)› ‹(>$_b$)› ⟨*proof*⟩

end

locale *dense-order-pair* =
  order-dense-order le_a ls_a le_b ls_b + ord_a: *dense-order* le_a ls_a
  for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow$ *bool*

locale *dense-order-pair-dual* = *dense-order-pair* le_a ls_a le_b ls_b
  for le_a ls_a :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** le_b ls_b :: $'b \Rightarrow 'b \Rightarrow$ *bool*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

sublocale *order-dense-order-dual* ⟨*proof*⟩
sublocale *ord-dual*: *dense-order-pair* ‹$(\leq_a)$› ‹$(<_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *dense-order-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *dense-order-pair* ‹$(\geq_a)$› ‹$(>_a)$› ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

5.4.6 (Unique) top and bottom elements

Abstract extremum.

locale *extremum* =
  fixes *extremum* :: $'a$

locale *ord-extremum* = *ord le ls* + *extremum extremum*
  for *le ls* :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and** *extremum* :: $'a$

Concrete syntax.

locale *bot* = *extremum bot* **for** *bot* :: $'a$
begin

notation *bot* (‹$\bot$›)

end

locale *top* = *extremum top* **for** *top* :: $'a$
begin

notation *top* (‹$\top$›)

end

5.4.7 (Unique) top and bottom elements for partial orders

Definitions

Abstract partial order with extremum.

locale *order-extremum* = *ord-extremum le ls extremum* + *order le ls*
  for *le ls* :: $'a \Rightarrow 'a \Rightarrow$ *bool*
  and *extremum* :: $'a$ +
  assumes *extremum*[*simp*]: *le a extremum*

Concrete syntax.

locale *order-bot* =
  order-dual le ls +

dual: *order-extremum* ‹λx y. *le y x*› ‹λx y. *ls y x*› *bot* +
bot bot
for *le ls* :: *'a* ⇒ *'a* ⇒ *bool* **and** *bot* :: *'a*

locale *order-top* = *order-dual le ls* + *order-extremum le ls top* + *top top*
for *le ls* :: *'a* ⇒ *'a* ⇒ *bool* **and** *top* :: *'a*

Results

context *order-extremum*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *extremum-uniqueI*:
assumes *extremum* ≤$_a$ *a*
shows *a* = *extremum*
⟨*proof*⟩

lemma *extremum-unique*: *extremum* ≤$_a$ *a* ⟷ *a* = *extremum*
⟨*proof*⟩

lemma *extremum-strict*[*simp*]: ¬ (*extremum* <$_a$ *a*)
⟨*proof*⟩

lemma *not-eq-extremum*: *a* ≠ *extremum* ⟷ *a* <$_a$ *extremum*
⟨*proof*⟩

end

5.4.8 Partial orders without top or bottom elements

Abstract partial orders without top or bottom elements.

locale *no-extremum* = *order le ls* **for** *le ls* :: *'a* ⇒ *'a* ⇒ *bool* +
assumes *gt-ex*: ∃ *y*. *ls x y*

Concrete syntax.

locale *no-top* = *order-dual le ls* + *no-extremum le ls*
for *le ls* :: *'a* ⇒ *'a* ⇒ *bool*

locale *no-bot* =
order-dual le ls +
dual: *no-extremum* ‹λx y. *le y x*› ‹λx y. *ls y x*›
for *le ls* :: *'a* ⇒ *'a* ⇒ *bool*

5.4.9 Least and greatest operators

definition *Least* :: [*'a set*, [*'a*, *'a*] ⇒ *bool*, *'a* ⇒ *bool*] ⇒ *'a option*
(‹(*on - with -* : «*Least*» -)› [1000, 1000, 1000] 10)
where
on U with op : «*Least*» *P* ≡ (*THE x on U*. *P x* ∧ (∀ *y*∈*U*. *P y* ⟶ *op x y*))

ctr relativization
synthesis *ctr-simps*
assumes [*transfer-domain-rule*, *transfer-rule*]: *Domainp A* = (λ*x*. *x* ∈ *U*)
and [*transfer-rule*]: *bi-unique A right-total A*
trp (*?'a A*)
in *Least-def*

context *ord-syntax*
begin

abbreviation *Least* **where** *Least* ≡ *Type-Simple-Orders.Least UNIV* ($\leq_a$)
abbreviation *Greatest* **where** *Greatest* ≡ *Type-Simple-Orders.Least UNIV* ($\geq_a$)

lemmas *Least-def* = *Least-def*[*of UNIV* ‹($\leq_a$)›]

end

context *order*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *Least-equality*:
assumes $P\ x$ **and** $\bigwedge y.\ P\ y \Longrightarrow x \leq_a y$
shows *Least* P = *Some* x
⟨*proof*⟩

lemma *LeastI2-order*:
assumes $P\ x$
and $\bigwedge y.\ P\ y \Longrightarrow x \leq_a y$
and $\bigwedge x.\ P\ x \Longrightarrow \forall y.\ P\ y \longrightarrow x \leq_a y \Longrightarrow Q\ x$
obtains z **where** *Least* P = *Some* z **and** $Q\ z$
⟨*proof*⟩

lemma *Least-ex1*:
assumes $\exists!x.\ P\ x \wedge (\forall y.\ P\ y \longrightarrow x \leq_a y)$
obtains x **where** *Least* P = *Some* x **and** $P\ x$ **and** $P\ z \Longrightarrow x \leq_a z$
⟨*proof*⟩

end

5.4.10 min and max

definition *min* :: [['a, 'a] ⇒ *bool*, 'a, 'a] ⇒ 'a **where**
min le a b = (*if le a b then a else b*)

ctr parametricity
in *min-def*

context *ord-syntax*
begin

abbreviation *min* **where** *min* ≡ *Type-Simple-Orders.min* ($\leq_a$)
abbreviation *max* **where** *max* ≡ *Type-Simple-Orders.min* ($\geq_a$)

end

context *ord*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *min-absorb1*: $x \leq_a y \Longrightarrow$ *min* $x\ y = x$
⟨*proof*⟩

end

context *order*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *min-absorb2*:
assumes $y \leq_a x$
shows *min x y* = *y*
⟨*proof*⟩

end

context *order-extremum*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *max-top*[*simp*]: *max extremum x* = *extremum*
⟨*proof*⟩

lemma *max-top2*[*simp*]: *max x extremum* = *extremum*
⟨*proof*⟩

lemma *min-top*[*simp*]: *min extremum x* = *x* ⟨*proof*⟩

lemma *min-top2*[*simp*]: *min x extremum* = *x* ⟨*proof*⟩

end

5.4.11 Monotonicity

definition *mono* ::
['*a set*, ['*a*, '*a*] ⇒ *bool*, ['*b*, '*b*] ⇒ *bool*, '*a* ⇒ '*b*] ⇒ *bool*
(‹(*on* - *with* - - : «*mono*» -)› [1000, 1000, 999, 1000] 10)
where
on U_a *with* op_1 op_2 : «*mono*» *f* ≡ ∀ *x*∈U_a. ∀ *y*∈U_a. op_1 *x y* ⟶ op_2 (*f x*) (*f y*)

ctr parametricity
in *mono-def*

context *ord-pair-syntax*
begin

abbreviation $mono_{ab}$
where $mono_{ab}$ ≡ *Type-Simple-Orders.mono UNIV* $(\leq_a)$ $(\leq_b)$
abbreviation $mono_{ba}$
where $mono_{ba}$ ≡ *Type-Simple-Orders.mono UNIV* $(\leq_b)$ $(\leq_a)$
abbreviation $antimono_{ab}$
where $antimono_{ab}$ ≡ *Type-Simple-Orders.mono UNIV* $(\leq_a)$ $(\geq_b)$
abbreviation $antimono_{ba}$
where $antimono_{ba}$ ≡ *Type-Simple-Orders.mono UNIV* $(\leq_b)$ $(\geq_a)$
abbreviation $strict\text{-}mono_{ab}$
where $strict\text{-}mono_{ab}$ ≡ *Type-Simple-Orders.mono UNIV* $(<_a)$ $(<_b)$
abbreviation $strict\text{-}mono_{ba}$

where *strict-mono*$_{ba}$ ≡ *Type-Simple-Orders.mono UNIV* (<$_b$) (<$_a$)
abbreviation *strict-antimono*$_{ab}$
where *strict-antimono*$_{ab}$ ≡ *Type-Simple-Orders.mono UNIV* (<$_a$) (>$_b$)
abbreviation *strict-antimono*$_{ba}$
where *strict-antimono*$_{ba}$ ≡ *Type-Simple-Orders.mono UNIV* (<$_b$) (>$_a$)

end

context *ord-pair*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

lemma *monoI*[*intro?*]:
assumes ⋀*x y*. *x* ≤$_a$ *y* ⟹ *f x* ≤$_b$ *f y*
shows *mono*$_{ab}$ *f*
⟨*proof*⟩

lemma *monoD*[*dest?*]:
assumes *mono*$_{ab}$ *f* **and** *x* ≤$_a$ *y*
shows *f x* ≤$_b$ *f y*
⟨*proof*⟩

lemma *monoE*:
assumes *mono*$_{ab}$ *f* **and** *x* ≤$_a$ *y*
obtains *f x* ≤$_b$ *f y*
⟨*proof*⟩

lemma *strict-monoI*[*intro?*]:
assumes ⋀*x y*. *x* <$_a$ *y* ⟹ *f x* <$_b$ *f y*
shows *strict-mono*$_{ab}$ *f*
⟨*proof*⟩

lemma *strict-monoD*[*dest?*]:
assumes *strict-mono*$_{ab}$ *f* **and** *x* <$_a$ *y*
shows *f x* <$_b$ *f y*
⟨*proof*⟩

lemma *strict-monoE*:
assumes *strict-mono*$_{ab}$ *f* **and** *x* <$_a$ *y*
obtains *f x* <$_b$ *f y*
⟨*proof*⟩

end

context *order-pair*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

lemma *strict-mono-mono*[*dest?*]:
assumes *strict-mono*$_{ab}$ *f*
shows *mono*$_{ab}$ *f*
⟨*proof*⟩

end

5.4.12 Set intervals

definition *ray* :: ['*a set*, ['*a*, '*a*] ⇒ *bool*, '*a*] ⇒ '*a set*
(‹(*on - with -* : {..⊏-})› [1000, 1000, 1000] 10)
where *on U with op* : {..⊏*u*} ≡ {*x* ∈ *U*. *op x u*}
definition *interval* ::
['*a set*, ['*a*, '*a*] ⇒ *bool*, ['*a*, '*a*] ⇒ *bool*, '*a*, '*a*] ⇒ '*a set*
(‹(*on - with - -* : {-⊏..⊏-})› [1000, 1000, 999, 1000, 1000] 10)
where *on U with op*$_1$ *op*$_2$: {*l*⊏..⊏*u*} ≡
(*on U with* (λ*x y*. *op*$_1$ *y x*) : {..⊏*l*}) ∩ (*on U with op*$_2$: {..⊏*u*})

lemma *ray-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows (*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> *A* ===> *rel-set A*) *ray ray*
⟨*proof*⟩

ctr relativization
assumes [*transfer-rule*]: *right-total A bi-unique A*
trp (*?'a A*)
in *interval-def*

lemma *interval-ge-le*:
(*on UNIV with* (λ*x y*. *le*$_a$ *y x*) (λ*x y*. *le*$_b$ *y x*) : {*l*⊏..⊏*h*}) =
(*on UNIV with le*$_b$ *le*$_a$: {*h*⊏..⊏*l*})
⟨*proof*⟩

context *ord-syntax*
begin

abbreviation *lessThan* (‹{..<$_a$-}›)
where {..<$_a$*u*} ≡ *on UNIV with* (<$_a$) : {..⊏*u*}
abbreviation *atMost* (‹{..≤$_a$-}›)
where {..≤$_a$*u*} ≡ *on UNIV with* (≤$_a$) : {..⊏*u*}
abbreviation *greaterThan* (‹{-<$_a$..}›)
where {*l*<$_a$..} ≡ *on UNIV with* (>$_a$) : {..⊏*l*}
abbreviation *atLeast* (‹{-≤$_a$..}›)
where {*l*≤$_a$..} ≡ *on UNIV with* (≥$_a$) : {..⊏*l*}
abbreviation *greaterThanLessThan* (‹{-<$_a$..<$_a$-}›)
where {*l*<$_a$..<$_a$*u*} ≡ *on UNIV with* (<$_a$) (<$_a$) : {*l*⊏..⊏*u*}
abbreviation *atLeastLessThan* (‹{-≤$_a$..<$_a$-}›)
where {*l*≤$_a$..<$_a$*u*} ≡ *on UNIV with* (≤$_a$) (<$_a$) : {*l*⊏..⊏*u*}
abbreviation *greaterThanAtMost* (‹{-<$_a$..≤$_a$-}›)
where {*l*<$_a$..≤$_a$*u*} ≡ *on UNIV with* (<$_a$) (≤$_a$) : {*l*⊏..⊏*u*}
abbreviation *atLeastAtMost* (‹{-≤$_a$..≤$_a$-}›)
where {*l*≤$_a$..≤$_a$*u*} ≡ *on UNIV with* (≤$_a$) (≤$_a$) : {*l*⊏..⊏*u*}
abbreviation *lessThanGreaterThan* (‹{->$_a$..>$_a$-}›)
where {*l*>$_a$..>$_a$*u*} ≡ *on UNIV with* (>$_a$) (>$_a$) : {*l*⊏..⊏*u*}
abbreviation *lessThanAtLeast* (‹{-≥$_a$..>$_a$-}›)
where {*l*≥$_a$..>$_a$*u*} ≡ *on UNIV with* (≥$_a$) (>$_a$) : {*l*⊏..⊏*u*}
abbreviation *atMostGreaterThan* (‹{->$_a$..≥$_a$-}›)
where {*l*>$_a$..≥$_a$*u*} ≡ *on UNIV with* (>$_a$) (≥$_a$) : {*l*⊏..⊏*u*}
abbreviation *atMostAtLeast* (‹{-≥$_a$..≥$_a$-}›)
where {*l*≥$_a$..≥$_a$*u*} ≡ *on UNIV with* (≥$_a$) (≥$_a$) : {*l*⊏..⊏*u*}

end

context *ord*

begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *lessThan-iff*[*iff*]: $(i \in \{..<_a k\}) = (i <_a k)$
⟨*proof*⟩

lemma *atLeast-iff*[*iff*]: $(i \in \{k\leq_a..\}) = (k \leq_a i)$
⟨*proof*⟩

lemma *greaterThanLessThan-iff*[*simp*]: $(i \in \{l<_a..<_a u\}) = (l <_a i \wedge i <_a u)$
⟨*proof*⟩

lemma *atLeastLessThan-iff*[*simp*]: $(i \in \{l\leq_a..<_a u\}) = (l \leq_a i \wedge i <_a u)$
⟨*proof*⟩

lemma *greaterThanAtMost-iff*[*simp*]: $(i \in \{l<_a..\leq_a u\}) = (l <_a i \wedge i \leq_a u)$
⟨*proof*⟩

lemma *atLeastAtMost-iff*[*simp*]: $(i \in \{l\leq_a..\leq_a u\}) = (l \leq_a i \wedge i \leq_a u)$
⟨*proof*⟩

lemma *greaterThanLessThan-eq*: $\{a<_a..<_a b\} = \{a<_a..\} \cap \{..<_a b\}$
⟨*proof*⟩

end

context *ord-pair-syntax*
begin

notation $ord_a.lessThan$ ($\langle\{..<_a\text{-}\}\rangle$)
and $ord_a.atMost$ ($\langle\{..\leq_a\text{-}\}\rangle$)
and $ord_a.greaterThan$ ($\langle\{\text{-}<_a..\}\rangle$)
and $ord_a.atLeast$ ($\langle\{\text{-}\leq_a..\}\rangle$)
and $ord_a.greaterThanLessThan$ ($\langle\{\text{-}<_a..<_a\text{-}\}\rangle$)
and $ord_a.atLeastLessThan$ ($\langle\{\text{-}\leq_a..<_a\text{-}\}\rangle$)
and $ord_a.greaterThanAtMost$ ($\langle\{\text{-}<_a..\leq_a\text{-}\}\rangle$)
and $ord_a.atLeastAtMost$ ($\langle\{\text{-}\leq_a..\leq_a\text{-}\}\rangle$)
and $ord_a.lessThanGreaterThan$ ($\langle\{\text{->}_a..>_a\text{-}\}\rangle$)
and $ord_a.lessThanAtLeast$ ($\langle\{\text{-}\geq_a..>_a\text{-}\}\rangle$)
and $ord_a.atMostGreaterThan$ ($\langle\{\text{->}_a..\geq_a\text{-}\}\rangle$)
and $ord_a.atMostAtLeast$ ($\langle\{\text{-}\geq_a..\geq_a\text{-}\}\rangle$)
and $ord_b.lessThan$ ($\langle\{..<_b\text{-}\}\rangle$)
and $ord_b.atMost$ ($\langle\{..\leq_b\text{-}\}\rangle$)
and $ord_b.greaterThan$ ($\langle\{\text{-}<_b..\}\rangle$)
and $ord_b.atLeast$ ($\langle\{\text{-}\leq_b..\}\rangle$)
and $ord_b.greaterThanLessThan$ ($\langle\{\text{-}<_b..<_b\text{-}\}\rangle$)
and $ord_b.atLeastLessThan$ ($\langle\{\text{-}\leq_b..<_b\text{-}\}\rangle$)
and $ord_b.greaterThanAtMost$ ($\langle\{\text{-}<_b..\leq_b\text{-}\}\rangle$)
and $ord_b.atLeastAtMost$ ($\langle\{\text{-}\leq_b..\leq_b\text{-}\}\rangle$)
and $ord_b.lessThanGreaterThan$ ($\langle\{\text{->}_b..>_b\text{-}\}\rangle$)
and $ord_b.lessThanAtLeast$ ($\langle\{\text{-}\geq_b..>_b\text{-}\}\rangle$)
and $ord_b.atMostGreaterThan$ ($\langle\{\text{->}_b..\geq_b\text{-}\}\rangle$)
and $ord_b.atMostAtLeast$ ($\langle\{\text{-}\geq_b..\geq_b\text{-}\}\rangle$)

end

context *preorder*

begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *Ioi-le-Ico*: $\{a<_a..\} \subseteq \{a\leq_a..\}$
⟨*proof*⟩

end

context *preorder*
begin

interpretation *ord-syntax* ⟨*proof*⟩

interpretation *preorder-dual le ls*
⟨*proof*⟩

lemma *single-Diff-lessThan*[*simp*]: $\{k\} - \{..<_a k\} = \{k\}$ ⟨*proof*⟩

lemma *atLeast-subset-iff*[*iff*]: $(\{x\leq_a..\} \subseteq \{y\leq_a..\}) = (y \leq_a x)$
⟨*proof*⟩

lemma *atLeastatMost-empty*[*simp*]:
assumes $b <_a a$
shows $\{a\leq_a..\leq_a b\} = \{\}$
⟨*proof*⟩

lemma *atLeastatMost-empty-iff*[*simp*]: $\{a\leq_a..\leq_a b\} = \{\} \longleftrightarrow (\neg\ a \leq_a b)$
⟨*proof*⟩

lemma *atLeastatMost-empty-iff2*[*simp*]: $\{\} = \{a\leq_a..\leq_a b\} \longleftrightarrow (\neg\ a \leq_a b)$
⟨*proof*⟩

lemma *atLeastLessThan-empty*[*simp*]:
assumes $b \leq_a a$
shows $\{a\leq_a..<_a b\} = \{\}$
⟨*proof*⟩

lemma *atLeastLessThan-empty-iff*[*simp*]: $\{a\leq_a..<_a b\} = \{\} \longleftrightarrow (\neg\ a <_a b)$
⟨*proof*⟩

lemma *atLeastLessThan-empty-iff2*[*simp*]: $\{\} = \{a\leq_a..<_a b\} \longleftrightarrow (\neg\ a <_a b)$
⟨*proof*⟩

lemma *greaterThanAtMost-empty*[*simp*]:
assumes $l \leq_a k$
shows $\{k<_a..\leq_a l\} = \{\}$
⟨*proof*⟩

lemma *greaterThanAtMost-empty-iff*[*simp*]: $\{k<_a..\leq_a l\} = \{\} \longleftrightarrow \neg\ k <_a l$
⟨*proof*⟩

lemma *greaterThanAtMost-empty-iff2*[*simp*]: $\{\} = \{k<_a..\leq_a l\} \longleftrightarrow \neg\ k <_a l$
⟨*proof*⟩

lemma *greaterThanLessThan-empty*[*simp*]:
assumes $l \leq_a k$
shows $\{k<_a..<_a l\} = \{\}$

⟨*proof*⟩

lemma *atLeastatMost-subset-iff*[*simp*]:
$\{a{\leq_a}..{\leq_a}b\} \leq \{c{\leq_a}..{\leq_a}d\} \longleftrightarrow (\neg\ a \leq_a b) \vee c \leq_a a \wedge b \leq_a d$
⟨*proof*⟩

lemma *atLeastatMost-psubset-iff*:
$\{a{\leq_a}..{\leq_a}b\} < \{c{\leq_a}..{\leq_a}d\} \longleftrightarrow$
$((\neg\ a \leq_a b) \vee c \leq_a a \wedge b \leq_a d \wedge (c <_a a \vee b <_a d)) \wedge c \leq_a d$
⟨*proof*⟩

lemma *Icc-subset-Ici-iff*[*simp*]:
$\{l{\leq_a}..{\leq_a}h\} \subseteq \{l'{\leq_a}..\} = (\neg\ l \leq_a h \vee l \geq_a l')$
⟨*proof*⟩

lemma *Icc-subset-Iic-iff*[*simp*]:
$\{l{\leq_a}..{\leq_a}h\} \subseteq \{..{\leq_a}h'\} = (\neg\ l \leq_a h \vee h \leq_a h')$
⟨*proof*⟩

lemma *not-Ici-eq-empty*[*simp*]: $\{l{\leq_a}..\} \neq \{\}$ ⟨*proof*⟩

lemmas *not-empty-eq-Ici-eq-empty*[*simp*] = *not-Ici-eq-empty*[*symmetric*]

lemma *Iio-Int-singleton*: $\{..{<_a}k\} \cap \{x\} = (\textit{if } x <_a k \textit{ then } \{x\} \textit{ else } \{\})$ ⟨*proof*⟩

lemma *ivl-disj-int-one*:
$\{..{\leq_a}l\} \cap \{l{<_a}..{<_a}u\} = \{\}$
$\{..{<_a}l\} \cap \{l{\leq_a}..{<_a}u\} = \{\}$
$\{..{\leq_a}l\} \cap \{l{<_a}..{\leq_a}u\} = \{\}$
$\{..{<_a}l\} \cap \{l{\leq_a}..{\leq_a}u\} = \{\}$
$\{l{<_a}..{\leq_a}u\} \cap \{u{<_a}..\} = \{\}$
$\{l{<_a}..{<_a}u\} \cap \{u{\leq_a}..\} = \{\}$
$\{l{\leq_a}..{\leq_a}u\} \cap \{u{<_a}..\} = \{\}$
$\{l{\leq_a}..{<_a}u\} \cap \{u{\leq_a}..\} = \{\}$
⟨*proof*⟩

lemma *ivl-disj-int-two*:
$\{l{<_a}..{<_a}m\} \cap \{m{\leq_a}..{<_a}u\} = \{\}$
$\{l{<_a}..{\leq_a}m\} \cap \{m{<_a}..{<_a}u\} = \{\}$
$\{l{\leq_a}..{<_a}m\} \cap \{m{\leq_a}..{<_a}u\} = \{\}$
$\{l{\leq_a}..{\leq_a}m\} \cap \{m{<_a}..{<_a}u\} = \{\}$
$\{l{<_a}..{<_a}m\} \cap \{m{\leq_a}..{\leq_a}u\} = \{\}$
$\{l{<_a}..{\leq_a}m\} \cap \{m{<_a}..{\leq_a}u\} = \{\}$
$\{l{\leq_a}..{<_a}m\} \cap \{m{\leq_a}..{\leq_a}u\} = \{\}$
$\{l{\leq_a}..{\leq_a}m\} \cap \{m{<_a}..{\leq_a}u\} = \{\}$
⟨*proof*⟩

end

context *order*
begin

interpretation *ord-syntax* ⟨*proof*⟩

interpretation *order-dual le ls*
⟨*proof*⟩

lemma *atMost-Int-atLeast*: $\{..{\leq_a}n\} \cap \{n{\leq_a}..\} = \{n\}$

⟨*proof*⟩

lemma *atLeast-eq-iff*[*iff*]: $(\{x\leq_a..\} = \{y\leq_a..\}) = (x = y)$
⟨*proof*⟩

lemma *atLeastLessThan-eq-atLeastAtMost-diff*: $\{a\leq_a..<_a b\} = \{a\leq_a..\leq_a b\} - \{b\}$
⟨*proof*⟩

lemma *greaterThanAtMost-eq-atLeastAtMost-diff*: $\{a<_a..\leq_a b\} = \{a\leq_a..\leq_a b\} - \{a\}$
⟨*proof*⟩

lemma *atLeastAtMost-singleton*[*simp*]: $\{a\leq_a..\leq_a a\} = \{a\}$
⟨*proof*⟩

lemma *atLeastAtMost-singleton′*:
assumes $a = b$
shows $\{a\leq_a..\leq_a b\} = \{a\}$
⟨*proof*⟩

lemma *Icc-eq-Icc*[*simp*]:
$\{l\leq_a..\leq_a h\} = \{l'\leq_a..\leq_a h'\} = (l = l' \wedge h = h' \vee \neg\ l \leq_a h \wedge \neg\ l' \leq_a h')$
⟨*proof*⟩

lemma *atLeastAtMost-singleton-iff*[*simp*]: $\{a\leq_a..\leq_a b\} = \{c\} \longleftrightarrow a = b \wedge b = c$
⟨*proof*⟩

end

context *order-extremum*
begin

interpretation *ord-syntax* ⟨*proof*⟩

lemma *atMost-eq-UNIV-iff*: $\{..\leq_a x\} = UNIV \longleftrightarrow x = extremum$
⟨*proof*⟩

end

context *no-extremum*
begin

interpretation *ord-syntax* ⟨*proof*⟩

interpretation *order-dual le ls*
⟨*proof*⟩

lemma *not-UNIV-le-Icc*[*simp*]: $\neg\ UNIV \subseteq \{l\leq_a..\leq_a h\}$
⟨*proof*⟩

lemma *not-UNIV-le-Iic*[*simp*]: $\neg\ UNIV \subseteq \{..\leq_a h\}$
⟨*proof*⟩

lemma *not-Ici-le-Icc*[*simp*]: $\neg\ \{l\leq_a..\} \subseteq \{l'\leq_a..\leq_a h'\}$
⟨*proof*⟩

lemma *not-Ici-le-Iic*[*simp*]: $\neg\ \{l\leq_a..\} \subseteq \{..\leq_a h'\}$
⟨*proof*⟩

lemma *not-UNIV-eq-Icc*[*simp*]: $UNIV \neq \{l'\leq_a..\leq_a h'\}$
⟨*proof*⟩

lemmas *not-Icc-eq-UNIV*[*simp*] = *not-UNIV-eq-Icc*[*symmetric*]

lemma *not-UNIV-eq-Iic*[*simp*]: $UNIV \neq \{..\leq_a h'\}$
⟨*proof*⟩

lemmas *not-Iic-eq-UNIV*[*simp*] = *not-UNIV-eq-Iic*[*symmetric*]

lemma *not-Icc-eq-Ici*[*simp*]: $\{l\leq_a..\leq_a h\} \neq \{l'\leq_a..\}$
⟨*proof*⟩

lemmas *not-Ici-eq-Icc*[*simp*] = *not-Icc-eq-Ici*[*symmetric*]

lemma *not-Iic-eq-Ici*[*simp*]: $\{..\leq_a h\} \neq \{l'\leq_a..\}$
⟨*proof*⟩

lemmas *not-Ici-eq-Iic*[*simp*] = *not-Iic-eq-Ici*[*symmetric*]

lemma *greaterThan-non-empty*[*simp*]: $\{x<_a..\} \neq \{\}$
⟨*proof*⟩

end

context *order*
begin

interpretation *ord-syntax* ⟨*proof*⟩
interpretation *order-pair le ls le ls* ⟨*proof*⟩
interpretation *ord-pair-syntax le ls le ls* ⟨*proof*⟩

lemma *mono-image-least*:
assumes *f-mono*: $mono_{a\,b}\ f$
and *f-img*: $f\ `\ \{m\leq_a..<_a n\} = \{m'\leq_a..<_a n'\}$
and $m <_a n$
shows $f\ m = m'$
⟨*proof*⟩

end

5.4.13 Bounded sets

definition *bdd* :: ['*a set*, ['*a*, '*a*] ⇒ *bool*, '*a set*] ⇒ *bool*
(‹(*on - with -* : «*bdd*» -)› [1000, 1000, 1000] 10)
where *bdd U op A* ⟷ $(\exists M{\in}U.\ \forall x \in A.\ op\ x\ M)$

ctr parametricity
in *bdd-def*

context *ord-syntax*
begin

abbreviation *bdd-above* **where** *bdd-above* ≡ *bdd UNIV* $(\leq_a)$
abbreviation *bdd-below* **where** *bdd-below* ≡ *bdd UNIV* $(\geq_a)$

end

context *preorder*
begin

interpretation *ord-syntax* ⟨*proof*⟩

interpretation *preorder-dual* ⟨*proof*⟩

lemma *bdd-aboveI*[*intro*]:
assumes $\bigwedge x.\ x \in A \Longrightarrow x \leq_a M$
shows *bdd-above A*
⟨*proof*⟩

lemma *bdd-belowI*[*intro*]:
assumes $\bigwedge x.\ x \in A \Longrightarrow m \leq_a x$
shows *bdd-below A*
⟨*proof*⟩

lemma *bdd-aboveI2*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq_a M$
shows *bdd-above* (*f* ‘ *A*)
⟨*proof*⟩

lemma *bdd-belowI2*:
assumes $\bigwedge x.\ x \in A \Longrightarrow m \leq_a f\ x$
shows *bdd-below* (*f* ‘ *A*)
⟨*proof*⟩

lemma *bdd-above-empty*[*simp*, *intro*]: *bdd-above* {}
⟨*proof*⟩

lemma *bdd-below-empty*[*simp*, *intro*]: *bdd-below* {}
⟨*proof*⟩

lemma *bdd-above-mono*:
assumes *bdd-above B* **and** $A \subseteq B$
shows *bdd-above A*
⟨*proof*⟩

lemma *bdd-below-mono*:
assumes *bdd-below B* **and** $A \subseteq B$
shows *bdd-below A*
⟨*proof*⟩

lemma *bdd-above-Int1*[*simp*]:
assumes *bdd-above A*
shows *bdd-above* ($A \cap B$)
⟨*proof*⟩

lemma *bdd-above-Int2*[*simp*]:
assumes *bdd-above B*
shows *bdd-above* ($A \cap B$)
⟨*proof*⟩

lemma *bdd-below-Int1*[*simp*]:
assumes *bdd-below A*
shows *bdd-below* ($A \cap B$)
⟨*proof*⟩

lemma *bdd-below-Int2*[*simp*]:
assumes *bdd-below B*
shows *bdd-below* $(A \cap B)$
⟨*proof*⟩

lemma *bdd-above-Ioo*[*simp*, *intro*]: *bdd-above* $\{a<_a..<_a b\}$
⟨*proof*⟩

lemma *bdd-above-Ico*[*simp*, *intro*]: *bdd-above* $\{a\leq_a..<_a b\}$
⟨*proof*⟩

lemma *bdd-above-Iio*[*simp*, *intro*]: *bdd-above* $\{..<_a b\}$
⟨*proof*⟩

lemma *bdd-above-Ioc*[*simp*, *intro*]: *bdd-above* $\{a<_a..\leq_a b\}$ ⟨*proof*⟩

lemma *bdd-above-Icc*[*simp*, *intro*]: *bdd-above* $\{a\leq_a..\leq_a b\}$
⟨*proof*⟩

lemma *bdd-above-Iic*[*simp*, *intro*]: *bdd-above* $\{..\leq_a b\}$
⟨*proof*⟩

lemma *bdd-below-Ioo*[*simp*, *intro*]: *bdd-below* $\{a<_a..<_a b\}$
⟨*proof*⟩

lemma *bdd-below-Ioc*[*simp*, *intro*]: *bdd-below* $\{a<_a..\leq_a b\}$
⟨*proof*⟩

lemma *bdd-below-Ioi*[*simp*, *intro*]: *bdd-below* $\{a<_a..\}$
⟨*proof*⟩

lemma *bdd-below-Ico*[*simp*, *intro*]: *bdd-below* $\{a\leq_a..<_a b\}$ ⟨*proof*⟩

lemma *bdd-below-Icc*[*simp*, *intro*]: *bdd-below* $\{a\leq_a..\leq_a b\}$ ⟨*proof*⟩

lemma *bdd-below-Ici*[*simp*, *intro*]: *bdd-below* $\{a\leq_a..\}$
⟨*proof*⟩

end

context *order-pair*
begin

interpretation *ord-pair-syntax* ⟨*proof*⟩

lemma *bdd-above-image-mono*:
assumes $mono_{ab}$ *f* **and** ord_a*.bdd-above A*
shows ord_b*.bdd-above* $(f\ `\ A)$
⟨*proof*⟩

lemma *bdd-below-image-mono*:
assumes $mono_{ab}$ *f* **and** ord_a*.bdd-below A*
shows ord_b*.bdd-below* $(f\ `\ A)$
⟨*proof*⟩

lemma *bdd-above-image-antimono*:
assumes $antimono_{ab}$ *f* **and** ord_a*.bdd-below A*
shows ord_b*.bdd-above* $(f\ `\ A)$

⟨*proof*⟩

lemma *bdd-below-image-antimono*:
assumes $antimono_{ab}$ f **and** ord_a.*bdd-above* A
shows ord_b.*bdd-below* (f ‘ A)
⟨*proof*⟩

end

context *order-extremum*
begin

interpretation *ord-syntax* ⟨*proof*⟩
interpretation *order-dual* ⟨*proof*⟩

lemma *bdd-above-top*[*simp*, *intro*!]: *bdd-above* A
⟨*proof*⟩

end

5.5 Abstract orders on explicit sets

5.5.1 Background

Some of the results presented in this section were ported (with amendments and additions) from the theories *Orderings* and *Set-Interval* in the main library of Isabelle/HOL.

5.5.2 Order operations

locale *ord-ow* =
  fixes $U :: 'a\ set$ **and** $le\ ls :: ['a, 'a] \Rightarrow bool$
begin

tts-register-sbts $le \mid U$
⟨*proof*⟩

tts-register-sbts $ls \mid U$
⟨*proof*⟩

end

locale *ord-syntax-ow* = *ord-ow U le ls*
  for $U :: 'a\ set$ **and** $le\ ls :: ['a, 'a] \Rightarrow bool$
begin

notation
  le ($\langle '(\leq_a ')\rangle$) **and**
  le (**infix** $\langle \leq_a \rangle$ 50) **and**
  ls ($\langle '(<_a ')\rangle$) **and**
  ls (**infix** $\langle <_a \rangle$ 50)

abbreviation (*input*) ge (**infix** $\langle \geq_a \rangle$ 50)
  where $x \geq_a y \equiv y \leq_a x$
abbreviation (*input*) gt (**infix** $\langle >_a \rangle$ 50)
  where $x >_a y \equiv y <_a x$

notation
  ge ($\langle '(\geq_a ')\rangle$) **and**
  ge (**infix** $\langle \geq_a \rangle$ 50) **and**
  gt ($\langle '(>_a ')\rangle$) **and**
  gt (**infix** $\langle >_a \rangle$ 50)

abbreviation *Least* **where** *Least* ≡ *Type-Simple-Orders.Least U* $(\leq_a)$
abbreviation *Greatest* **where** *Greatest* ≡ *Type-Simple-Orders.Least U* $(\geq_a)$

abbreviation *min* **where** *min* ≡ *Type-Simple-Orders.min* $(\leq_a)$
abbreviation *max* **where** *max* ≡ *Type-Simple-Orders.min* $(\geq_a)$

abbreviation *lessThan* $(\langle \{..<_a\text{-}\} \rangle)$ **where** $\{..<_a u\} \equiv$ *on U with* $(<_a) : \{..\sqsubset u\}$
abbreviation *atMost* $(\langle \{..\leq_a\text{-}\} \rangle)$ **where** $\{..\leq_a u\} \equiv$ *on U with* $(\leq_a) : \{..\sqsubset u\}$
abbreviation *greaterThan* $(\langle \{\text{-}<_a..\} \rangle)$ **where** $\{l<_a..\} \equiv$ *on U with* $(>_a) : \{..\sqsubset l\}$
abbreviation *atLeast* $(\langle \{\text{-}\leq_a..\} \rangle)$ **where** $\{l\leq_a..\} \equiv$ *on U with* $(\geq_a) : \{..\sqsubset l\}$
abbreviation *greaterThanLessThan* $(\langle \{\text{-}<_a..<_a\text{-}\} \rangle)$
  where $\{l<_a..<_a u\} \equiv$ *on U with* $(<_a)\ (<_a) : \{l\sqsubset..\sqsubset u\}$
abbreviation *atLeastLessThan* $(\langle \{\text{-}\leq_a..<_a\text{-}\} \rangle)$
  where $\{l\leq_a..<_a u\} \equiv$ *on U with* $(\leq_a)\ (<_a) : \{l\sqsubset..\sqsubset u\}$
abbreviation *greaterThanAtMost* $(\langle \{\text{-}<_a..\leq_a\text{-}\} \rangle)$
  where $\{l<_a..\leq_a u\} \equiv$ *on U with* $(<_a)\ (\leq_a) : \{l\sqsubset..\sqsubset u\}$

abbreviation *atLeastAtMost* (‹{-≤$_a$..≤$_a$-}›)
  where {l≤$_a$..≤$_a$$u$} ≡ *on U with* (≤$_a$) (≤$_a$) : {$l$⊏..⊏$u$}
abbreviation *lessThanGreaterThan* (‹{->$_a$..>$_a$-}›)
  where {l>$_a$..>$_a$$u$} ≡ *on U with* (>$_a$) (>$_a$) : {$l$⊏..⊏$u$}
abbreviation *lessThanAtLeast* (‹{-≥$_a$..>$_a$-}›)
  where {l≥$_a$..>$_a$$u$} ≡ *on U with* (≥$_a$) (>$_a$) : {$l$⊏..⊏$u$}
abbreviation *atMostGreaterThan* (‹{->$_a$..≥$_a$-}›)
  where {l>$_a$..≥$_a$$u$} ≡ *on U with* (>$_a$) (≥$_a$) : {$l$⊏..⊏$u$}
abbreviation *atMostAtLeast* (‹{-≥$_a$..≥$_a$-}›)
  where {l≥$_a$..≥$_a$$u$} ≡ *on U with* (≥$_a$) (≥$_a$) : {$l$⊏..⊏$u$}

abbreviation *bdd-above* **where** *bdd-above* ≡ *bdd U* (≤$_a$)
abbreviation *bdd-below* **where** *bdd-below* ≡ *bdd U* (≥$_a$)

end

locale *ord-dual-ow* = *ord-syntax-ow U le ls*
  for *U* :: *'a set* **and** *le ls* :: ['*a*, '*a*] ⇒ *bool*
begin

sublocale *dual*: *ord-ow U ge gt* ⟨*proof*⟩

end

locale *ord-pair-ow* = *ord*$_a$: *ord-ow U*$_a$ *le*$_a$ *ls*$_a$ + *ord*$_b$: *ord-ow U*$_b$ *le*$_b$ *ls*$_b$
  for *U*$_a$ :: *'a set* **and** *le*$_a$ *ls*$_a$ **and** *U*$_b$ :: *'b set* **and** *le*$_b$ *ls*$_b$
begin

sublocale *rev*: *ord-pair-ow U*$_b$ *le*$_b$ *ls*$_b$ *U*$_a$ *le*$_a$ *ls*$_a$ ⟨*proof*⟩

end

locale *ord-pair-syntax-ow* = *ord-pair-ow U*$_a$ *le*$_a$ *ls*$_a$ *U*$_b$ *le*$_b$ *ls*$_b$
  for *U*$_a$ :: *'a set* **and** *le*$_a$ *ls*$_a$ **and** *U*$_b$ :: *'b set* **and** *le*$_b$ *ls*$_b$
begin

sublocale *ord*$_a$: *ord-syntax-ow U*$_a$ *le*$_a$ *ls*$_a$ + *ord*$_b$: *ord-syntax-ow U*$_b$ *le*$_b$ *ls*$_b$ ⟨*proof*⟩

notation *le*$_a$ (‹'(≤$_a$')›)
  and *le*$_a$ (**infix** ‹≤$_a$› 50)
  and *ls*$_a$ (‹'(<$_a$')›)
  and *ls*$_a$ (**infix** ‹<$_a$› 50)
  and *le*$_b$ (‹'(≤$_b$')›)
  and *le*$_b$ (**infix** ‹≤$_b$› 50)
  and *ls*$_b$ (‹'(<$_b$')›)
  and *ls*$_b$ (**infix** ‹<$_b$› 50)

notation *ord*$_a$.*ge* (‹'(≥$_a$')›)
  and *ord*$_a$.*ge* (**infix** ‹≥$_a$› 50)
  and *ord*$_a$.*gt* (‹'(>$_a$')›)
  and *ord*$_a$.*gt* (**infix** ‹>$_a$› 50)
  and *ord*$_b$.*ge* (‹'(≥$_b$')›)
  and *ord*$_b$.*ge* (**infix** ‹≥$_b$› 50)
  and *ord*$_b$.*gt* (‹'(>$_b$')›)
  and *ord*$_b$.*gt* (**infix** ‹>$_b$› 50)

abbreviation *mono*$_{ab}$
  where *mono*$_{ab}$ ≡ *Type-Simple-Orders.mono U*$_a$ (≤$_a$) (≤$_b$)

abbreviation $mono_{ba}$
  where $mono_{ba} \equiv$ *Type-Simple-Orders.mono* U_b $(\leq_b)$ $(\leq_a)$
abbreviation $antimono_{ab}$
  where $antimono_{ab} \equiv$ *Type-Simple-Orders.mono* U_a $(\leq_a)$ $(\geq_b)$
abbreviation $antimono_{ba}$
  where $antimono_{ba} \equiv$ *Type-Simple-Orders.mono* U_b $(\leq_b)$ $(\geq_a)$
abbreviation $strict\text{-}mono_{ab}$
  where $strict\text{-}mono_{ab} \equiv$ *Type-Simple-Orders.mono* U_a $(<_a)$ $(<_b)$
abbreviation $strict\text{-}mono_{ba}$
  where $strict\text{-}mono_{ba} \equiv$ *Type-Simple-Orders.mono* U_b $(<_b)$ $(<_a)$
abbreviation $strict\text{-}antimono_{ab}$
  where $strict\text{-}antimono_{ab} \equiv$ *Type-Simple-Orders.mono* U_a $(<_a)$ $(>_b)$
abbreviation $strict\text{-}antimono_{ba}$
  where $strict\text{-}antimono_{ba} \equiv$ *Type-Simple-Orders.mono* U_b $(<_b)$ $(>_a)$

notation $ord_a.lessThan$ $(\langle\{..<_a\text{-}\}\rangle)$
  and $ord_a.atMost$ $(\langle\{..\leq_a\text{-}\}\rangle)$
  and $ord_a.greaterThan$ $(\langle\{\text{-}<_a..\}\rangle)$
  and $ord_a.atLeast$ $(\langle\{\text{-}\leq_a..\}\rangle)$
  and $ord_a.greaterThanLessThan$ $(\langle\{\text{-}<_a..<_a\text{-}\}\rangle)$
  and $ord_a.atLeastLessThan$ $(\langle\{\text{-}\leq_a..<_a\text{-}\}\rangle)$
  and $ord_a.greaterThanAtMost$ $(\langle\{\text{-}<_a..\leq_a\text{-}\}\rangle)$
  and $ord_a.atLeastAtMost$ $(\langle\{\text{-}\leq_a..\leq_a\text{-}\}\rangle)$
  and $ord_a.lessThanGreaterThan$ $(\langle\{\text{->}_a..>_a\text{-}\}\rangle)$
  and $ord_a.lessThanAtLeast$ $(\langle\{\text{-}\geq_a..>_a\text{-}\}\rangle)$
  and $ord_a.atMostGreaterThan$ $(\langle\{\text{->}_a..\geq_a\text{-}\}\rangle)$
  and $ord_a.atMostAtLeast$ $(\langle\{\text{-}\geq_a..\geq_a\text{-}\}\rangle)$
  and $ord_b.lessThan$ $(\langle\{..<_b\text{-}\}\rangle)$
  and $ord_b.atMost$ $(\langle\{..\leq_b\text{-}\}\rangle)$
  and $ord_b.greaterThan$ $(\langle\{\text{-}<_b..\}\rangle)$
  and $ord_b.atLeast$ $(\langle\{\text{-}\leq_b..\}\rangle)$
  and $ord_b.greaterThanLessThan$ $(\langle\{\text{-}<_b..<_b\text{-}\}\rangle)$
  and $ord_b.atLeastLessThan$ $(\langle\{\text{-}\leq_b..<_b\text{-}\}\rangle)$
  and $ord_b.greaterThanAtMost$ $(\langle\{\text{-}<_b..\leq_b\text{-}\}\rangle)$
  and $ord_b.atLeastAtMost$ $(\langle\{\text{-}\leq_b..\leq_b\text{-}\}\rangle)$
  and $ord_b.lessThanGreaterThan$ $(\langle\{\text{->}_b..>_b\text{-}\}\rangle)$
  and $ord_b.lessThanAtLeast$ $(\langle\{\text{-}\geq_b..>_b\text{-}\}\rangle)$
  and $ord_b.atMostGreaterThan$ $(\langle\{\text{->}_b..\geq_b\text{-}\}\rangle)$
  and $ord_b.atMostAtLeast$ $(\langle\{\text{-}\geq_b..\geq_b\text{-}\}\rangle)$

end

locale *ord-pair-dual-ow* = *ord-pair-syntax-ow* U_a le_a ls_a U_b le_b ls_b
  for $U_a :: 'a\ set$ **and** le_a ls_a **and** $U_b :: 'b\ set$ **and** le_b ls_b

context *ord-pair-dual-ow*
begin

sublocale *ord-dual*: *ord-pair-ow* U_a $\langle(\leq_a)\rangle$ $\langle(<_a)\rangle$ U_b $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ ⟨*proof*⟩
sublocale *dual-ord*: *ord-pair-ow* U_a $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ U_b $\langle(\leq_b)\rangle$ $\langle(<_b)\rangle$ ⟨*proof*⟩
sublocale *dual-dual*: *ord-pair-ow* U_a $\langle(\geq_a)\rangle$ $\langle(>_a)\rangle$ U_b $\langle(\geq_b)\rangle$ $\langle(>_b)\rangle$ ⟨*proof*⟩

end

Relativization

context *ord-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
rewriting *ctr-simps*
eliminating through *auto*
begin

tts-lemma *lessThan-iff*:
assumes $i \in U$ **and** $k \in U$
shows $(i \in \{..<_a k\}) = (i <_a k)$
is *ord.lessThan-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-iff*:
assumes $i \in U$ **and** $l \in U$ **and** $u \in U$
shows $(i \in \{l<_a..<_a u\}) = (i <_a u \wedge l <_a i)$
is *ord.greaterThanLessThan-iff*⟨*proof*⟩

tts-lemma *greaterThanLessThan-eq*:
assumes $a \in U$ **and** $b \in U$
shows $\{a<_a..<_a b\} = \{a<_a..\} \cap \{..<_a b\}$
is *ord.greaterThanLessThan-eq*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
rewriting *ctr-simps*
eliminating through *auto*
begin

tts-lemma *min-absorb*1:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_a y$
shows $min\ x\ y = x$
is *ord.min-absorb*1⟨*proof*⟩

tts-lemma *atLeast-iff*:
assumes $i \in U$ **and** $k \in U$
shows $(i \in \{k\leq_a..\}) = (k \leq_a i)$
is *ord.atLeast-iff*⟨*proof*⟩

tts-lemma *atLeastAtMost-iff*:
assumes $i \in U$ **and** $l \in U$ **and** $u \in U$
shows $(i \in \{l\leq_a..\leq_a u\}) = (i \leq_a u \wedge l \leq_a i)$
is *ord.atLeastAtMost-iff*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
and (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
rewriting *ctr-simps*
eliminating through *auto*
begin

tts-lemma *atLeastLessThan-iff*:
assumes $i \in U$ **and** $l \in U$ **and** $u \in U$
shows $(i \in \{l \leq_a .. <_a u\}) = (l \leq_a i \wedge i <_a u)$
is *ord.atLeastLessThan-iff*⟨*proof*⟩

tts-lemma *greaterThanAtMost-iff*:
assumes $i \in U$ **and** $l \in U$ **and** $u \in U$
shows $(i \in \{l <_a .. \leq_a u\}) = (i \leq_a u \wedge l <_a i)$
is *ord.greaterThanAtMost-iff*⟨*proof*⟩

end

end

context *ord-pair-ow*
begin

interpretation *ord-pair-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** U_a) **and** (*?'b* **to** U_b)
sbterms: (‹*?le*$_a$::*?'a*⇒*?'a*⇒*bool*› **to** le_a) **and** (‹*?le*$_b$::*?'b*⇒*?'b*⇒*bool*› **to** le_b)
rewriting *ctr-simps*
eliminating through (*simp add*: *Type-Simple-Orders.mono-def*)
begin

tts-lemma *monoD*:
assumes $x \in U_a$ **and** $y \in U_a$ **and** $mono_{ab}\ f$ **and** $x \leq_a y$
shows $f\ x \leq_b f\ y$
is *ord-pair.monoD*⟨*proof*⟩

tts-lemma *monoI*:
assumes $\bigwedge x\ y.\ [\![x \in U_a;\ y \in U_a;\ x \leq_a y]\!] \Longrightarrow f\ x \leq_b f\ y$
shows $mono_{ab}\ f$
is *ord-pair.monoI*⟨*proof*⟩

tts-lemma *monoE*:
assumes $x \in U_a$
and $y \in U_a$
and $mono_{ab}\ f$
and $x \leq_a y$
and $f\ x \leq_b f\ y \Longrightarrow$ *thesis*
shows *thesis*
is *ord-pair.monoE*⟨*proof*⟩

end

tts-context
tts: (*?'a* **to** U_a) **and** (*?'b* **to** U_b)
sbterms: (‹*?ls*$_a$::*?'a*⇒*?'a*⇒*bool*› **to** le_a) **and** (‹*?ls*$_b$::*?'b*⇒*?'b*⇒*bool*› **to** le_b)
rewriting *ctr-simps*
eliminating through (*simp add*: *Type-Simple-Orders.mono-def*)
begin

tts-lemma *strict-monoD*:
assumes $x \in U_a$
and $y \in U_a$

and $mono_{ab}\ f$
and $x \leq_a y$
shows $f\ x \leq_b f\ y$
is *ord-pair.strict-monoD*⟨*proof*⟩

tts-lemma *strict-monoI*:
assumes $\bigwedge x\ y.\ [\![x \in U_a;\ y \in U_a;\ x \leq_a y]\!] \Longrightarrow f\ x \leq_b f\ y$
shows $mono_{ab}\ f$
is *ord-pair.strict-monoI*⟨*proof*⟩

tts-lemma *strict-monoE*:
assumes $x \in U_a$
and $y \in U_a$
and $mono_{ab}\ f$
and $x \leq_a y$
and $f\ x \leq_b f\ y \Longrightarrow thesis$
shows *thesis*
is *ord-pair.strict-monoE*⟨*proof*⟩

end

end

5.5.3 Preorders

Definitions and common properties

locale *preorder-ow* = *ord-ow U le ls*
for $U :: 'a\ set$ **and** *le ls* +
assumes *less-le-not-le*: $[\![\ x \in U;\ y \in U\ ]\!] \Longrightarrow ls\ x\ y \longleftrightarrow le\ x\ y \wedge \neg\ (le\ y\ x)$
and *order-refl*[*iff*]: $x \in U \Longrightarrow le\ x\ x$
and *order-trans*: $[\![\ x \in U;\ y \in U;\ z \in U;\ le\ x\ y;\ le\ y\ z\ ]\!] \Longrightarrow le\ x\ z$

locale *preorder-dual-ow* = *preorder-ow U le ls* **for** $U :: 'a\ set$ **and** *le ls*
begin

sublocale *ord-dual-ow* ⟨*proof*⟩

sublocale *dual*: *preorder-ow U ge gt*
⟨*proof*⟩

end

locale *ord-preorder-ow* =
ord-pair-ow $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$ + ord_b: *preorder-ow* $U_b\ le_b\ ls_b$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$

locale *ord-preorder-dual-ow* = *ord-preorder-ow* $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$
begin

sublocale *ord-pair-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *ord-preorder-ow* $U_a\ ‹(\leq_a)›\ ‹(<_a)›\ U_b\ ‹(\geq_b)›\ ‹(>_b)›$
⟨*proof*⟩
sublocale *dual-ord*: *ord-preorder-ow* $U_a\ ‹(\geq_a)›\ ‹(>_a)›\ U_b\ ‹(\leq_b)›\ ‹(<_b)›$
⟨*proof*⟩
sublocale *dual-dual*: *ord-preorder-ow* $U_a\ ‹(\geq_a)›\ ‹(>_a)›\ U_b\ ‹(\geq_b)›\ ‹(>_b)›$
⟨*proof*⟩

end

locale *preorder-pair-ow* =
ord-preorder-ow U_a le_a ls_a U_b le_b ls_b + ord_a: *preorder-ow* U_a le_a ls_a
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *rev*: *preorder-pair-ow* U_b le_b ls_b U_a le_a ls_a ⟨*proof*⟩

end

locale *preorder-pair-dual-ow* = *preorder-pair-ow* U_a le_a ls_a U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *ord-preorder-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-pair-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *preorder-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
⟨*proof*⟩
sublocale *dual-dual*: *preorder-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

Transfer rules

lemma *preorder-ow*[*ud-with*]: *preorder* = *preorder-ow UNIV*
⟨*proof*⟩

lemma *ord-preorder-ow*[*ud-with*]: *ord-preorder* = *ord-preorder-ow UNIV*
⟨*proof*⟩

lemma *preorder-pair-ow*[*ud-with*]:
preorder-pair =
(λle_a ls_a le_b ls_b. *preorder-pair-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
⟨*proof*⟩

context
includes *lifting-syntax*
begin

lemma *preorder-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
preorder-ow preorder-ow
⟨*proof*⟩

lemma *ord-preorder-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
ord-preorder-ow ord-preorder-ow
⟨*proof*⟩

lemma *preorder-pair-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A right-total B*
shows

(
rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
(=)
) *preorder-pair-ow preorder-pair-ow*
⟨*proof*⟩

end

Relativization

context *preorder-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
and (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
rewriting *ctr-simps*
substituting *preorder-ow-axioms*
eliminating through *auto*
begin

tts-lemma *less-irrefl*:
assumes *x* ∈ *U*
shows ¬ *x* <$_a$ *x*
is *preorder.less-irrefl*⟨*proof*⟩

tts-lemma *eq-refl*:
assumes *y* ∈ *U* **and** *x* = *y*
shows *x* ≤$_a$ *y*
is *preorder.eq-refl*⟨*proof*⟩

tts-lemma *less-imp-le*:
assumes *x* ∈ *U* **and** *y* ∈ *U* **and** *x* <$_a$ *y*
shows *x* ≤$_a$ *y*
is *preorder.less-imp-le*⟨*proof*⟩

tts-lemma *strict-implies-not-eq*:
assumes *b* ∈ *U* **and** *a* <$_a$ *b*
shows *a* ≠ *b*
is *preorder.strict-implies-not-eq*⟨*proof*⟩

tts-lemma *less-not-sym*:
assumes *x* ∈ *U* **and** *y* ∈ *U* **and** *x* <$_a$ *y*
shows ¬ *y* <$_a$ *x*
is *preorder.less-not-sym*⟨*proof*⟩

tts-lemma *not-empty-eq-Ici-eq-empty*:
assumes *l* ∈ *U*
shows {} ≠ {*l*≤$_a$..}
is *preorder.not-empty-eq-Ici-eq-empty*⟨*proof*⟩

tts-lemma *not-Ici-eq-empty*:
assumes *l* ∈ *U*
shows {*l*≤$_a$..} ≠ {}

is *preorder.not-Ici-eq-empty⟨proof⟩*

tts-lemma *asym*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_a b$ **and** $b <_a a$
shows *False*
is *preorder.asym⟨proof⟩*

tts-lemma *less-asym'*:
assumes $a \in U$ **and** $b \in U$ **and** $a <_a b$ **and** $b <_a a$
shows P
is *preorder.less-asym'⟨proof⟩*

tts-lemma *less-imp-not-less*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_a y$
shows $(\neg\ y <_a x) = True$
is *preorder.less-imp-not-less⟨proof⟩*

tts-lemma *single-Diff-lessThan*:
assumes $k \in U$
shows $\{k\} - \{..<_a k\} = \{k\}$
is *preorder.single-Diff-lessThan⟨proof⟩*

tts-lemma *less-imp-triv*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_a y$
shows $(y <_a x \longrightarrow P) = True$
is *preorder.less-imp-triv⟨proof⟩*

tts-lemma *ivl-disj-int-one*:
assumes $l \in U$ **and** $u \in U$
shows
$\{..\leq_a l\} \cap \{l<_a..<_a u\} = \{\}$
$\{..<_a l\} \cap \{l\leq_a..<_a u\} = \{\}$
$\{..\leq_a l\} \cap \{l<_a..\leq_a u\} = \{\}$
$\{..<_a l\} \cap \{l\leq_a..\leq_a u\} = \{\}$
$\{l<_a..\leq_a u\} \cap \{u<_a..\} = \{\}$
$\{l<_a..<_a u\} \cap \{u\leq_a..\} = \{\}$
$\{l\leq_a..\leq_a u\} \cap \{u<_a..\} = \{\}$
$\{l\leq_a..<_a u\} \cap \{u\leq_a..\} = \{\}$
is *preorder.ivl-disj-int-one⟨proof⟩*

tts-lemma *atLeastatMost-empty-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(\{\} = \{a\leq_a..\leq_a b\}) = (\neg\ a \leq_a b)$
is *preorder.atLeastatMost-empty-iff2⟨proof⟩*

tts-lemma *atLeastLessThan-empty-iff2*:
assumes $a \in U$ **and** $b \in U$
shows $(\{\} = \{a\leq_a..<_a b\}) = (\neg\ a <_a b)$
is *preorder.atLeastLessThan-empty-iff2⟨proof⟩*

tts-lemma *greaterThanAtMost-empty-iff2*:
assumes $k \in U$ **and** $l \in U$
shows $(\{\} = \{k<_a..\leq_a l\}) = (\neg\ k <_a l)$
is *preorder.greaterThanAtMost-empty-iff2⟨proof⟩*

tts-lemma *atLeastatMost-empty-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(\{a\leq_a..\leq_a b\} = \{\}) = (\neg\ a \leq_a b)$

is *preorder.atLeastatMost-empty-iff*⟨*proof*⟩

tts-lemma *atLeastLessThan-empty-iff*:
assumes $a \in U$ **and** $b \in U$
shows $(\{a\leq_a..<_a b\} = \{\}) = (\neg\ a <_a b)$
is *preorder.atLeastLessThan-empty-iff*⟨*proof*⟩

tts-lemma *greaterThanAtMost-empty-iff*:
assumes $k \in U$ **and** $l \in U$
shows $(\{k<_a..\leq_a l\} = \{\}) = (\neg\ k <_a l)$
is *preorder.greaterThanAtMost-empty-iff*⟨*proof*⟩

tts-lemma *atLeastLessThan-empty*:
assumes $b \in U$ **and** $a \in U$ **and** $b \leq_a a$
shows $\{a\leq_a..<_a b\} = \{\}$
is *preorder.atLeastLessThan-empty*⟨*proof*⟩

tts-lemma *greaterThanAtMost-empty*:
assumes $l \in U$ **and** $k \in U$ **and** $l \leq_a k$
shows $\{k<_a..\leq_a l\} = \{\}$
is *preorder.greaterThanAtMost-empty*⟨*proof*⟩

tts-lemma *greaterThanLessThan-empty*:
assumes $l \in U$ **and** $k \in U$ **and** $l \leq_a k$
shows $\{k<_a..<_a l\} = \{\}$
is *preorder.greaterThanLessThan-empty*⟨*proof*⟩

tts-lemma *le-less-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x \leq_a y$ **and** $y <_a z$
shows $x <_a z$
is *preorder.le-less-trans*⟨*proof*⟩

tts-lemma *atLeastatMost-empty*:
assumes $b \in U$ **and** $a \in U$ **and** $b <_a a$
shows $\{a\leq_a..\leq_a b\} = \{\}$
is *preorder.atLeastatMost-empty*⟨*proof*⟩

tts-lemma *less-le-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_a y$ **and** $y \leq_a z$
shows $x <_a z$
is *preorder.less-le-trans*⟨*proof*⟩

tts-lemma *less-trans*:
assumes $x \in U$ **and** $y \in U$ **and** $z \in U$ **and** $x <_a y$ **and** $y <_a z$
shows $x <_a z$
is *preorder.less-trans*⟨*proof*⟩

tts-lemma *ivl-disj-int-two*:
assumes $l \in U$ **and** $m \in U$ **and** $u \in U$
shows
$\{l<_a..<_a m\} \cap \{m\leq_a..<_a u\} = \{\}$
$\{l<_a..\leq_a m\} \cap \{m<_a..<_a u\} = \{\}$
$\{l\leq_a..<_a m\} \cap \{m\leq_a..<_a u\} = \{\}$
$\{l\leq_a..\leq_a m\} \cap \{m<_a..<_a u\} = \{\}$
$\{l<_a..<_a m\} \cap \{m\leq_a..\leq_a u\} = \{\}$
$\{l<_a..\leq_a m\} \cap \{m<_a..\leq_a u\} = \{\}$
$\{l\leq_a..<_a m\} \cap \{m\leq_a..\leq_a u\} = \{\}$
$\{l\leq_a..\leq_a m\} \cap \{m<_a..\leq_a u\} = \{\}$

is *preorder.ivl-disj-int-two⟨proof⟩*

tts-lemma *less-asym*:
assumes $x \in U$ **and** $y \in U$ **and** $x <_a y$ **and** $\neg P \Longrightarrow y <_a x$
shows P
is *preorder.less-asym⟨proof⟩*

tts-lemma *Iio-Int-singleton*:
assumes $k \in U$ **and** $x \in U$
shows $\{..<_a k\} \cap \{x\} = (\mathit{if}\ x <_a k\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})$
is *preorder.Iio-Int-singleton⟨proof⟩*

tts-lemma *Ioi-le-Ico*:
assumes $a \in U$
shows $\{a<_a..\} \subseteq \{a\leq_a..\}$
is *preorder.Ioi-le-Ico⟨proof⟩*

tts-lemma *Icc-subset-Iic-iff*:
assumes $l \in U$ **and** $h \in U$ **and** $h' \in U$
shows $(\{l\leq_a..\leq_a h\} \subseteq \{..\leq_a h'\}) = (\neg\ l \leq_a h \vee h \leq_a h')$
is *preorder.Icc-subset-Iic-iff⟨proof⟩*

tts-lemma *atLeast-subset-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(\{x\leq_a..\} \subseteq \{y\leq_a..\}) = (y \leq_a x)$
is *preorder.atLeast-subset-iff⟨proof⟩*

tts-lemma *Icc-subset-Ici-iff*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$
shows $(\{l\leq_a..\leq_a h\} \subseteq \{l'\leq_a..\}) = (\neg\ l \leq_a h \vee l' \leq_a l)$
is *preorder.Icc-subset-Ici-iff⟨proof⟩*

tts-lemma *atLeastatMost-subset-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a\leq_a..\leq_a b\} \subseteq \{c\leq_a..\leq_a d\}) = (\neg\ a \leq_a b \vee b \leq_a d \wedge c \leq_a a)$
is *preorder.atLeastatMost-subset-iff⟨proof⟩*

tts-lemma *atLeastatMost-psubset-iff*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$ **and** $d \in U$
shows $(\{a\leq_a..\leq_a b\} \subset \{c\leq_a..\leq_a d\}) =$
$(c \leq_a d \wedge (\neg\ a \leq_a b \vee c \leq_a a \wedge b \leq_a d \wedge (c <_a a \vee b <_a d)))$
is *preorder.atLeastatMost-psubset-iff⟨proof⟩*

tts-lemma *bdd-above-empty*:
assumes $U \neq \{\}$
shows *bdd-above* $\{\}$
is *preorder.bdd-above-empty⟨proof⟩*

tts-lemma *bdd-above-Iic*:
assumes $b \in U$
shows *bdd-above* $\{..\leq_a b\}$
is *preorder.bdd-above-Iic⟨proof⟩*

tts-lemma *bdd-above-Iio*:
assumes $b \in U$
shows *bdd-above* $\{..<_a b\}$
is *preorder.bdd-above-Iio⟨proof⟩*

tts-lemma *bdd-below-empty*:
assumes $U \neq \{\}$
shows *bdd-below* $\{\}$
is *preorder.bdd-below-empty*⟨*proof*⟩

tts-lemma *bdd-above-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a{\leq_a}..{\leq_a}b\}$
is *preorder.bdd-above-Icc*⟨*proof*⟩

tts-lemma *bdd-above-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a{\leq_a}..{<_a}b\}$
is *preorder.bdd-above-Ico*⟨*proof*⟩

tts-lemma *bdd-above-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a{<_a}..{\leq_a}b\}$
is *preorder.bdd-above-Ioc*⟨*proof*⟩

tts-lemma *bdd-above-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-above* $\{a{<_a}..{<_a}b\}$
is *preorder.bdd-above-Ioo*⟨*proof*⟩

tts-lemma *bdd-above-Int*1:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-above* A
shows *bdd-above* $(A \cap B)$
is *preorder.bdd-above-Int*1⟨*proof*⟩

tts-lemma *bdd-above-Int*2:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-above* B
shows *bdd-above* $(A \cap B)$
is *preorder.bdd-above-Int*2⟨*proof*⟩

tts-lemma *bdd-below-Icc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a{\leq_a}..{\leq_a}b\}$
is *preorder.bdd-below-Icc*⟨*proof*⟩

tts-lemma *bdd-below-Ico*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a{\leq_a}..{<_a}b\}$
is *preorder.bdd-below-Ico*⟨*proof*⟩

tts-lemma *bdd-below-Ioc*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a{<_a}..{\leq_a}b\}$
is *preorder.bdd-below-Ioc*⟨*proof*⟩

tts-lemma *bdd-below-Ioo*:
assumes $a \in U$ **and** $b \in U$
shows *bdd-below* $\{a{<_a}..{<_a}b\}$
is *preorder.bdd-below-Ioo*⟨*proof*⟩

tts-lemma *bdd-below-Ici*:
assumes $a \in U$
shows *bdd-below* $\{a{\leq_a}..\}$

is *preorder.bdd-below-Ici*⟨*proof*⟩

tts-lemma *bdd-below-Ioi*:
assumes $a \in U$
shows *bdd-below* $\{a<_a..\}$
is *preorder.bdd-below-Ioi*⟨*proof*⟩

tts-lemma *bdd-above-mono*:
assumes $B \subseteq U$ **and** *bdd-above* B **and** $A \subseteq B$
shows *bdd-above* A
is *preorder.bdd-above-mono*⟨*proof*⟩

tts-lemma *bdd-aboveI*:
assumes $A \subseteq U$ **and** $M \in U$ **and** $\bigwedge x.\ [\![x \in U;\ x \in A]\!] \Longrightarrow x \leq_a M$
shows *bdd-above* A
is *preorder.bdd-aboveI*⟨*proof*⟩

tts-lemma *bdd-aboveI2*:
assumes *range* $f \subseteq U$ **and** $M \in U$ **and** $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq_a M$
shows *bdd-above* $(f\ `\ A)$
is *preorder.bdd-aboveI2*⟨*proof*⟩

tts-lemma *bdd-below-Int1*:
assumes $A \subseteq U$ **and** $B \subseteq U$ **and** *bdd-below* A
shows *bdd-below* $(A \cap B)$
is *preorder.bdd-below-Int1*⟨*proof*⟩

tts-lemma *bdd-below-Int2*:
assumes $B \subseteq U$ **and** $A \subseteq U$ **and** *bdd-below* B
shows *bdd-below* $(A \cap B)$
is *preorder.bdd-below-Int2*⟨*proof*⟩

tts-lemma *bdd-belowI*:
assumes $A \subseteq U$ **and** $m \in U$ **and** $\bigwedge x.\ [\![x \in U;\ x \in A]\!] \Longrightarrow m \leq_a x$
shows *bdd-below* A
is *preorder.bdd-belowI*⟨*proof*⟩

tts-lemma *bdd-below-mono*:
assumes $B \subseteq U$ **and** *bdd-below* B **and** $A \subseteq B$
shows *bdd-below* A
is *preorder.bdd-below-mono*⟨*proof*⟩

tts-lemma *bdd-belowI2*:
assumes $m \in U$ **and** *range* $f \subseteq U$ **and** $\bigwedge x.\ x \in A \Longrightarrow m \leq_a f\ x$
shows *bdd-below* $(f\ `\ A)$
is *preorder.bdd-belowI2*[**where** $'b='d$]⟨*proof*⟩

end

end

5.5.4 Partial orders

locale *order-ow* = *preorder-ow U le ls*
for $U :: 'a\ set$ **and** *le ls* +
assumes *antisym*: $x \in U \Longrightarrow y \in U \Longrightarrow le\ x\ y \Longrightarrow le\ y\ x \Longrightarrow x = y$

locale *order-dual-ow* = *order-ow U le ls*

for $U :: 'a\ set$ **and** $le\ ls$
begin

sublocale *preorder-dual-ow* ⟨*proof*⟩

sublocale *dual*: *order-ow* $U\ ge\ gt$
⟨*proof*⟩

end

locale *ord-order-ow* =
ord-preorder-ow $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$ + ord_b: *order-ow* $U_b\ le_b\ ls_b$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$

locale *ord-order-dual-ow* = *ord-order-ow* $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$
begin

sublocale *ord-preorder-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *ord-order-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩
sublocale *dual-ord*: *ord-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
⟨*proof*⟩
sublocale *dual-dual*: *ord-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩

end

locale *preorder-order-ow* =
ord-order-ow $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$ + ord_a: *preorder-ow* $U_a\ le_a\ ls_a$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$
begin

sublocale *preorder-pair-ow* ⟨*proof*⟩

end

locale *preorder-order-dual-ow* = *preorder-order-ow* $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$
begin

sublocale *ord-order-dual-ow* ⟨*proof*⟩
sublocale *preorder-pair-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-order-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *preorder-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$› ⟨*proof*⟩
sublocale *dual-dual*: *preorder-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

locale *order-pair-ow* =
preorder-order-ow $U_a\ le_a\ ls_a\ U_b\ le_b\ ls_b$ + ord_a: *order-ow* $U_a\ le_a\ ls_a$
for $U_a :: 'a\ set$ **and** $le_a\ ls_a$ **and** $U_b :: 'b\ set$ **and** $le_b\ ls_b$
begin

sublocale *rev*: *order-pair-ow* $U_b\ le_b\ ls_b\ U_a\ le_a\ ls_a$ ⟨*proof*⟩

end

locale *order-pair-dual-ow* = *order-pair-ow* U_a le_a ls_a U_b le_b ls_b
  for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *preorder-order-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *order-pair-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *order-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
  ⟨*proof*⟩
sublocale *dual-dual*: *order-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

Transfer rules

lemma *order-ow*[*ud-with*]: *order* = *order-ow UNIV*
  ⟨*proof*⟩

lemma *ord-order-ow*[*ud-with*]: *ord-order* = *ord-order-ow UNIV*
  ⟨*proof*⟩

lemma *preorder-order-ow*[*ud-with*]:
  preorder-order =
    (λle_a ls_a le_b ls_b. *preorder-order-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
  ⟨*proof*⟩

lemma *order-pair-ow*[*ud-with*]:
  order-pair = (λle_a ls_a le_b ls_b. *order-pair-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
  ⟨*proof*⟩

context
  includes *lifting-syntax*
begin

lemma *order-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
      order-ow order-ow
  ⟨*proof*⟩

lemma *ord-order-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (
      rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
      (=)
) *ord-order-ow ord-order-ow*
  ⟨*proof*⟩

lemma *preorder-order-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*
  shows
    (
      rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
      rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
      (=)
) *preorder-order-ow preorder-order-ow*
  ⟨*proof*⟩

lemma *order-pair-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
(
rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
(=)
) *order-pair-ow order-pair-ow*
⟨*proof*⟩

end

Relativization

context *order-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
and (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *auto*
begin

tts-lemma *atLeastAtMost-singleton*:
assumes $a \in U$
shows $\{a\leq_a..\leq_a a\} = \{a\}$
is *order.atLeastAtMost-singleton*⟨*proof*⟩

tts-lemma *less-imp-not-eq*:
assumes $y \in U$ **and** $x <_a y$
shows $(x = y) = False$
is *order.less-imp-not-eq*⟨*proof*⟩

tts-lemma *less-imp-not-eq2*:
assumes $y \in U$ **and** $x <_a y$
shows $(y = x) = False$
is *order.less-imp-not-eq2*⟨*proof*⟩

tts-lemma *eq-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(x = y) = (x \leq_a y \wedge y \leq_a x)$
is *order.eq-iff*⟨*proof*⟩

tts-lemma *le-less*:
assumes $x \in U$ **and** $y \in U$
shows $(x \leq_a y) = (x <_a y \vee x = y)$
is *order.le-less*⟨*proof*⟩

tts-lemma *min-absorb2*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_a x$
shows *min x y* = *y*

is *order.min-absorb2⟨proof⟩*

tts-lemma *less-le*:
assumes $x \in U$ **and** $y \in U$
shows $(x <_a y) = (x \leq_a y \wedge x \neq y)$
is *order.less-le⟨proof⟩*

tts-lemma *le-imp-less-or-eq*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_a y$
shows $x <_a y \vee x = y$
is *order.le-imp-less-or-eq⟨proof⟩*

tts-lemma *antisym-conv*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_a x$
shows $(x \leq_a y) = (x = y)$
is *order.antisym-conv⟨proof⟩*

tts-lemma *le-neq-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $a \leq_a b$ **and** $a \neq b$
shows $a <_a b$
is *order.le-neq-trans⟨proof⟩*

tts-lemma *neq-le-trans*:
assumes $a \in U$ **and** $b \in U$ **and** $a \neq b$ **and** $a \leq_a b$
shows $a <_a b$
is *order.neq-le-trans⟨proof⟩*

tts-lemma *atLeastAtMost-singleton′*:
assumes $b \in U$ **and** $a = b$
shows $\{a{\leq_a}..{\leq_a}b\} = \{a\}$
is *order.atLeastAtMost-singleton′⟨proof⟩*

tts-lemma *atLeastLessThan-eq-atLeastAtMost-diff*:
assumes $a \in U$ **and** $b \in U$
shows $\{a{\leq_a}..{<_a}b\} = \{a{\leq_a}..{\leq_a}b\} - \{b\}$
is *order.atLeastLessThan-eq-atLeastAtMost-diff⟨proof⟩*

tts-lemma *greaterThanAtMost-eq-atLeastAtMost-diff*:
assumes $a \in U$ **and** $b \in U$
shows $\{a{<_a}..{\leq_a}b\} = \{a{\leq_a}..{\leq_a}b\} - \{a\}$
is *order.greaterThanAtMost-eq-atLeastAtMost-diff⟨proof⟩*

tts-lemma *atMost-Int-atLeast*:
assumes $n \in U$
shows $\{..{\leq_a}n\} \cap \{n{\leq_a}..\} = \{n\}$
is *order.atMost-Int-atLeast⟨proof⟩*

tts-lemma *atLeast-eq-iff*:
assumes $x \in U$ **and** $y \in U$
shows $(\{x{\leq_a}..\} = \{y{\leq_a}..\}) = (x = y)$
is *order.atLeast-eq-iff⟨proof⟩*

tts-lemma *Least-equality*:
assumes $x \in U$ **and** $P\ x$
and $\bigwedge y.\ [\![y \in U;\ P\ y]\!] \Longrightarrow x \leq_a y$
shows *Least P = Some x*
is *order.Least-equality⟨proof⟩*

tts-lemma *Icc-eq-Icc*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $(\{l \leq_a .. \leq_a h\} = \{l' \leq_a .. \leq_a h'\}) =$
$(h = h' \wedge l = l' \vee \neg\ l' \leq_a h' \wedge \neg\ l \leq_a h)$
is *order.Icc-eq-Icc⟨proof⟩*

tts-lemma *LeastI2-order*:
assumes $x \in U$
and $P\ x$
and $\bigwedge y.\ [\![y \in U;\ P\ y]\!] \Longrightarrow x \leq_a y$
and $\bigwedge x.\ [\![x \in U;\ P\ x;\ \forall\, y{\in}U.\ P\ y \longrightarrow x \leq_a y]\!] \Longrightarrow Q\ x$
and $\bigwedge z.\ [\![z \in U;\ Least\ P = Some\ z;\ Q\ z]\!] \Longrightarrow thesis$
shows *thesis*
is *order.LeastI2-order⟨proof⟩*

tts-lemma *mono-image-least*:
assumes $\forall\, x{\in}U.\ f\ x \in U$
and $m \in U$
and $n \in U$
and $m' \in U$
and $n' \in U$
and *on U with* $(\leq_a)$ $(\leq_a)$: «*mono*» f
and $f\ `\ \{m \leq_a .. <_a n\} = \{m' \leq_a .. <_a n'\}$
and $m <_a n$
shows $f\ m = m'$
is *order.mono-image-least⟨proof⟩*

tts-lemma *antisym-conv1*:
assumes $x \in U$ **and** $y \in U$ **and** $\neg\ x <_a y$
shows $(x \leq_a y) = (x = y)$
is *order.antisym-conv1⟨proof⟩*

tts-lemma *antisym-conv2*:
assumes $x \in U$ **and** $y \in U$ **and** $x \leq_a y$
shows $(\neg\ x <_a y) = (x = y)$
is *order.antisym-conv2⟨proof⟩*

tts-lemma *leD*:
assumes $y \in U$ **and** $x \in U$ **and** $y \leq_a x$
shows $\neg\ x <_a y$
is *order.leD⟨proof⟩*

end

tts-context
tts: (*?'a* **to** U)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating ‹*?A* ≠ {}› **through** *auto*
begin

tts-lemma *atLeastAtMost-singleton-iff*:
assumes $a \in U$
and $b \in U$
and $c \in U$
shows $(\{a \leq_a .. \leq_a b\} = \{c\}) = (a = b \wedge b = c)$
is *order.atLeastAtMost-singleton-iff⟨proof⟩*

end

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
and (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
rewriting *ctr-simps*
substituting *order-ow-axioms*
eliminating through *auto*
begin

tts-lemma *Least-ex1*:
assumes $z \in U$
and $\exists!x.\ x \in U \wedge P\ x \wedge (\forall\, y{\in}U.\ P\ y \longrightarrow x \leq_a y)$
and $\bigwedge x.\ [\![x \in U;\ Least\ P = Some\ x;\ P\ x;\ P\ z \Longrightarrow x \leq_a z]\!] \Longrightarrow thesis$
shows *thesis*
is *order.Least-ex1*⟨*proof*⟩

end

end

context *order-pair-ow*
begin

interpretation *ord-pair-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** U_a) **and** (*?'b* **to** U_b)
sbterms: (‹$?ls_a$::*?'a* ⇒ *?'a* ⇒ *bool*› **to** ls_a)
and (‹$?le_a$::*?'a* ⇒ *?'a* ⇒ *bool*› **to** le_a)
and (‹$?ls_b$::*?'b* ⇒ *?'b* ⇒ *bool*› **to** ls_b)
and (‹$?le_b$::*?'b* ⇒ *?'b* ⇒ *bool*› **to** le_b)
rewriting *ctr-simps*
substituting *order-pair-ow-axioms*
eliminating through (*auto simp*: *mono-def bdd-def*)
begin

tts-lemma *strict-mono-mono*:
assumes $\forall\, x{\in}U_a.\ f\ x \in U_b$ **and** $strict\text{-}mono_{ab}\ f$
shows $mono_{ab}\ f$
is *order-pair.strict-mono-mono*⟨*proof*⟩

tts-lemma *bdd-above-image-mono*:
assumes $\forall\, x{\in}U_a.\ f\ x \in U_b$ **and** $A \subseteq U_a$ **and** $mono_{ab}\ f$ **and** $ord_a.bdd\text{-}above\ A$
shows $ord_b.bdd\text{-}above\ (f\ `\ A)$
is *order-pair.bdd-above-image-mono*⟨*proof*⟩

tts-lemma *bdd-below-image-mono*:
assumes $\forall\, x{\in}U_a.\ f\ x \in U_b$ **and** $A \subseteq U_a$ **and** $mono_{ab}\ f$ **and** $ord_a.bdd\text{-}below\ A$
shows $ord_b.bdd\text{-}below\ (f\ `\ A)$
is *order-pair.bdd-below-image-mono*⟨*proof*⟩

tts-lemma *bdd-below-image-antimono*:
assumes $\forall\, x{\in}U_a.\ f\ x \in U_b$
and $A \subseteq U_a$
and $antimono_{ab}\ f$
and $ord_a.bdd\text{-}above\ A$

shows ord_b.*bdd-below* (f ‘ A)
is *order-pair.bdd-below-image-antimono*⟨*proof*⟩

tts-lemma *bdd-above-image-antimono*:
assumes $\forall\, x{\in}U_a.\ f\ x \in U_b$
and $A \subseteq U_a$
and $antimono_{a\,b}\ f$
and ord_a.*bdd-below* A
shows ord_b.*bdd-above* (f ‘ A)
is *order-pair.bdd-above-image-antimono*⟨*proof*⟩

end

end

5.5.5 Dense orders

Definitions and common properties

locale *dense-order-ow* = *order-ow U le ls*
for U :: $'a$ *set* **and** *le ls* +
assumes *dense*: $[\![\ x \in U;\ y \in U;\ ls\ x\ y\ ]\!] \Longrightarrow (\exists\, z{\in}U.\ ls\ x\ z \wedge ls\ z\ y)$

locale *dense-order-dual-ow* = *dense-order-ow U le ls*
for U :: $'a$ *set* **and** *le ls*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

sublocale *order-dual-ow* ⟨*proof*⟩

sublocale *dual*: *dense-order-ow U ge gt*
⟨*proof*⟩

end

locale *ord-dense-order-ow* =
ord-order-ow U_a le_a ls_a U_b le_b ls_b + ord_b: *dense-order-ow* U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b

locale *ord-dense-order-dual-ow* = *ord-dense-order-ow* U_a le_a ls_a U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *ord-order-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *ord-dense-order-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩
sublocale *dual-ord*: *ord-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
⟨*proof*⟩
sublocale *dual-dual*: *ord-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩

end

locale *preorder-dense-order-ow* =
ord-dense-order-ow U_a le_a ls_a U_b le_b ls_b + ord_a: *preorder-ow* U_a le_a ls_a
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *preorder-order-ow* ⟨*proof*⟩

end

locale *preorder-dense-order-dual-ow* =
preorder-dense-order-ow U_a le_a ls_a U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *ord-dense-order-dual-ow* ⟨*proof*⟩
sublocale *preorder-order-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *preorder-dense-order-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩
sublocale *dual-ord*: *preorder-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
⟨*proof*⟩
sublocale *dual-dual*: *preorder-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$›
⟨*proof*⟩

end

locale *order-dense-order-ow* =
preorder-dense-order-ow U_a le_a ls_a U_b le_b ls_b + ord_a: *order-ow* U_a le_a ls_a
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *order-pair-ow* ⟨*proof*⟩

end

locale *order-dense-order-dual-ow* = *order-dense-order-ow* U_a le_a ls_a U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *preorder-dense-order-dual-ow* ⟨*proof*⟩
sublocale *order-pair-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *order-dense-order-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *order-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$› ⟨*proof*⟩
sublocale *dual-dual*: *order-dense-order-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

locale *dense-order-pair-ow* =
order-dense-order-ow U_a le_a ls_a U_b le_b ls_b + ord_a: *dense-order-ow* U_a le_a ls_a
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b

locale *dense-order-pair-dual-ow* = *dense-order-pair-ow* U_a le_a ls_a U_b le_b ls_b
for U_a :: $'a$ *set* **and** le_a ls_a **and** U_b :: $'b$ *set* **and** le_b ls_b
begin

sublocale *order-dense-order-dual-ow* ⟨*proof*⟩
sublocale *ord-dual*: *dense-order-pair-ow* U_a ‹$(\leq_a)$› ‹$(<_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩
sublocale *dual-ord*: *dense-order-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\leq_b)$› ‹$(<_b)$›
⟨*proof*⟩
sublocale *dual-dual*: *dense-order-pair-ow* U_a ‹$(\geq_a)$› ‹$(>_a)$› U_b ‹$(\geq_b)$› ‹$(>_b)$› ⟨*proof*⟩

end

Transfer rules

lemma *dense-order-ow*[*ud-with*]: *dense-order = dense-order-ow UNIV*
⟨*proof*⟩

lemma *ord-dense-order-ow*[*ud-with*]: *ord-dense-order = ord-dense-order-ow UNIV*
⟨*proof*⟩

lemma *preorder-dense-order-ow*[*ud-with*]:
preorder-dense-order =
(λle_a ls_a le_b ls_b. *preorder-dense-order-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
⟨*proof*⟩

lemma *order-dense-order-ow*[*ud-with*]:
order-dense-order =
(λle_a ls_a le_b ls_b. *order-dense-order-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
⟨*proof*⟩

lemma *dense-order-pair-ow*[*ud-with*]:
dense-order-pair =
(λle_a ls_a le_b ls_b. *dense-order-pair-ow UNIV* le_a ls_a *UNIV* le_b ls_b)
⟨*proof*⟩

context
includes *lifting-syntax*
begin

lemma *desne-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
dense-order-ow dense-order-ow
⟨*proof*⟩

lemma *ord-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(
rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
(=)
) *ord-dense-order-ow ord-dense-order-ow*
⟨*proof*⟩

lemma *preorder-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*
shows
(
rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
(=)
) *preorder-dense-order-ow preorder-dense-order-ow*
⟨*proof*⟩

lemma *order-dense-order-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
(

rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
(=)
) *order-dense-order-ow order-dense-order-ow*
⟨*proof*⟩

lemma *dense-order-pair-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
(
rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===>
rel-set B ===> (*B* ===> *B* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===>
(=)
) *dense-order-pair-ow dense-order-pair-ow*
⟨*proof*⟩

end

5.5.6 (Unique) top and bottom elements

locale *extremum-ow* =
fixes *U* :: *'a set* **and** *extremum*
assumes *extremum-closed*[*simp*]: *extremum* ∈ *U*

locale *bot-ow* = *extremum-ow U bot* **for** *U* :: *'a set* **and** *bot*
begin

notation *bot* (‹⊥›)

end

locale *top-ow* = *extremum-ow U top* **for** *U* :: *'a set* **and** *top*
begin

notation *top* (‹⊤›)

end

locale *ord-extremum-ow* = *ord-ow U le ls* + *extremum-ow U extremum*
for *U* :: *'a set* **and** *le ls extremum*

locale *order-extremum-ow* = *ord-extremum-ow U le ls extremum* + *order-ow U le ls*
for *U* :: *'a set* **and** *le ls extremum* +
assumes *extremum*[*simp*]: *a* ∈ *U* ⟹ *le a extremum*

locale *order-bot-ow* =
order-dual-ow U le ls + *dual*: *order-extremum-ow U ge gt bot* + *bot-ow U bot*
for *U* :: *'a set* **and** *le ls bot*

locale *order-top* =
order-dual-ow U le ls + *order-extremum-ow U le ls top* + *top-ow U top*
for *U* :: *'a set* **and** *le ls top*

Transfer rules

lemma *order-extremum-ow*[*ud-with*]: *order-extremum* = *order-extremum-ow UNIV*
⟨*proof*⟩

context
  includes *lifting-syntax*
begin

lemma *extremum-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A*
  shows (*rel-set A* ===> *A* ===> (=)) *extremum-ow extremum-ow*
  ⟨*proof*⟩

lemma *ord-extremum-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A*
  shows (*rel-set A* ===> *A* ===> (=)) *ord-extremum-ow ord-extremum-ow*
  ⟨*proof*⟩

lemma *order-extremum-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (
      rel-set A ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> *A* ===>
      (=)
) *order-extremum-ow order-extremum-ow*
  ⟨*proof*⟩

end

Relativization

context *order-extremum-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context
  tts: (*?'a* **to** *U*)
  sbterms: (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
    and (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
  rewriting *ctr-simps*
  substituting *order-extremum-ow-axioms*
  eliminating through *force*
begin

tts-lemma *extremum-strict*:
  assumes *a* ∈ *U*
  shows ¬ *extremum* <$_a$ *a*
    is *order-extremum.extremum-strict*⟨*proof*⟩

tts-lemma *bdd-above-top*:
  assumes *A* ⊆ *U*
  shows *bdd-above A*
    is *order-extremum.bdd-above-top*⟨*proof*⟩

tts-lemma *min-top*:
  assumes *x* ∈ *U*
  shows *min extremum x* = *x*
  is *order-extremum.min-top*⟨*proof*⟩

tts-lemma *min-top*2:

assumes $x \in U$
shows *min x extremum* = x
is *order-extremum.min-top2*⟨*proof*⟩

tts-lemma *extremum-unique*:
assumes $a \in U$
shows $(extremum \leq_a a) = (a = extremum)$
is *order-extremum.extremum-unique*⟨*proof*⟩

tts-lemma *not-eq-extremum*:
assumes $a \in U$
shows $(a \neq extremum) = (a <_a extremum)$
is *order-extremum.not-eq-extremum*⟨*proof*⟩

tts-lemma *extremum-uniqueI*:
assumes $a \in U$ **and** $extremum \leq_a a$
shows $a = extremum$
is *order-extremum.extremum-uniqueI*⟨*proof*⟩

tts-lemma *max-top*:
assumes $x \in U$
shows *max extremum x* = *extremum*
is *order-extremum.max-top*⟨*proof*⟩

tts-lemma *max-top2*:
assumes $x \in U$
shows *max x extremum* = *extremum*
is *order-extremum.max-top2*⟨*proof*⟩

tts-lemma *atMost-eq-UNIV-iff*:
assumes $x \in U$
shows $(\{..\leq_a x\} = U) = (x = extremum)$
is *order-extremum.atMost-eq-UNIV-iff*⟨*proof*⟩

end

end

5.5.7 Absence of top or bottom elements

locale *no-extremum-ow* = *order-ow U le ls* **for** $U :: 'a\ set$ **and** *le ls* +
assumes *gt-ex*: $x \in U \Longrightarrow \exists\, y \in U.\ ls\ x\ y$

locale *no-top-ow* = *order-dual-ow U le ls* + *no-extremum-ow U le ls*
for $U :: 'a\ set$ **and** *le ls*

locale *no-bot-ow* = *order-dual-ow U le ls* + *dual*: *no-extremum-ow U ge gt*
for $U :: 'a\ set$ **and** *le ls*

Transfer rules

lemma *no-extremum-ow*[*ud-with*]: *no-extremum* = *no-extremum-ow UNIV*
⟨*proof*⟩

context
includes *lifting-syntax*
begin

lemma *no-extremum-ow-axioms-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (=))
no-extremum-ow-axioms no-extremum-ow-axioms
⟨*proof*⟩

lemma *no-extremum-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))
no-extremum-ow no-extremum-ow
⟨*proof*⟩

end

Relativization

lemma *right-total-UNIV-transfer'*[*transfer-rule*]:
assumes *right-total A* **and** *Domainp A* = (λ*x*. *x* ∈ *U*)
shows *rel-set A U UNIV*
⟨*proof*⟩

context *no-extremum-ow*
begin

interpretation *ord-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?le*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *le*)
and (‹*?ls*::*?'a* ⇒ *?'a* ⇒ *bool*› **to** *ls*)
rewriting *ctr-simps*
substituting *no-extremum-ow-axioms*
eliminating through *force*
begin

tts-lemma *not-UNIV-eq-Iic*:
assumes $h' \in U$
shows $U \neq \{..\leq_a h'\}$
is *no-extremum.not-UNIV-eq-Iic*⟨*proof*⟩

tts-lemma *not-Iic-eq-UNIV*:
assumes $h' \in U$
shows $\{..\leq_a h'\} \neq U$
is *no-extremum.not-Iic-eq-UNIV*⟨*proof*⟩

tts-lemma *not-UNIV-le-Iic*:
assumes $h \in U$
shows $\neg\ U \subseteq \{..\leq_a h\}$
is *no-extremum.not-UNIV-le-Iic*⟨*proof*⟩

tts-lemma *not-UNIV-eq-Icc*:
assumes $l' \in U$ **and** $h' \in U$
shows $U \neq \{l'\leq_a..\leq_a h'\}$
is *no-extremum.not-UNIV-eq-Icc*⟨*proof*⟩

tts-lemma *not-Icc-eq-UNIV*:

assumes $l' \in U$ **and** $h' \in U$
shows $\{l' \leq_a .. \leq_a h'\} \neq U$
is *no-extremum.not-Icc-eq-UNIV*⟨*proof*⟩

tts-lemma *not-UNIV-le-Icc*:
assumes $l \in U$ **and** $h \in U$
shows $\neg\ U \subseteq \{l \leq_a .. \leq_a h\}$
is *no-extremum.not-UNIV-le-Icc*⟨*proof*⟩

tts-lemma *greaterThan-non-empty*:
assumes $x \in U$
shows $\{x <_a ..\} \neq \{\}$
is *no-extremum.greaterThan-non-empty*⟨*proof*⟩

tts-lemma *not-Iic-eq-Ici*:
assumes $h \in U$ **and** $l' \in U$
shows $\{.. \leq_a h\} \neq \{l' \leq_a ..\}$
is *no-extremum.not-Iic-eq-Ici*⟨*proof*⟩

tts-lemma *not-Ici-eq-Iic*:
assumes $l' \in U$ **and** $h \in U$
shows $\{l' \leq_a ..\} \neq \{.. \leq_a h\}$
is *no-extremum.not-Ici-eq-Iic*⟨*proof*⟩

tts-lemma *not-Ici-le-Iic*:
assumes $l \in U$ **and** $h' \in U$
shows $\neg\ \{l \leq_a ..\} \subseteq \{.. \leq_a h'\}$
is *no-extremum.not-Ici-le-Iic*⟨*proof*⟩

tts-lemma *not-Icc-eq-Ici*:
assumes $l \in U$ **and** $h \in U$ **and** $l' \in U$
shows $\{l \leq_a .. \leq_a h\} \neq \{l' \leq_a ..\}$
is *no-extremum.not-Icc-eq-Ici*⟨*proof*⟩

tts-lemma *not-Ici-eq-Icc*:
assumes $l' \in U$ **and** $l \in U$ **and** $h \in U$
shows $\{l' \leq_a ..\} \neq \{l \leq_a .. \leq_a h\}$
is *no-extremum.not-Ici-eq-Icc*⟨*proof*⟩

tts-lemma *not-Ici-le-Icc*:
assumes $l \in U$ **and** $l' \in U$ **and** $h' \in U$
shows $\neg\ \{l \leq_a ..\} \subseteq \{l' \leq_a .. \leq_a h'\}$
is *no-extremum.not-Ici-le-Icc*⟨*proof*⟩

end

end

declare *right-total-UNIV-transfer'*[*transfer-rule del*]

5.6 Abstract semigroups on types

5.6.1 Background

The results presented in this section were ported (with amendments and additions) from the theory *Groups* in the main library of Isabelle/HOL.

5.6.2 Preliminaries

named-theorems *tts-ac-simps assoc. and comm. simplification rules*
and *tts-algebra-simps algebra simplification rules*
and *tts-field-simps algebra simplification rules for fields*

5.6.3 Binary operations

Abstract operation.

locale *binary-op* =
fixes $f :: 'a \Rightarrow 'a \Rightarrow 'a$

locale *binary-op-syntax* = *binary-op f* **for** $f :: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation f (**infixl** $\langle \oplus_a \rangle$ 65)

end

Concrete syntax.

locale *plus* = *binary-op plus* **for** *plus* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *plus* (**infixl** $\langle +_a \rangle$ 65)

end

locale *minus* = *binary-op minus* **for** *minus* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *minus* (**infixl** $\langle -_a \rangle$ 65)

end

locale *times* = *binary-op times* **for** *times* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *times* (**infixl** $\langle *_a \rangle$ 70)

end

locale *divide* = *binary-op divide* **for** *divide* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

notation *divide* (**infixl** $\langle '/_a \rangle$ 70)

end

Pairs.

locale *binary-op-pair* = alg_a: *binary-op* f_a + alg_b: *binary-op* f_b

for $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$ **and** $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$

locale *binary-op-pair-syntax* = *binary-op-pair* f_a f_b
for $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$ **and** $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$
begin

notation f_a (**infixl** ‹$\oplus_a$› 65)
notation f_b (**infixl** ‹$\oplus_b$› 65)

end

5.6.4 Simple semigroups

Definitions

Abstract semigroups.

locale *semigroup* = *binary-op* f **for** $f :: 'a \Rightarrow 'a \Rightarrow 'a$ +
assumes *assoc*[*tts-ac-simps*, *tts-algebra-simps*]: $f\ (f\ a\ b)\ c = f\ a\ (f\ b\ c)$

locale *semigroup-syntax* = *binary-op-syntax* f **for** $f :: 'a \Rightarrow 'a \Rightarrow 'a$

Concrete syntax.

locale *semigroup-add* = *semigroup plus* **for** *plus* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *plus plus* ⟨*proof*⟩

end

locale *semigroup-mult* = *semigroup times* **for** *times* $:: 'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *times times* ⟨*proof*⟩

end

Pairs.

locale *semigroup-pair* = *alg*$_a$: *semigroup* f_a + *alg*$_b$: *semigroup* f_b
for $f_a :: 'a \Rightarrow 'a \Rightarrow 'a$ **and** $f_b :: 'b \Rightarrow 'b \Rightarrow 'b$
begin

sublocale *binary-op-pair* f_a f_b ⟨*proof*⟩
sublocale *rev*: *semigroup-pair* f_b f_a ⟨*proof*⟩

end

locale *semigroup-pair-syntax* = *binary-op-pair-syntax*

5.6.5 Commutative semigroups

Definitions

Abstract commutative semigroup.

locale *comm-semigroup* = *semigroup* f **for** $f :: 'a \Rightarrow 'a \Rightarrow 'a$ +
assumes *commute*[*tts-ac-simps*, *tts-algebra-simps*]: $f\ a\ b = f\ b\ a$

locale *comm-semigroup-syntax* = *semigroup-syntax*

Concrete syntax.

locale *comm-semigroup-add* = *comm-semigroup plus* **for** *plus* :: $'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *semigroup-add plus* ⟨*proof*⟩

end

locale *comm-semigroup-mult* = *comm-semigroup times* **for** *times* :: $'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *semigroup-mult times* ⟨*proof*⟩

end

Pairs.

locale *comm-semigroup-pair* = alg_a: *comm-semigroup* f_a + alg_b: *comm-semigroup* f_b
  for f_a :: $'a \Rightarrow 'a \Rightarrow 'a$ **and** f_b :: $'b \Rightarrow 'b \Rightarrow 'b$
begin

sublocale *semigroup-pair* f_a f_b ⟨*proof*⟩
sublocale *rev*: *comm-semigroup-pair* f_b f_a ⟨*proof*⟩

end

locale *comm-semigroup-pair-syntax* = *semigroup-pair-syntax*

Results

context *comm-semigroup*
begin

interpretation *comm-semigroup-syntax f* ⟨*proof*⟩

lemma *left-commute*[*tts-ac-simps*, *tts-algebra-simps*, *field-simps*]:
  $b \oplus_a (a \oplus_a c) = a \oplus_a (b \oplus_a c)$
⟨*proof*⟩

end

5.6.6 Cancellative semigroups

Definitions

Abstract cancellative semigroup.

locale *cancel-semigroup* = *semigroup f* **for** *f* :: $'a \Rightarrow 'a \Rightarrow 'a$ +
  assumes *add-left-imp-eq*: $f\ a\ b = f\ a\ c \Longrightarrow b = c$
  assumes *add-right-imp-eq*: $f\ b\ a = f\ c\ a \Longrightarrow b = c$

locale *cancel-semigroup-syntax* = *semigroup-syntax f* **for** *f* :: $'a \Rightarrow 'a \Rightarrow 'a$

Concrete syntax.

locale *cancel-semigroup-add* = *cancel-semigroup plus*
  for *plus* :: $'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *semigroup-add plus* ⟨*proof*⟩

end

locale *cancel-semigroup-mult* = *cancel-semigroup times*
  for *times* :: $'a \Rightarrow 'a \Rightarrow 'a$
begin

sublocale *semigroup-mult times* ⟨*proof*⟩

end

Pairs.

locale *cancel-semigroup-pair* =
  alg_a: *cancel-semigroup* f_a + alg_b: *cancel-semigroup* f_b
  for f_a :: $'a \Rightarrow 'a \Rightarrow 'a$ **and** f_b :: $'b \Rightarrow 'b \Rightarrow 'b$
begin

sublocale *semigroup-pair* f_a f_b ⟨*proof*⟩
sublocale *rev*: *cancel-semigroup-pair* f_b f_a ⟨*proof*⟩

end

locale *cancel-semigroup-pair-syntax* = *semigroup-pair-syntax* f_a f_b
  for f_a :: $'a \Rightarrow 'a \Rightarrow 'a$ **and** f_b :: $'b \Rightarrow 'b \Rightarrow 'b$

Results

context *cancel-semigroup*
begin

interpretation *cancel-semigroup-syntax f* ⟨*proof*⟩

lemma *add-left-cancel*[*simp*]: $a \oplus_a b = a \oplus_a c \longleftrightarrow b = c$
  ⟨*proof*⟩

lemma *add-right-cancel*[*simp*]: $b \oplus_a a = c \oplus_a a \longleftrightarrow b = c$
  ⟨*proof*⟩

lemma *inj-on-add*[*simp*]: *inj-on* $((\oplus_a)\ a)\ A$ ⟨*proof*⟩

lemma *inj-on-add'*[*simp*]: *inj-on* $(\lambda b.\ b \oplus_a a)\ A$ ⟨*proof*⟩

lemma *bij-betw-add*[*simp*]: *bij-betw* $((\oplus_a)\ a)\ A\ B \longleftrightarrow (\oplus_a)\ a\ `\ A = B$
  ⟨*proof*⟩

end

5.6.7 Cancellative commutative semigroups

Definitions

Abstract cancellative commutative semigroups.

locale *cancel-comm-semigroup* = *comm*: *comm-semigroup f* + *binary-op fi*
  for *f fi* :: $'a \Rightarrow 'a \Rightarrow 'a$ +
  assumes *add-diff-cancel-left'*[*simp*]: *fi* (*f a b*) *a* = *b*
    and *diff-diff-add*[*tts-algebra-simps*, *tts-field-simps*]:
    fi (*fi a b*) *c* = *fi a* (*f b c*)

locale *cancel-comm-semigroup-syntax = comm-semigroup-syntax f + binary-op fi*
  for *f fi* :: *'a* ⇒ *'a* ⇒ *'a*
begin

notation *fi* (**infixl** ‹⊖$_a$› 65)

end

Concrete syntax.

locale *cancel-comm-semigroup-add = cancel-comm-semigroup plus minus*
  for *plus minus* :: *'a* ⇒ *'a* ⇒ *'a*
begin

sublocale *comm-semigroup-add plus* ⟨*proof*⟩
sublocale *minus minus* ⟨*proof*⟩

end

locale *cancel-comm-semigroup-mult = cancel-comm-semigroup times divide*
  for *times divide* :: *'a* ⇒ *'a* ⇒ *'a*
begin

sublocale *comm-semigroup-mult times* ⟨*proof*⟩
sublocale *divide divide* ⟨*proof*⟩

end

Pairs.

locale *cancel-comm-semigroup-pair =*
  alg$_a$: *cancel-comm-semigroup f$_a$ fi$_a$ + alg$_b$*: *cancel-comm-semigroup f$_b$ fi$_b$*
  for *f$_a$ fi$_a$* :: *'a* ⇒ *'a* ⇒ *'a* **and** *f$_b$ fi$_b$* :: *'b* ⇒ *'b* ⇒ *'b*
begin

sublocale *comm-semigroup-pair f$_a$ f$_b$* ⟨*proof*⟩
sublocale *rev*: *cancel-comm-semigroup-pair f$_b$ fi$_b$ f$_a$ fi$_a$* ⟨*proof*⟩

end

locale *cancel-comm-semigroup-pair-syntax =*
  comm-semigroup-pair-syntax f$_a$ f$_b$ + binary-op fi$_a$ + binary-op fi$_b$
  for *f$_a$ fi$_a$ f$_b$ fi$_b$*
begin

notation *fi$_a$* (**infixl** ‹⊖$_a$› 65)
notation *fi$_b$* (**infixl** ‹⊖$_b$› 65)

end

Results

context *cancel-comm-semigroup*
begin

interpretation *cancel-comm-semigroup-syntax* ⟨*proof*⟩

lemma *add-diff-cancel-right'*[*simp*]: (*a* ⊕$_a$ *b*) ⊖$_a$ *b* = *a*
  ⟨*proof*⟩

sublocale *cancel*: *cancel-semigroup*
⟨*proof*⟩

lemmas *cancel-semigroup-axioms* = *cancel.cancel-semigroup-axioms*

lemma *add-diff-cancel-left*[*simp*]: $(c \oplus_a a) \ominus_a (c \oplus_a b) = a \ominus_a b$
⟨*proof*⟩

lemma *add-diff-cancel-right*[*simp*]: $(a \oplus_a c) \ominus_a (b \oplus_a c) = a \ominus_a b$
⟨*proof*⟩

lemma *diff-right-commute*: $a \ominus_a c \ominus_a b = a \ominus_a b \ominus_a c$
⟨*proof*⟩

end

context *cancel-comm-semigroup-pair*
begin

sublocale *cancel*: *cancel-semigroup-pair* ⟨*proof*⟩

lemmas *cancel-semigroup-pair-axioms* = *cancel.cancel-semigroup-pair-axioms*

end

5.7 Extension of the theory *Lifting-Set*

context
includes *lifting-syntax*
begin

lemma *set-pred-eq-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A*
shows
((*rel-set A* ===> (=)) ===> (*rel-set A* ===> (=)) ===> (=))
(λ*X Y*. ∀ *s*⊆*Collect* (*Domainp A*). *X s* = *Y s*)
((=)::['*b set* ⇒ *bool*, '*b set* ⇒ *bool*] ⇒ *bool*)
⟨*proof*⟩ **lemma** *vimage-fst-transfer-h*:

pred-prod (*Domainp A*) (*Domainp B*) *x* =
(*x* ∈ *Collect* (*Domainp A*) × *Collect* (*Domainp B*))

⟨*proof*⟩

lemma *vimage-fst-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A right-total B*
shows
((*rel-prod A B* ===> *A*) ===> *rel-set A* ===> *rel-set* (*rel-prod A B*))
(λ*f S*. (*f* −‘ *S*) ∩ ((*Collect* (*Domainp A*)) × (*Collect* (*Domainp B*))))
vimage
⟨*proof*⟩

lemma *vimage-snd-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *right-total A bi-unique B right-total B*
shows
((*rel-prod A B* ===> *B*) ===> *rel-set B* ===> *rel-set* (*rel-prod A B*))
(λ*f S*. (*f* −‘ *S*) ∩ ((*Collect* (*Domainp A*)) × (*Collect* (*Domainp B*))))
vimage
⟨*proof*⟩

lemma *vimage-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique B right-total A*
shows
((*A* ===> *B*) ===> (*rel-set B*) ===> *rel-set A*)
(λ*f s*. (*vimage f s*) ∩ (*Collect* (*Domainp A*))) (−‘)
⟨*proof*⟩

lemma *pairwise-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A*
shows ((*A* ===> *A* ===> (=)) ===> *rel-set A* ===> (=)) *pairwise pairwise*
⟨*proof*⟩

lemma *disjnt-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A*
shows (*rel-set A* ===> *rel-set A* ===> (=)) *disjnt disjnt*
⟨*proof*⟩

lemma *bij-betw-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A bi-unique B*
shows ((*A* ===> *B*) ===> *rel-set A* ===> *rel-set B* ===> (=)) *bij-betw bij-betw*
⟨*proof*⟩

end

5.8 Abstract semigroups on sets

5.8.1 Background

The results presented in this section were ported (with amendments and additions) from the theory *Groups* in the main library of Isabelle/HOL.

5.8.2 Binary operations

Abstract binary operation.

locale *binary-op-base-ow* =
  fixes $U :: 'a\ set$ **and** $f :: 'a \Rightarrow 'a \Rightarrow 'a$

locale *binary-op-ow* = *binary-op-base-ow* $U\ f$ **for** $U :: 'a\ set$ **and** f +
  assumes *op-closed*: $x \in U \Longrightarrow y \in U \Longrightarrow f\ x\ y \in U$

locale *binary-op-syntax-ow* = *binary-op-base-ow* $U\ f$ **for** $U :: 'a\ set$ **and** f
begin

notation f (**infixl** $\langle \oplus_a \rangle$ 70)

end

Concrete syntax.

locale *plus-ow* = *binary-op-ow* U *plus* **for** $U :: 'a\ set$ **and** *plus*
begin

notation *plus* (**infixl** $\langle +_a \rangle$ 65)

end

locale *minus-ow* = *binary-op-ow* U *minus* **for** $U :: 'a\ set$ **and** *minus*
begin

notation *minus* (**infixl** $\langle -_a \rangle$ 65)

end

locale *times-ow* = *binary-op-ow* U *times* **for** $U :: 'a\ set$ **and** *times*
begin

notation *times* (**infixl** $\langle *_a \rangle$ 70)

end

locale *divide-ow* = *binary-op-ow* U *divide* **for** $U :: 'a\ set$ **and** *divide*
begin

notation *divide* (**infixl** $\langle '/_a \rangle$ 70)

end

Pairs.

locale *binary-op-base-pair-ow* =
  alg_a: *binary-op-base-ow* $U_a\ f_a$ + alg_b: *binary-op-base-ow* $U_b\ f_b$
  for $U_a :: 'a\ set$ **and** f_a **and** $U_b :: 'b\ set$ **and** f_b

locale *binary-op-pair-ow = alg$_a$: binary-op-ow U_a f_a + alg$_b$: binary-op-ow U_b f_b*
for $U_a :: 'a\ set$ **and** f_a **and** $U_b :: 'b\ set$ **and** f_b
begin

sublocale *binary-op-base-pair-ow U_a f_a U_b f_b ⟨proof⟩*
sublocale *rev: binary-op-base-pair-ow U_b f_b U_a f_a ⟨proof⟩*

end

locale *binary-op-pair-syntax-ow = binary-op-base-pair-ow U_a f_a U_b f_b*
for $U_a :: 'a\ set$ **and** f_a **and** $U_b :: 'b\ set$ **and** f_b
begin

notation f_a (**infixl** ‹$\oplus_a$› 70)
notation f_b (**infixl** ‹$\oplus_b$› 70)

end

Results

context *binary-op-ow*
begin

interpretation *binary-op-syntax-ow ⟨proof⟩*

lemma *op-closed'[simp]: $\forall x{\in}U.\ \forall y{\in}U.\ x \oplus_a y \in U$ ⟨proof⟩*

tts-register-sbts ‹$(\oplus_a)$› | U *⟨proof⟩*

end

5.8.3 Simple semigroups

Definitions

Abstract semigroup.

locale *semigroup-ow = binary-op-ow U f* **for** $U :: 'a\ set$ **and** f +
assumes *assoc[tts-ac-simps]*:
$[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow f\ (f\ a\ b)\ c = f\ a\ (f\ b\ c)$

locale *semigroup-syntax-ow = binary-op-syntax-ow U f* **for** $U :: 'a\ set$ **and** f

Concrete syntax.

locale *semigroup-add-ow = semigroup-ow U plus* **for** $U :: 'a\ set$ **and** *plus*
begin

sublocale *plus-ow U plus ⟨proof⟩*

end

locale *semigroup-mult-ow = semigroup-ow U times* **for** $U :: 'a\ set$ **and** *times*
begin

sublocale *times-ow U times ⟨proof⟩*

end

Pairs.

locale *semigroup-pair-ow* = alg_a: *semigroup-ow* U_a f_a + alg_b: *semigroup-ow* U_b f_b
  for U_a :: $'a$ *set* **and** f_a **and** U_b :: $'b$ *set* **and** f_b
begin

sublocale *binary-op-pair-ow* U_a f_a U_b f_b ⟨*proof*⟩
sublocale *rev*: *semigroup-pair-ow* U_b f_b U_a f_a ⟨*proof*⟩

end

locale *semigroup-pair-syntax-ow* = *binary-op-pair-syntax-ow* U_a f_a U_b f_b
  for U_a :: $'a$ *set* **and** f_a **and** U_b :: $'b$ *set* **and** f_b

Transfer rules

lemma *semigroup-ow*[*ud-with*]: *semigroup* = *semigroup-ow UNIV*
  ⟨*proof*⟩

lemma *semigroup-pair-ow*[*ud-with*]:
  semigroup-pair = (λf_a f_b. *semigroup-pair-ow UNIV* f_a *UNIV* f_b)
  ⟨*proof*⟩

context
  includes *lifting-syntax*
begin

lemma *semigroup-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=)) *semigroup-ow semigroup-ow*
  ⟨*proof*⟩

lemma *semigroup-pair-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]:
    bi-unique A right-total A bi-unique B right-total B
  shows
    (
      rel-set A ===> (*A* ===> *A* ===> *A*) ===>
      rel-set B ===> (*B* ===> *B* ===> *B*) ===>
      (=)
)
    semigroup-pair-ow semigroup-pair-ow
  ⟨*proof*⟩

end

5.8.4 Commutative semigroups

Definitions

Abstract commutative semigroup.

locale *comm-semigroup-ow* = *semigroup-ow U f* **for** U :: $'a$ *set* **and** f +
  assumes *commute*[*tts-ac-simps*]: $a \in U \Longrightarrow b \in U \Longrightarrow f\ a\ b = f\ b\ a$

locale *comm-semigroup-syntax-ow* = *semigroup-syntax-ow U f*
  for U :: $'a$ *set* **and** f

Concrete syntax.

locale *comm-semigroup-add-ow* = *comm-semigroup-ow U plus*

for $U :: 'a\ set$ **and** *plus*
begin

sublocale *semigroup-add-ow U plus* ⟨*proof*⟩

end

locale *comm-semigroup-mult-ow* = *comm-semigroup-ow U times*
for $U :: 'a\ set$ **and** *times*
begin

sublocale *semigroup-mult-ow U times* ⟨*proof*⟩

end

Pairs.

locale *comm-semigroup-pair-ow* =
alg_a: *comm-semigroup-ow* $U_a\ f_a$ + alg_b: *comm-semigroup-ow* $U_b\ f_b$
for $U_a :: 'a\ set$ **and** f_a **and** $U_b :: 'b\ set$ **and** f_b
begin

sublocale *semigroup-pair-ow* $U_a\ f_a\ U_b\ f_b$ ⟨*proof*⟩
sublocale *rev*: *comm-semigroup-pair-ow* $U_b\ f_b\ U_a\ f_a$ ⟨*proof*⟩

end

locale *comm-semigroup-pair-syntax-ow* = *semigroup-pair-syntax-ow* $U_a\ f_a\ U_b\ f_b$
for $U_a :: 'a\ set$ **and** f_a **and** $U_b :: 'b\ set$ **and** f_b

Transfer rules

lemma *comm-semigroup-ow*[*ud-with*]: *comm-semigroup* = *comm-semigroup-ow UNIV*
⟨*proof*⟩

lemma *comm-semigroup-pair-ow*[*ud-with*]:
comm-semigroup-pair = (λ$f_a\ f_b$. *comm-semigroup-pair-ow UNIV* f_a *UNIV* f_b)
⟨*proof*⟩

context
includes *lifting-syntax*
begin

lemma *comm-semigroup-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]: *bi-unique A right-total A*
shows
(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
comm-semigroup-ow comm-semigroup-ow
⟨*proof*⟩

lemma *comm-semigroup-pair-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
(
rel-set A ===> (*A* ===> *A* ===> *A*) ===>
rel-set B ===> (*B* ===> *B* ===> *B*) ===>
(=)
)

comm-semigroup-pair-ow comm-semigroup-pair-ow
⟨*proof*⟩

end

Relativization

context *comm-semigroup-ow*
begin

interpretation *comm-semigroup-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
substituting *comm-semigroup-ow-axioms*
eliminating through *auto*
begin

tts-lemma *left-commute*:
assumes $b \in U$
and $a \in U$
and $c \in U$
shows $b \oplus_a (a \oplus_a c) = a \oplus_a (b \oplus_a c)$
is *comm-semigroup.left-commute*⟨*proof*⟩

end

end

5.8.5 Cancellative semigroups

Definitions

Abstract cancellative semigroup.

locale *cancel-semigroup-ow* = *semigroup-ow U f* **for** *U* :: *'a set* **and** *f* +
assumes *add-left-imp-eq*:
$[\![a \in U; b \in U; c \in U; f\ a\ b = f\ a\ c]\!] \Longrightarrow b = c$
assumes *add-right-imp-eq*:
$[\![b \in U; a \in U; c \in U; f\ b\ a = f\ c\ a]\!] \Longrightarrow b = c$

locale *cancel-semigroup-syntax-ow* = *semigroup-syntax-ow U f*
for *U* :: *'a set* **and** *f*

Concrete syntax.

locale *cancel-semigroup-add-ow* = *cancel-semigroup-ow U plus*
for *U* :: *'a set* **and** *plus*
begin

sublocale *semigroup-add-ow U plus* ⟨*proof*⟩

end

locale *cancel-semigroup-mult-ow* = *cancel-semigroup-ow U times*
for *U* :: *'a set* **and** *times*
begin

sublocale *semigroup-mult-ow U times* ⟨*proof*⟩

end

Pairs.

locale *cancel-semigroup-pair-ow* =
  alg_a: *cancel-semigroup-ow* U_a f_a + alg_b: *cancel-semigroup-ow* U_b f_b
  for U_a :: $'a$ *set* **and** f_a **and** U_b :: $'b$ *set* **and** f_b
begin

sublocale *semigroup-pair-ow* U_a f_a U_b f_b ⟨*proof*⟩
sublocale *rev*: *cancel-semigroup-pair-ow* U_b f_b U_a f_a ⟨*proof*⟩

end

locale *cancel-semigroup-pair-syntax-ow* = *semigroup-pair-syntax-ow* U_a f_a U_b f_b
  for U_a :: $'a$ *set* **and** f_a **and** U_b :: $'b$ *set* **and** f_b

Transfer rules

lemma *cancel-semigroup-ow*[*ud-with*]:
  cancel-semigroup = *cancel-semigroup-ow UNIV*
  ⟨*proof*⟩

lemma *cancel-semigroup-pair-ow*[*ud-with*]:
  cancel-semigroup-pair = (λf_a f_b. *cancel-semigroup-pair-ow UNIV* f_a *UNIV* f_b)
  ⟨*proof*⟩

context
  includes *lifting-syntax*
begin

lemma *cancel-semigroup-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows
    (*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
      cancel-semigroup-ow cancel-semigroup-ow
  ⟨*proof*⟩

lemma *cancel-semigroup-pair-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]:
    bi-unique A right-total A bi-unique B right-total B
  shows
    (
      rel-set A ===> (*A* ===> *A* ===> *A*) ===>
      rel-set B ===> (*B* ===> *B* ===> *B*) ===>
      (=)
) *cancel-semigroup-pair-ow cancel-semigroup-pair-ow*
  ⟨*proof*⟩

end

Relativization

context *cancel-semigroup-ow*
begin

interpretation *cancel-semigroup-syntax-ow* ⟨*proof*⟩

tts-context

tts: (*?'a* **to** *U*)
rewriting *ctr-simps*
substituting *cancel-semigroup-ow-axioms*
eliminating through *auto*
begin

tts-lemma *add-right-cancel*:
assumes $b \in U$ **and** $a \in U$ **and** $c \in U$
shows $(b \oplus_a a = c \oplus_a a) = (b = c)$
is *cancel-semigroup.add-right-cancel*⟨*proof*⟩

tts-lemma *add-left-cancel*:
assumes $a \in U$ **and** $b \in U$ **and** $c \in U$
shows $(a \oplus_a b = a \oplus_a c) = (b = c)$
is *cancel-semigroup.add-left-cancel*⟨*proof*⟩

tts-lemma *inj-on-add'*:
assumes $a \in U$ **and** $A \subseteq U$
shows *inj-on* $(\lambda b.\ b \oplus_a a)\ A$
is *cancel-semigroup.inj-on-add'*⟨*proof*⟩

tts-lemma *inj-on-add*:
assumes $a \in U$ **and** $A \subseteq U$
shows *inj-on* $((\oplus_a)\ a)\ A$
is *cancel-semigroup.inj-on-add*⟨*proof*⟩

tts-lemma *bij-betw-add*:
assumes $a \in U$ **and** $A \subseteq U$ **and** $B \subseteq U$
shows *bij-betw* $((\oplus_a)\ a)\ A\ B = ((\oplus_a)\ a\ `\ A = B)$
is *cancel-semigroup.bij-betw-add*⟨*proof*⟩

end

end

5.8.6 Cancellative commutative semigroups

Definitions

Abstract cancellative commutative semigroups.

locale *cancel-comm-semigroup-ow = comm-semigroup-ow U f + binary-op-ow U fi*
for *U* :: *'a set* **and** *f fi* +
assumes *add-diff-cancel-left'*[*simp*]: $[\![\ a \in U;\ b \in U\ ]\!] \Longrightarrow fi\ (f\ a\ b)\ a = b$
and *diff-diff-add*[*tts-algebra-simps*, *tts-field-simps*]:
$[\![\ a \in U;\ b \in U;\ c \in U\ ]\!] \Longrightarrow fi\ (fi\ a\ b)\ c = fi\ a\ (f\ b\ c)$

locale *cancel-comm-semigroup-syntax-ow =*
comm-semigroup-syntax-ow U f + binary-op-base-ow U fi
for *U* :: *'a set* **and** *f fi*
begin

notation *fi* (**infixl** ‹$\ominus_a$› 65)

end

Concrete syntax.

locale *cancel-comm-semigroup-add-ow = cancel-comm-semigroup-ow U plus minus*
for *U* :: *'a set* **and** *plus minus*

begin

sublocale *comm-semigroup-add-ow U plus* ⟨*proof*⟩
sublocale *minus-ow U minus* ⟨*proof*⟩

end

locale *cancel-comm-semigroup-mult* = *cancel-comm-semigroup-ow U times divide*
  for *U* :: *'a set* **and** *times divide*
begin

sublocale *comm-semigroup-mult-ow U times* ⟨*proof*⟩
sublocale *divide-ow U divide* ⟨*proof*⟩

end

Pairs.

locale *cancel-comm-semigroup-pair-ow* =
  alg_a: *cancel-comm-semigroup-ow* U_a f_a fi_a +
  alg_b: *cancel-comm-semigroup-ow* U_b f_b fi_b
  for U_a :: *'a set* **and** f_a fi_a **and** U_b :: *'b set* **and** f_b fi_b
begin

sublocale *comm-semigroup-pair-ow* U_a f_a U_b f_b ⟨*proof*⟩
sublocale *rev*: *cancel-comm-semigroup-pair-ow* U_b f_b fi_b U_a f_a fi_a ⟨*proof*⟩

end

locale *cancel-comm-semigroup-pair-syntax-ow* =
  comm-semigroup-pair-syntax-ow U_a f_a U_b f_b +
  binary-op-ow U_a fi_a +
  binary-op-ow U_b fi_b
  for U_a :: *'a set* **and** f_a fi_a **and** U_b :: *'b set* **and** f_b fi_b
begin

notation fi_a (**infixl** ‹$\ominus_a$› 65)
notation fi_b (**infixl** ‹$\ominus_b$› 65)

end

Transfer rules

lemma *cancel-comm-semigroup-ow*[*ud-with*]:
  cancel-comm-semigroup = *cancel-comm-semigroup-ow UNIV*
  ⟨*proof*⟩

lemma *cancel-comm-semigroup-pair-ow*[*ud-with*]:
  cancel-comm-semigroup-pair =
    (λf_a fi_a f_b fi_b. *cancel-comm-semigroup-pair-ow UNIV* f_a fi_a *UNIV* f_b fi_b)
  ⟨*proof*⟩

context
  includes *lifting-syntax*
begin

lemma *cancel-comm-semigroup-ow-transfer*[*transfer-rule*]:
  assumes [*transfer-rule*]: *bi-unique A right-total A*
  shows

(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===> (=))
cancel-comm-semigroup-ow cancel-comm-semigroup-ow
⟨*proof*⟩

lemma *cancel-comm-semigroup-pair-ow-transfer*[*transfer-rule*]:
assumes [*transfer-rule*]:
bi-unique A right-total A bi-unique B right-total B
shows
(
rel-set A ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===>
rel-set B ===> (*B* ===> *B* ===> *B*) ===> (*B* ===> *B* ===> *B*) ===>
(=)
) *cancel-comm-semigroup-pair-ow cancel-comm-semigroup-pair-ow*
⟨*proof*⟩

end

Relativization

context *cancel-comm-semigroup-ow*
begin

interpretation *cancel-comm-semigroup-syntax-ow* ⟨*proof*⟩

tts-context
tts: (*?'a* **to** *U*)
sbterms: (‹*?f*::*?'a* ⇒ *?'a* ⇒ *?'a*› **to** *f*)
and (‹*?fi*::*?'a* ⇒ *?'a* ⇒ *?'a*› **to** *fi*)
rewriting *ctr-simps*
substituting *cancel-comm-semigroup-ow-axioms*
eliminating through *auto*
begin

tts-lemma *add-diff-cancel-right'*:
assumes $a \in U$ **and** $b \in U$
shows $a \oplus_a b \ominus_a b = a$
is *cancel-comm-semigroup.add-diff-cancel-right'*⟨*proof*⟩

tts-lemma *add-diff-cancel-right*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $a \oplus_a c \ominus_a b \oplus_a c = a \ominus_a b$
is *cancel-comm-semigroup.add-diff-cancel-right*⟨*proof*⟩

tts-lemma *add-diff-cancel-left*:
assumes $c \in U$ **and** $a \in U$ **and** $b \in U$
shows $c \oplus_a a \ominus_a c \oplus_a b = a \ominus_a b$
is *cancel-comm-semigroup.add-diff-cancel-left*⟨*proof*⟩

tts-lemma *diff-right-commute*:
assumes $a \in U$ **and** $c \in U$ **and** $b \in U$
shows $a \ominus_a c \ominus_a b = a \ominus_a b \ominus_a c$
is *cancel-comm-semigroup.diff-right-commute*⟨*proof*⟩

tts-lemma *cancel-semigroup-axioms*:
assumes $U \neq \{\}$
shows *cancel-semigroup-ow U* ($\oplus_a$)
is *cancel-comm-semigroup.cancel-semigroup-axioms*⟨*proof*⟩

end

sublocale *cancel-semigroup-ow*
⟨*proof*⟩

end

context *cancel-comm-semigroup-pair-ow*
begin

sublocale *cancel-semigroup-pair-ow* ⟨*proof*⟩

end

Bibliography

[1] Isabelle/HOL Standard Library, 2020. URL https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL/index.html.

[2] Isabelle/HOL Analysis, 2020. URL https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL-Analysis/index.html.

[3] C. Ballarin. Locales and Locale Expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, volume 3085, pages 34–50. Springer, Heidelberg, 2004. ISBN 978-3-540-22164-7.

[4] C. Ballarin. Locales: A Module System for Mathematical Theories. *Journal of Automated Reasoning*, 52(2):123–153, 2014.

[5] C. Ballarin, editor. *The Isabelle/HOL Algebra Library*. 2020. URL https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL-Algebra/document.pdf.

[6] A. J. Cain. *Nine Chapters on the Semigroup Art*. Lisbon, 2019.

[7] A. Chaieb and M. Wenzel. Context Aware Calculation and Deduction: Ring Equalities Via Gröbner Bases in Isabelle. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, volume 4573, pages 27–39. Springer, Heidelberg, 2007. ISBN 978-3-540-73083-5.

[8] J. Divasón, O. Kunčar, R. Thiemann, and A. Yamada. Perron-Frobenius Theorem for Spectral Radius Analysis. *Archive of Formal Proofs*, 2016.

[9] J. Gilcher, A. Lochbihler, and D. Traytel. Conditional Parametricity in Isabelle/HOL. In *TABLEAUX/FroCoS/ITP*, Brasília, Brazil, 2017.

[10] F. Haftmann and M. Wenzel. Constructive Type Classes in Isabelle. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs*, volume 4502, pages 160–174. Springer, Heidelberg, 2007. ISBN 978-3-540-74464-1.

[11] B. Huffman and O. Kunčar. Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL. In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs*, volume 8307, pages 131–146. Springer, Heidelberg, 2013. ISBN 978-3-319-03545-1.

[12] F. Immler. Automation for unverloading definitions, 2019. URL http://isabelle.in.tum.de/repos/isabelle/rev/ab5a8a2519b0.

[13] F. Immler. Re: [isabelle] Several questions in relation to a use case for "types to sets", 2019. URL https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2019-January/msg00072.html.

[14] F. Immler and B. Zhan. Smooth Manifolds and Types to Sets for Linear Algebra in Isabelle/HOL. In A. Mahboubi and M. O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal*, CPP 2019, pages 65–77. ACM, New York, 2019. ISBN 978-1-4503-6222-1.

[15] F. Kammüller, M. Wenzel, and L. C. Paulson. Locales A Sectioning Concept for Isabelle. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin-Mohring, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 149–165, Heidelberg, 1999. Springer. ISBN 978-3-540-48256-7.

[16] K. Kappelmann, L. Bulwahn, and S. Willenbrink. SpecCheck - Specification-Based Testing for Isabelle/ML. *Archive of Formal Proofs*, 2021.

[17] A. Krauss and A. Schropp. A Mechanized Translation from Higher-Order Logic to Set Theory. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving*, volume 6172, pages 323–338. Springer, Heidelberg, 2010. ISBN 978-3-642-14051-8.

[18] O. Kunčar. *Types, Abstraction and Parametric Polymorphism in Higher-Order Logic*. PhD thesis, Technische Universität München, Munich, 2015.

[19] O. Kunčar and A. Popescu. From Types to Sets by Local Type Definitions in Higher-Order Logic. In J. C. Blanchette and S. Merz, editors, *Interactive Theorem Proving*, volume 9807, pages 200–218. Springer, Heidelberg, 2016. ISBN 978-3-319-43144-4.

[20] O. Kunčar and A. Popescu. Comprehending Isabelle/HOL's Consistency. In H. Yang, editor, *Programming Languages and Systems*, volume 10201, pages 724–749. Springer, Heidelberg, 2017. ISBN 978-3-662-54433-4.

[21] O. Kunčar and A. Popescu. From Types to Sets by Local Type Definition in Higher-Order Logic. *Journal of Automated Reasoning*, 62(2):237–260, 2019.

[22] P. Lammich. Automatic Data Refinement. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, pages 84–99, Heidelberg, 2013. Springer. ISBN 978-3-642-39634-2.

[23] A. Maletzky. Hilbert's Nullstellensatz. *Archive of Formal Proofs*, 2019.

[24] M. Milehins. Conditional Transfer Rule. *Archive of Formal Proofs*, 2021.

[25] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (revised)*. MIT Press, Cambridge, Massachusetts, 1997. ISBN 978-0-262-63181-5.

[26] T. Nipkow and G. Snelting. Type Classes and Overloading Resolution via Order-Sorted Unification. In J. Hughes, editor, *Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 1–14. Springer, Heidelberg, 1991. ISBN 978-3-540-47599-6.

[27] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.

[28] L. C. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[29] C. Urban. *The Isabelle Cookbook: A Gentle Tutorial for Programming Isabelle/ML*. 2019.

[30] M. Wenzel. Type Classes and Overloading in Higher-Order Logic. In E. L. Gunter and A. P. Felty, editors, *Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, pages 307–322. Springer, Heidelberg, 1997. ISBN 978-3-540-69526-4.

[31] M. Wenzel. Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin-Mohring, editors, *Theorem Proving in Higher Order Logics*, volume 1690, pages 167–183. Springer, Heidelberg, 1999. ISBN 978-3-540-66463-5.

[32] M. Wenzel. Isabelle/Isar — a Generic Framework for Human-Readable Proof Documents. *Studies in Logic, Grammar and Rhetoric*, 10(23):277–297, 2007.

[33] M. Wenzel. The Isabelle/Isar Implementation. 2019.

[34] M. Wenzel. The Isabelle/Isar Reference Manual. 2019.

[35] M. M. Wenzel. *Isabelle/Isar — A Versatile Environment for Human-Readable Formal Proof Documents*. PhD thesis, Technische Universität München, Munich, 2001.