PROOF OF THE BASIC THEOREM ON CONCEPT LATTICES IN ISABELLE/HOL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

BARIŞ SERTKAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

JULY 2003

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Tayfur Öztürk
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Andreas Tiefenbach
Co-Supervisor

Assist. Prof. Dr. Halit Oğuztüzün
Supervisor

Examining Committee Members

Prof. Dr. Faruk Polat

Assoc. Prof. Dr. Cem Bozşahin

Assist. Prof. Dr. Halit Oğuztüzün

Dr. Ayşenur Birtürk

Dr. Sıtkı Irk

ABSTRACT

PROOF OF THE BASIC THEOREM ON CONCEPT LATTICES
IN ISABELLE/HOL

Sertkaya, Barış

MS, Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Halit Oğuztüzün

Co-Supervisor: Assist. Prof. Dr. Andreas Tiefenbach

July 2003, 72 pages

Formal Concept Analysis is an emerging field of applied mathematics based on a lattice-theoretic formalization of the notions of concept and conceptual hierarchy. It thereby facilitates mathematical thinking for conceptual data analysis and knowledge processing.

Isabelle, on the other hand, is a generic interactive theory development environment for implementing logical formalisms. It has been instantiated to support reasoning in several object-logics. Specialization of Isabelle for Higher Order Logic is called Isabelle/HOL.

Our long term goal is to formalize the theory of Formal Concept Analysis in Isabelle/HOL. This will provide a mechanized theory for researchers who want to prove their own theorems with utmost precision, and for developers who want to design knowledge processing algorithms. The specific accomplishment of this thesis is a machine-checked version of the proof of the Basic Theorem of Concept Lattices, which appears in the book "Formal Concept Analysis" by Ganter and Wille. As a by-product, the underlying lattice theory by F. Kammueller has been extended.

Keywords: Formal Concept Analysis, Isabelle, Higher Order Logic, Formalized Mathematics

ÖZ

KAVRAM ÖRGÜLERİ HAKKINDA TEMEL TEOREMİN
ISABELLE/HOL'DA İSPATI

Sertkaya, Barış
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü Bölümü
Tez Yöneticisi: Assist. Prof. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi: Assist. Prof. Dr. Andreas Tiefenbach

Temmuz 2003, 72 sayfa

Formal Kavram Analizi, uygulamalı matematiğin gelişmekte olan bir dalıdır. Kavramlar ve kavramların hiyerarşisinin örgü kuramı temellerine dayandırılarak matematiksel olarak formalize edilmesini amaçlar. Böylece kavramsal veri analizi ve bilgi işlemenin matematiksel düşünme yöntemini sağlar.

Isabelle, mantıksal formalizasyon yapmak için kullanılan jenerik bir kuram geliştirme ortamıdır. Çeşitli mantıklarda çıkarım yapmayı destekler. Isabelle'in yüksek dereceli mantık icin özelleşmiş haline Isabelle/HOL denir.

Bu çalışmanın uzun vadeli amacı Formal Kavram Analizi kuramının Isabelle/HOL ortamında formalize edilmesidir. Formalizasyon, Formal Kavram Analizi konusunda kendi teoremlerini ispatlamak isteyen araştırmacılara ve bilgi işleme algoritmaları tasarlamak isteyen geliştiricilere bilgisayar tarafından kontrol edilmiş bir kuram sunmayı amaçlamaktadır. Bu çalışmanın belirgin başarısı, Ganter ve Wille tarafından yazılmış olan Formal Concept Analysis kitabındaki Kavram Örgüleri Hakkındaki Temel Teoremin bilgisayara kontrol ettirilmiş olmasıdır. Çalışmanın yan ürünü olarak F. Kammueller tarafından geliştirilmiş olan örgü kuramı genişletilmiştir.

Anahtar Kelimeler: Formal Kavram Analizi, Isabelle, Yüksek Dereceli Mantık, Formelleştirilmiş Matematik

To Whom It May Be Useful To

ACKNOWLEDGMENTS

I am deeply indebted to my supervisor Halit Oğuztüzün whose stimulating motivation and valuable ideas helped me to complete this work.

I would like to thank to Lawrence Paulson and Tobias Nipkow for their help through the isabelle-users mailing list.

Thanks to my roommates Sinan and Ersan for the fun we had in our office in the spare times.

Special thanks to my love İrem for her moral support and precious love, who was always standing by me in my hard times during this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

I.1 Formal Concept Analysis Overview

Formal Concept Analysis is a field of applied mathematics based on the mathematical formalization of the philosophical understanding of concept and conceptual hierarchy, borrowing mathematical foundations from lattice theory. It thereby activates mathematical thinking for conceptual data analysis and knowledge processing.

The method is mainly used for the *analysis of data* , i.e. for investigating and processing explicitly given information. Such data is structured into units which are formal abstractions of *concepts* of human thought allowing meaningful and comprehensible interpretation. The prefix *formal* is used to emphasize that these formal concepts are mathematical entities and must not be identified with the concepts of the mind. The same prefix indicates that the basic data format, that of a *formal context*, is merely a formalization that encodes only a small portion of what is usually referred to as a "context". A concept is constituted by two parts: its extension, which consists of all the objects belonging to the concept, and its intention which contains the attributes common to all objects of the concept. This formalization allows to form all concepts of a context and introduce a subsumption hierarchy between the concepts, giving a complete lattice called the concept lattice of the context.

I.2 Isabelle Proof Assistant Overview

Isabelle [4, 5] is a generic interactive theory development environment for implement-

ing logical formalisms. It has been instantiated to support reasoning in several object-logics like first-order-logic, higher-order-logic, Zermelo-Fraenkel set theory and T, S4 and S43 modal logic systems. Its main usage areas are verification of compilers, security and cryptographic protocols, software, hardware and formalization of programming languages, logics and mathematics.

Unlike automatic theorem provers, Isabelle and other interactive theorem provers are directed by the user during a proof. After stating the goal, the user directs the prover by some operations on the goal, called tactics at each step. Isabelle provides various kinds of tactics like rewrite, simplification, resolution, tableu, assumption, induction and so on. By applying the tactics, the user tries to solve the goal. Tactics may lead to zero or more subgoals, and the user tries to solve all of the resultant subgoals. At the end, the user has a formal proof. So, it is called a *proof assistant* or a *proof checker*.

It is generic in the sense that the the reasoning can be made in various object logics. Specialization of Isabelle for Higher Order Logic is called Isabelle/HOL.

I.3 Formalized Mathematics

Formalized mathematics means expressing mathematics, both statements and proofs, in a formal language with strict rules of grammar and unambigious semantics. It addresses both precision and correctness. The idea of formalization is considered in two parts:

- Formalizing the statements of theorems, and the implicit context (definitions etc.) on which they depend.
- Formalizing the proofs of the results and subjecting them to precise checking.

The early efforts of formalizing mathematics, were mainly concerned with completely automatic proofs of theorems. They were based on emulating the way mathematicians actually think or using exhaustive search for a proof in certain formal systems for first-order logic.

Nevertheless, it was soon noticed that automatic theorem proving efforts were unable to prove many substantial mathematical theorems unaided. So, the efforts moved to the development of proof assistants or proof checkers and interactive theorem provers. Here the idea is to not to prove difficult theorems automatically, but to assist in con-

structing a formal proof. Several pioneering projects for the computer formalization of mathematics appeared in the 1970s. A notable example was the Automath effort led by de Bruijn [6]. Here, a language for formal mathematics was designed, together with a computer system for checking the correctness of formal texts. Significant parts of mathematics were proof-checked. Jutting formalized the famous book on the construction of the real number field by Landau [7]. The history of the project and the lessons derived from it are detailed by Nederpelt [8]. Though the project was important and influential, the Automath system is hardly used today.

I.4 Thesis Overview

In this thesis, our aim is to formalize the theory of Formal Concept Analysis in Isabelle/HOL. Our motivation is to provide a machine-checked theory of Formal Concept Analysis for researchers who want to prove their own theorems and for developers who want to implement knowledge representation tools. Our formalization is based on the book **Formal Concept Analysis** [13]. We give the formalization of the basic datatypes, fundamental propositions and the **Basic Theorem on Concept Lattices** of the theory in the first chapter of the book.

From the formalized mathematics point of view, the thesis is yet another effort to formalize a piece of mathematics. Our contribution is that, this is the is the first known attempt to formalize Formal Concept Analysis, which is fairly a new subject. From a practical point of view it can be a contribution to the Isabelle/HOL theory library by some extensions and make-up.

Chapter II gives some background on interactive theorem proving and its history. Later, it specializes on the Isabelle/HOL Proof Assistant. It gives some insights on the internals and usage of Isabelle. It introduces types, terms and formulae of Isabelle and theory development in Isabelle. A simple proof example is also provided. Then most commonly used tactics are explained. Formal Concept Analysis section is a rough introduction to Formal Concept Analysis. It gives basic definitions of FCA and an example. Similar formalized mathematics efforts are given in Related Work section.

Chapter III gives the formalization details of datatypes, proofs of fundamental and auxiliary lemmata and proof of the Basic Theorem on Concept Lattices. The proofs appearing in the book and the proofs of our formalization are presented in an interleaved

fashion. The commentary of the formalization gives references to the actual proof script in Appendix B.

Chapter IV concludes with discussions on the tool and the proofs in the textbook and the book Introduction to Lattices and Order. Possible future work on the formalization of Formal Concept Analysis is given in the Future Work section.

CHAPTER II

BACKGROUND

II.1 Interactive Theorem Proving

In 1969, Dana Scott devised a logic (but not published until 1993 [1]) which is later called Logic for Computable Functions by Robin Milner. The LCF logic has terms from the typed λ-calculus and formulae from predicate calculus. Types are interpreted as Scott domains (CPOs) and the logic is intended for reasoning, using fixed-point induction, about recursively defined functions of the sort used in denotational semantic definitions.

Around 1972, Robin Milner developed a proof checker for LCF. The system was called Stanford LCF, which is described by Milner as follows [3]:

> The proof-checking program is designed to allow the user interactively to generate formal proofs about computable functions and functionals over a variety of domains, including those of interest to the computer scientist - for example, integers, lists and computer programs and their semantics. The user's task is alleviated by two features: a subgoaling facility and a powerful simplification mechanism.

His results led him to seek a compromise between fully automatic theorem proving, which seemed impossibly difficult, and single step proof checking which seemed impossibly tedious. Around 1977, his group developed a *programmable* proof checker, Edinburgh LCF. Inference rules were expressed as functions in a programmable meta-language (called ML). By writing programs in ML, users could automate the proof process as much as desired.

LCF also permits a proof to be constructed backwards from a *goal*, the conjecture to be proved. An LCF *tactic* is a function that reduces a goal to zero or more subgoals. Once all the subgoals have been solved, LCF (using complex bookkeeping) constructs the corresponding forwards proof, yielding the desired theorem. *Tacticals* combine tactics in various ways. Tactics and tacticals constitute a powerful control language; they can describe search methods such as 'repeatedly apply Rule X then repeatedly apply either Rule Y or Rule Z'.

Edinburgh LCF proved a great success. Many similar systems were developed: including Cambridge LCF (Paulson, 1987); Nuprl for Constructive Type Theory (Constable et al., 1986); HOL for higher-order logic (Gordon, 1988); and Isabelle [4, 5].

II.2 Isabelle Proof Assistant

Isabelle is one of the successors of LCF. It is a *generic* interactive theorem prover, designed for reasoning in a variety of formal theories. It is based on the typed λ-calculus. But its primary inference rule is a generalization of Horn clause resolution. It uses resolution to implement proof checking in a generic (logic-independent) way. It provides no uniform search strategy, but several tools based on lazy lists. They can express depth-first, best-first and iterative-deepening strategies, and can be combined to yield automatic proof procedures.

Isabelle is generic in the sense that it provides proof procedures for Constructive Type Theory (Per Matin-L of, 1984), various first-order logics, some systems of Modal Logics, Zermelo-Fraenkel Set Theory, and Higher-Order Logic, which are called *object-logics*. Objects-logics are formalized within Isabelle's *meta-logic*, which is intutionistic higher-order logic with implication, universal quantifiers, and equality. The implication $\phi \Longrightarrow \psi$ means 'ϕ implies ψ' and expresses logical entailment. The quantification $\bigwedge x.\phi$ means 'ϕ is true for all x' and expresses generality in the rules and axiom schemes. The equality $a \equiv b$ means 'a equals b' and allows new symbols to be defined as abbreviations. For instance Isabelle represents the inference rule

$$\frac{P \quad Q}{P \,\&\, Q}$$

by the following axiom in the meta-logic:

$$\bigwedge P.\, \bigwedge Q.\, P \Longrightarrow (Q \Longrightarrow P \,\&\, Q)$$

The structure of rules generalizes PROLOG's Horn clauses; proof procedures can exploit logic programming techniques. The structure of the Isabelle is given in figure II.1. The specialization of Isabelle for HOL is called Isabelle/HOL. It is the most widely used object-logic for proof-checking jobs.

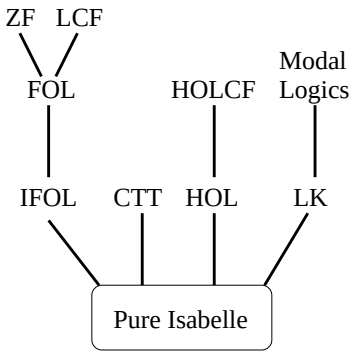

Figure II.1: Structure of Isabelle

II.2.1 Isabelle Theories

Working with Isabelle/HOL means creating theories. Roughly speaking, a **theory** is a named collection of types, functions, and theorems, much like a module in a programming language or a specification in a specification language. The general format of a theory `T` is

```
theory T = B1 + ... + Bn:
declarations, definitions, and proofs
end
```

where `B1, ... , Bn` are the names of existing (parent) theories that `T` is based on and `declarations, definitions and proofs` represents the newly introduced concepts (types, functions etc.) and proofs about them. Everything defined in the parent theories (and their parents recursively) is automatically visible.

II.2.2 Types, Terms and Formulae

Embedded in a theory are the types, terms and formulae of HOL. HOL is a typed logic whose types system resembles that of functional programming languages like ML or Haskell. Thus there are

- base types, in particular `bool`, the type of truth values, and `nat` the type of natural numbers.
- type constructors, in particular `list`, the type of lists, and `set` the type of sets (e.g. `nat list`).
- function types, denoted by $\Rightarrow$. In HOL $\Rightarrow$ represents *total* functions only. Curried functions are supported.
- type variables, denteod by `'a`, `'b` etc., just like in ML.

Terms are formed as in functional programming by applying functions to arguments and **Formulae** are terms of type `bool`.

II.2.3 Theorem Proving with Isabelle

Proof trees are derived rules, and are built by joining rules together. This comprimises both forwards and backwards proof. Backwards proof works by matching a goal with conclusion of a rule; the premises become the subgoals. Forwards proof works by matching theorems to the premises of a rules, making a new theorem.

A typical proof starts with first stating the goal using the `Goal` command, proceeds with applying tactics aiming to solve this goal using the `by` command, and ends with the `qed` (*quod erat demonstrandum*: what is to be shown in Latin) command which names and stores the proved theorem. Tactics may lead to zero or more subgoals. The proof process continues until no subgoals are left. Here is a very simple example which proves `P & Q` from the assumptions `P` and `Q` using conjunction introduction (The lines starting with > are the commands typed in, others are Isabelle's responses):

```
> Goal "[| P ; Q |] ==> P & Q";
Level 0 (1 subgoal)
[| P; Q |] ==> P & Q
 1. [| P; Q |] ==> P & Q
> by(rtac conjI 1);
Level 1 (2 subgoals)
[| P; Q |] ==> P & Q
 1. [| P; Q |] ==> P
 2. [| P; Q |] ==> Q
```

```
> by(atac 1);
Level 2 (1 subgoal)
[| P; Q |] ==> P & Q
 1. [| P; Q |] ==> Q
> by(atac 1);
Level 3
[| P; Q |] ==> P & Q
No subgoals!
> qed "example";
```

The first line is the initial goal we state to Isabelle. In lines 2,3 and 4 Isabelle responses with top-level goal we stated. In line 5, we apply conjunction introduction (`conjI`) rule using a resolution tactic (`rtac`). It responses with 2 subgoals in lines 8 and 9. We solve both subgoals from the assumptions by applying the assumption tactic (`atac`), which tries to solve the subgoal from the assumptions, twice and finish our proof. In the last line, we store the proof with name `example` using the `qed` command.

II.2.4 Commonly Used Tactics

Most widely used tactics are; resolution, rewrite, induction, assumption, tableu, automatic and simplification tactics. Some tactics from these main groups are

- resolution
  - `resolve_tac` *thm i*: resolve the rule's (*thm*) conclusion with subgoal `i`.
  - `eresolve_tac` *thms i*: perform elimination-resolution.
  - `dresolve_tac` *thms i*: perform destruction-resolution.
- instantiation
  - `res_inst_tac` *insts thm i*: instantiates the *thm* with the instantiations in *insts*, then performs resolution on subgoal *i*
  - `eres_inst_tac` *insts thm i*: like `res_inst_tac`, but performs elimination resolution
  - `dres_inst_tac` *insts thm i*: like `res_inst_tac`, but performs destruction resolution

- rewrite
  - `rewrite_goals_tac` *defs*: unfold *defs* throughout the subgoals
  - `rewrite_tac` *defs*: unfold *defs* throughout the subgoals including the main goal
  - `fold_goals_tac` *defs*: fold *defs* throughout the subgoals
  - `fold_tac` *defs*: fold *defs* throughout the subgoals including the main goal
- induction
  - `induct_tac` *x i*: structural induction on variable *x* to subgoal *i*.
- assumption
  - `assume_tac` *i*: solve subgoal by assumption
  - `eq_assume_tac` *i*: similar, but does not use unification
- tableu prover
  - `blast_tac`: search proof using a fast tableu prover coded in ML
- automatic
  - `auto_tac`: prove trivial subgoals, leaves if it can not
- simplification
  - `simp_tac` *i*: simplify subgoal *i* using the simpset

II.2.5 Locales

Locales are a concept of local proof contexts. They are introduced as named syntactic objects within theories and can be opened in any descendant theory. A locale is declared in a theory section that starts with the keyword `locale`. It consists of three parts, the `fixes` part, the `assumes` part and the `defines` part.

The `fixes` part declares the locale constants. The `assumes` part specifies the locale rules. Locale rules admit the statement of local assumptions about the locale constants. The `defines` part introduces the definitions that are available in the locale. Locale constants declared in the `fixes` section are defined using the meta-equality.

II.3 Formal Concept Analysis

Formal Concept Analysis arose around 1980's as a result of a systematic framework development of lattice theory applications by a research group in Darmstadt University of Technology, Germany. It is a field of applied mathematics used for deriving implicit conceptual structures out of explicit knowledge.

Formal Concept Analysis is based on the mathematical formalization of the philosophical understanding of concept and conceptual hierarchy, borrowing mathematical foundations from lattice theory. A concept is constituted by two parts: its extension, which consists of all the objects belonging to the concept, and its intention which contains the attributes common to all objects of the concept. This formalization allows to form all concepts of a context and introduce a subsumption hierarchy between the concepts, giving a complete lattice called the concept lattice of the context.

Formal Concept Analysis looks at knowledge representation and processing from a mathematical order theoretic point of view. It allows graphical representation of structured knowledge as conceptual hierarchies and mathematical thinking for conceptual data analysis and knowledge processing. Knowledge is represented as concepts in the nodes of the concept lattice ordered by the subsumption relation, giving a taxonomy. Concept lattice is used to query the knowledge and to derive implicit information about the knowledge.

II.3.1 Formal Contexts and Concept Lattices

A triple (G, M, I) is called a **formal context** if G and M are sets, and $I \subseteq G \times M$ is a binary relation between G and M. Elements of G are called the **objects**, elements of M are called the **attributes** and I is called the **incidence relation** of the context (G, M, I).

A tuple (A, B) is called a **formal concept** of the context (G, M, I), if A is a subset of G, B is a subset of M, B is the set of attributes common to all objects in A and A is the set of objects which have all attributes in B.

Graphically, we represent a formal context as a cross-table. The rows of the table are headed by the object names and the columns are headed by the attribute names. A cross in row g and column m means that the object g has the attribute m.

The cross-table in table II.1 is the context of Living Beings and Water example in

Table II.1: Context of Living Beings

		a	b	c	d	e	f	g	h	i
1	Leech	x	x					x		
2	Bream	x	x					x	x	
3	Frog	x	x	x				x	x	
4	Dog	x		x				x	x	x
5	Spike - weed	x	x		x		x			
6	Reed	x	x	x	x		x			
7	Bean	x		x	x	x				
8	Maize	x		x	x		x			

a: needs water to live
b: lives in water
c: lives on land
d: needs chlorophyll to produce food
e: two seed leaves
f: one seed leaves
g: can move around
h: has limbs
i: suckles its offspring

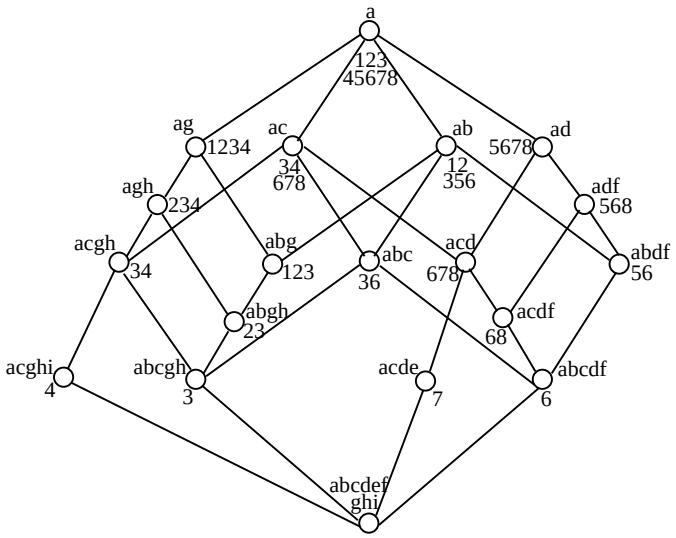

Figure II.2: Concept Lattice of the context "Living Beings and Water"

[13]. The context has 19 concepts. Figure II.2 shows the concept lattice. In the concept lattice, the extent (objects) of a concept is given as numbers and the intent (attributes) of a concept is given as letters.

II.4 Related Work

There are numerous formalization of mathematics efforts in Isabelle. Some of the important efforts are:

- Tobias Nipkow has proved the Church-Rosser Theorem for the lambda-calculus using a novel formalization, covering both the beta and the eta rules.
- Krzysztof Grabczewski has mechanized the first two chapters of Equivalents of

the Axiom of Choice by Rubin and Rubin, in ZF.

- Jacques Fleuriot has mechanized the ultrapower construction of the hyperreals from Nonstandard Analysis (NSA) in Isabelle. Concepts from NSA and geometry theorem proving have been combined and applied to the mechanization of Propositions from Newton's Principia for his PhD thesis. This includes the famous Proposition Kepleriana. The framework has also been used to mechanize real analysis using nonstandard techniques.

- Jacob Frost has mechanized Milner & Tofte's coinduction example in both HOL and ZF.

- Tobias Nipkow has mechanized the first 100 pages of Winskel's The Formal Semantics of Programming Languages, in HOL. This project extends the work of Ltzbeyer and Sandner.

- Jeremy Avigad, David Gray, and Adam Kramer are working to develop number theory in Isabelle, and have formalized Gauss' law of quadratic reciprocity.

Apart from formalized mathematics efforts, there are various verification projects.

- Tobias Nipkow, Cornelia Pusch, David von Oheimb, Gerwin Klein, Leonor Prensa Nieto, Martin Strecker, Norbert Schirmer, and Martin Wildmoser have formalized large parts of the programming language Java and the Java Virtual Machine. The project is developed Bali and Verificard at Technical University of Munich.

- Tobias Nipkow has verified an abstract lexical analyzer generator turning regular expressions into automata.

- Tobias Nipkow and Leonor Prensa Nieto have implemented the Owicki/Gries method for the Hoare-style verification of concurrent programs in Isabelle/HOL.

- Larry Paulson has developed a new approach to the verification of cryptographic protocols. The operational semantics of all agents in the network (including an active intruder) is modelled using a series of inductive definitions.

- Starting from an operational semantics for Prolog, Cornelia Pusch presents some refinement steps towards the Warren Abstract Machine (WAM). The correctness

and completeness proofs for each step have been elaborated with the theorem prover Isabelle using the logic HOL.

- Simon Thompson and Steve Hill have used Isabelle to reason about functional programs written in Miranda.

More information is availabe at Isabelle Web Site (`http://www.cl.cam.ac.uk/Research/HVG/Isabelle/projects.html`).

CHAPTER III

FORMALIZATION

In the commentary of the formalization, proofs appearing in the book are presented in the *Proof* environment, and formalization steps are given enumerated with the step numbers. For the fundamental lemmata, we first give the proof in the book, then give its formalization in Isabelle explaining the details of formalization. Proof and formalization of the basic theorem are given in two parts, again in an interleaved manner.

III.1 Definitions and Data Types

Definition 1. *A* **Formal Context** $\mathbb{K} := (G, M, I)$ *consists of two sets* G *and* M *and a relation* I *between* G *and* M. *The elements of* G *are called the* **objects** *and the elements of* M *are called the* **attributes** *of the context. The* I *relation between an object* g *and an attribute* m *is written as* gIm *or* $(g, m) \in I$ *and read as "the object* g **has** *the attribute* m". *The relation* I *is also called the* **incidence relation** *of the context.*

In the formalization, a formal context type is represented as a record type with fields `object_set`, `attribute_set` and `incidence_rel` as:

```
record ('a,'b) formal_context_type =
    object_set      :: "'a set"
    attribute_set   :: "'b set"
    incidence_rel   :: "('a * 'b) set"
```

Object set if of type `"'a set"`, attribute set if of type `"'b set"`, and the incidence relation type is set of product of types `'a` and `'b` where `'a` and `'b` are type variables.

Definition 2. *A* **Formal Concept** *of the context* $\mathbb{K} := (G, M, I)$ *is a pair* (A, B) *with* $A \subseteq G$, $B \subseteq M$, $A' = B$ *and* $B' = A$.*A is called the* **extent** *and* B *is called the* **intent** *of the concept* (A, B).

A formal concept type is represented as a record type with fields `extent` and `intent` as:

```
record ('a,'b) formal_concept_type =
    extent          :: "'a set"
    intent          :: "'b set"
```

The restrictions on sets A and B will be given later in the theory.

The `object_set`, `attribute_set` and `incidence_rel` fields of a formal context K are accessed as $K.<OS>$, $K.<AS>$, $K.<IR>$ and `extent` and `intent` fields of a concept C are accessed as $C.<E>$ and $C.<I>$ respectively, following the notation in the text book through the syntactic translations:

```
translations
    "formal_context.<OS>"   == "object_set formal_context"
    "formal_context.<AS>"   == "attribute_set formal_context"
    "formal_context.<IR>"   == "incidence_rel formal_context"
    "formal_concept.<E>"    == "extent formal_concept"
    "formal_concept.<I>"    == "intent formal_concept"
```

III.2 Main Functions

Definition 3. *For a set* $A \subseteq G$, *the set of attributes common to the objects in* A *is defined as:*

$$A' = \{m \in M \mid (g, m) \in I \; for \; all \; g \in A\}$$

This operation is formalized as the function `common_attributes` which takes an object set `os` and a context `K` and returns set of attributes common to the objects in this set as:

```
common_attributes :: "'a set => ('a,'b) formal_context_type =>
   'b set"
"common_attributes os fc == {
```

```
   m . m : fc.<AS> & (! g : os . (g,m) : fc.<IR>) & os <= fc.<OS>
}"
```

Definition 4. *Correspondingly, for a set* $B \in \subseteq M$*, the set of objects which have all attributes in is defined as* B*:*

$$B' = \{g \in G \mid (g,m) \in I \, for \, all \, m \in B\}$$

This operation is formalized as the function `common_objects` which takes an attribute set `as` and a context `K` and returns the set of objects which have all of these attributes as:

```
common_objects :: "'b set => ('a,'b) formal_context_type =>
   'a set"
"common_objects as fc == {
   g . g : fc.<OS> & (! m : as . (g,m) : fc.<IR>) & as <= fc.<AS>
}"
```

The polymorphic "$\prime$" operator is formalized as two separate functions to avoid confusion.

Given a tuple `C` of type `formal_concept_type` and a triple `K` of type `formal_context_type`, the predicate `FormalConcept` checks if `C` is a formal concept of `K` according to the restrictions given in the definition of formal concept above:

```
FormalConcept :: "('a,'b) formal_concept_type =>
   ('a,'b) formal_context_type => bool"
"FormalConcept C K == C.<E> <= K.<OS> & C.<I> <= K.<AS> &
   C.<E> = common_objects (C.<I>) K &
   common_attributes (C.<E>) K = C.<I>"
```

Definition 5. *If* (A_1, B_1) *and* (A_2, B_2) *are concepts of a context,* (A_1, B_1) *is called a* **subconcept** *of* (A_2, B_2)*, provided that* $A_1 \subseteq A_2$ *(which is equivalent to* $B_2 \subseteq A_2$*).In this case,* (A_2, B_2) *is a* **superconcept** *of* (A_1, B_1) *and the ordering is written as* $(A_1, B_1) \leq (A_2, B_2)$*.The relation* $\leq$ *is called the* **hierarchical order** *(or simply* **order***) of the concepts.The set of all concepts of* (G, M, I) *ordered in this way is denoted by* $\underline{\mathfrak{B}}(G, M, I)$ *and is called the* **Concept Lattice** *of the context* (G, M, I)*.*

The concept lattice of a context `K` is formalized with the function `ConceptLattice` as:

```
ConceptLattice :: "('a,'b) formal_context_type =>
   ((('a,'b) formal_concept_type) potype)"
"ConceptLattice K == (|
   pset = {C . (FormalConcept C K)},
   order = { (C1,C2) . FormalConcept C1 K & FormalConcept C2 K &
      C1.<E> <= C2.<E> & C2.<I> <= C1.<I> } |)"
```

It takes a context `K` and returns the concept lattice as a partial order type.Partial order type comes from the underlying lattice theory used.Its field `pset` is the set of objects in the lattice, `order` is the ordering of these objects as pairs like (a, b) denoting that $a \leq b$.

The locale `concept_lattice` gives the axioms of the theory about universal set.Object and attribute sets of the context are taken as the subsets of the universal set. Common attributes of an empty object set and common objects of an empty attribute set are also equal to the universal set.

```
locale concept_lattice = CL +
   fixes
      K :: "('a,'b) formal_context_type"
      S :: "(('a,'b) formal_concept_type) set"
   assumes
      ax1 "(INTER {} extent) == (K.<OS>)"
      univ_ax1 "UNIV == K.<OS>"
      univ_ax2 "UNIV == common_objects {} K"
      univ_ax3 "UNIV == K.<AS>"
      univ_ax4 "UNIV == common_attributes {} K"
```

III.3 Basic Lemmata

Lemma 1. *If (G, M, I) is a context, $A, A_1, A_2 \subseteq G$ are sets of objects and $B, B_1, B_2 \subseteq M$ are sets of attributes, then*

1) $A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1'$ 1′) $B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1'$

2) $A \subseteq A''$ 2′) $B \subseteq B''$

3) $A' = A'''$ 3′) $B' = B'''$

4) $A \subseteq B' \Leftrightarrow B \subseteq A' \Leftrightarrow A \times B \subseteq I$

Proof. 1) If $m \in A_2'$, then gIm for all $g \in A_2$, i.e., in particular gIm for all $g \in A_1$, if $A_1 \subseteq A_2$ and thus $m \in A_1'$.

2) If $g \in A$, then gIm for all $m \in A'$, which implies $g \in A''$

3) $A' \subseteq A'''$ follows immediately from 2′), and $A \subseteq A''$ together with 1) yields $A''' \subseteq A'$.

4) follows directly from the definition. □

Each part of the lemma is formalized as a separate lemma.(1) is proved with the `auto` tactic, which can do simple set theoretic operations and solve the goal in this case,given the definition of the `common_attributes` function as:

```
Goal "[| A1 <= K.<OS> ; A2 <= K.<OS> ; A1 <= A2 |] ==>
  (common_attributes A2 K) <= (common_attributes A1 K)";
  by (auto_tac (claset(), simpset() addsimps [common_attributes_def]));
qed "proposition_10_1";
```

Similarly, (2), (3), (1′), (2′), (3′), are proved with definitions of the `common_attributes` and `common_objects` functions whose formalization are given on page 39.

Lemma 2. *If T is an index set and, for every $t \in T$, $A_t \subseteq G$ is a set of objects, then*

$$\left(\bigcup_{t \in T} A_t\right)' = \bigcap_{t \in T} A_t'$$

The same holds for the sets of attributes.

Proof.

$$\begin{aligned} m \in \left(\bigcup_{t \in T} A_t\right)' &\Longleftrightarrow gIm\ for\ all\ g \in \bigcup_{t \in T} A_t \\ &\Longleftrightarrow gIm\ for\ all\ g \in A_t\ for\ all\ t \in T \\ &\Longleftrightarrow m \in A_t'\ for\ all\ t \in T \\ &\Longleftrightarrow m \in \bigcap_{t \in T} A_t' \end{aligned}$$

□

In the proof, the case that the index set T can be empty is not worked on explicitly.But in the formalization we need to do a case analysis for T being empty or not, since the set theory does not "know" what the prime of the empty set is.This case is handled separately with `univ_ax4` given above in the `locale` definition which states that common attributes of the empty set is equal to the universal set.`F` is an arbitrary mapping from an index to an attribute set. The formalization is given as `proposition_11_1` on page 40. Correspondingly, the same property for the sets of attributes is formalized as `proposition_11_2` on page 40. This time, the case index set T can be empty is handled with axiom `univ_ax2` which states that common objects of an empty attribute set is again equal to the universal set.Similarly, `H` is an arbitrary mapping from index set to an object set.

III.4 Basic Theorem on Concept Lattices

Theorem 1 (The Basic Theorem on Concept Lattices, part 1). *The concept lattice* $\underline{\mathfrak{B}}(G, M, I)$ *is a complete lattice in which infimum and supremum are given by:*

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right)$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t, \right)'' , \bigcap_{t \in T} B_t \right)$$

Proof. The formula for the infimum is derived as follows: Since $A_t = B_t'$ for each $t \in T$,

$$\left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right)$$

by Lemma 2 on page 19 can be transformed into

$$\left(\left(\bigcup_{t \in T} B_t \right)', \left(\bigcup_{t \in T} B_t \right)'' \right)$$

i.e., it has the form (X', X'') and is therefore certainly a concept. That this can only be the infimum, i.e., the largest common subconcept of the concepts (A_t, B_t), follows directly from the fact that the extent of this concept is exactly the intersection of the extents of (A_t, B_t). The formula for the supremum is substantiated correspondingly. Thus, we have proven that $\underline{\mathfrak{B}}(G, M, I)$ is a complete lattice. □

The formalization steps show the proof in more detail.

Proof. Let $\alpha = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t\right)''\right)$ be the infimum of the set S, subset of the concept lattice $\underline{\mathfrak{B}}(G, M, I)$. Since the infimum should be in the lattice, we need to show that α is in the concept lattice, that is, it is a formal concept of the context (G, M, I).This is proved by the auxiliary lemma `aux_lm1` in Appendix B on page 42.The lemma uses the correspondence of the property for attributes given in Lemma 2 for objects.It is formalized as `proposition_11_2` on page 40.

Next we need to show that α is a lower bound.That is, the extent $\bigcap_{t \in T} A_t$ of α is smaller than all other extents. This is proved automatically with set theoretic operations (by tactic `auto_tac`) given the definitions of functions `common_objects` and `common_attributes`.

Then we show that this is the greatest lower bound. Similarly that is also proved automatically. The whole proof script is given as `inf_cl` on page 43. □

The statement about the supremum of the concept lattice is proved correspondingly as follows:

Proof. Let $\alpha = \left(\left(\bigcup_{t \in T} A_t,\right)'', \bigcap_{t \in T} B_t\right)$ be the supremum of the set S subset of the concept lattice $\underline{\mathfrak{B}}(G, M, I)$. We show that α is in the concept lattice by the auxiliary lemma `aux_lm2` on page 44. The lemma uses the property stated as 2 which is formalized as `aux_lm2` on page 44.

Next we need to show that α is an upper bound, i.e. the intent $\bigcap_{t \in T} B_t$ is smaller than all other intents.This is proved again automatically with set theoretic operations (by tactic `auto_tac`) given the definitions of functions `common_objects` and `common_attributes`.

The case that α is least is again proved automatically. The whole proof script is given as `sup_cl` on page 45. □

Theorem 2 (The Basic Theorem on Concept Lattices, part 2). *A complete lattice* $\mathbf{V}$ *is isomorphic to* $\underline{\mathfrak{B}}(G, M, I)$ *if and only if there are mappings* $\widetilde{\gamma} : G \to V$ *and* $\widetilde{\mu} : M \to V$ *such that* $\widetilde{\gamma}(G)$ *is supremum-dense in* $\mathbf{V}$, $\widetilde{\mu}(M)$ *is infimum-dense in* $\mathbf{V}$ *and* gIm *is equivalent to* $\widetilde{\gamma}g \leq \widetilde{\mu}m$ *for all* $g \in G$ *and all* $m \in M$.*In particular,* $\mathbf{V} \cong \underline{\mathfrak{B}}(\mathbf{V}, \mathbf{V}, \leq)$

Proof. Now we prove, first for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$, the existence of the

mappings $\widetilde{\gamma}$ and $\widetilde{\mu}$ with the required properties.We set

$$\widetilde{\gamma}g := (\{g\}'', \{g\}') \, for \, g \in G \text{ and } \widetilde{\mu}m := (\{m\}', \{m\}'') \, for \, m \in M$$

As claimed, we have $\widetilde{\gamma}g \leq \widetilde{\mu}m \iff \{g\}'' \subseteq \{m\}' \iff \{g\}' \supseteq \{m\} \iff m \in \{g\}' \iff gIm$. Furthermore, on account of the formulas proved above,

$$\bigvee_{g \in A} (\{g\}'', \{g\}') = (A, B) = \bigwedge_{m \in B} (\{m\}', \{m\}'')$$

holds for every concept (A, B), i.e., $\widetilde{\gamma}(G)$ is supremum-dense and $\widetilde{\mu}(M)$ is infimum-dense in $\underline{\mathfrak{B}}(G, M, I)$. □

Now formalize this much of the theorem. We start with the *only if* direction of the double implication. In this direction, we show that if a complete lattice $\mathbf{V}$ is isomorphic to a concept lattice $\underline{\mathfrak{B}}(G, M, I)$, then there are mappings $\widetilde{\gamma} : G \to V$ and $\widetilde{\mu} : M \to V$ such that $\widetilde{\gamma}(G)$ is supremum-dense in $\mathbf{V}$, $\widetilde{\mu}(M)$ is infimum-dense in $\mathbf{V}$ and gIm is equivalent to $\widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$. Setting $\widetilde{\gamma}g := (\{g\}'', \{g\}')$ and $\widetilde{\mu}m := (\{m\}', \{m\}'')$, first we prove this for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$.

Step 1 We start with the formalization of $\widetilde{\gamma}(G)$ is supremum-dense in $\underline{\mathfrak{B}}(G, M, I)$. This is formalized as a separate lemma.

Step 1.1 For this, we first show that image of G under $\widetilde{\gamma}$ is a subset of the concept lattice $\underline{\mathfrak{B}}(G, M, I)$.That is; we need to show that for all g in G, $\widetilde{\gamma}g := (\{g\}'', \{g\}')$ is in the concept lattice, that is; it is a formal concept of the context (G, M, I). This is proved by the lemma `fc_dp_p` on page 41 which says that for any object g in G, $(\{g\}'', \{g\}')$ is a formal concept.

Step 1.2 Then we show that every element of $\underline{\mathfrak{B}}(G, M, I)$ can be written as the supremum of a subset of $\widetilde{\gamma}(G)$. For this, we pick an arbitrary concept (A, B) from $\underline{\mathfrak{B}}(G, M, I)$, and show that there exists a subset of $\widetilde{\gamma}(G)$ whose supremum is (A, B).We take this subset as $\widetilde{\gamma}(A)$.

Step 1.2.1 We first show that $\widetilde{\gamma}(A)$ is a subset of $\widetilde{\gamma}(G)$ by simplification and set theoretic operations from the fact that $A \subseteq G$ (with `full_simp_tac` and `auto_tac` tactics given the definitions of `FormalConcept` and `ConceptLattice` functions).

Step 1.2.2 Then we need to show that (A, B) is the supremum of $\widetilde{\gamma}(A)$. To be able to use the formula proved above for the supremum of a subset of a concept lattice, we rewrite the concept (A, B) as $\left(\left(\bigcup_{(X,Y)\in\widetilde{\gamma}(A)} X,\right)'', \bigcap_{(X,Y)\in\widetilde{\gamma}(A)} Y\right)$.This equivalence is proved in lemma `aux_lm6` on page 49. Rewriting the concept in this form, we prove that it is the supremum of $\widetilde{\gamma}(A)$ using the lemma `sup_cl` proved above which is formalized on page 45.

Thus we showed that $\widetilde{\gamma}(A)$ is supremum-dense in $\widetilde{\gamma}(G)$ for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$. The proof is given as the lemma `gamma_sup_dense` on page 53 in the formalization.

Step 2 Next we show that $\widetilde{\mu}(M)$ is infimum-dense in $\underline{\mathfrak{B}}(G, M, I)$. Similarly, this is also proved a separate lemma.

Step 2.1 We start with showing that the image of M under $\widetilde{\mu}$ is a subset of the concept lattice $\underline{\mathfrak{B}}(G, M, I)$. For this, we need to show that for all m in M, $\widetilde{\mu}m := (\{m\}', \{m\}'')$ is in the concept lattice, that is; it is a formal concept of the context (G, M, I). This is proved by the auxiliary lemma `fc_p_dp` on page 41 which states that for any attribute m in M, $(\{m\}', \{m\}'')$ is a formal concept.

Step 2.2 Then we show that every element of $\underline{\mathfrak{B}}(G, M, I)$ can be written as the infimum of a subset of $\widetilde{\mu}(M)$. For this, we pick an arbitrary concept (A, B) from $\underline{\mathfrak{B}}(G, M, I)$, and show that there exists a subset of $\widetilde{\mu}(M)$ whose infimum is (A, B). We take this subset as $\widetilde{\mu}(B)$.

Step 2.2.1 We first show that $\widetilde{\mu}(B)$ is a subset of $\widetilde{\mu}(M)$. This follows from the fact that $B \subseteq M$ by simplification and set theoretic operations (with `full_simp_tac` and `auto_tac` given the definitions of the `FormalConcept` and `ConceptLattice` functions).

Step 2.2.2 Then we need to show that (A, B) is the infimum of $\widetilde{\mu}(B)$. To be able to use the formula proved above for the infimum of a subset of a concept lattice, we rewrite the concept (A, B) in the form of $\left(\bigcap_{(X,Y)\in\widetilde{\mu}(B)} X, \left(\bigcup_{(X,Y)\in\widetilde{\mu}(B)} Y,\right)''\right)$. This equivalence is proved in lemma `aux_lm6_prime` on page 49.

Thus we showed that $\widetilde{\mu}(B)$ is infimum-dense in $\widetilde{\mu}(M)$ for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$. The proof is given as the lemma `mu_inf_dense` on page 54 in the formalization.

Step 3 Next we show that gIm is equivalent to $\widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$.To show the equivalence, we need to show both sides of it.

Step 3.1 We start the formalization with the *only if* direction.In this direction we need to show that $gIm \longrightarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$.Simplifying with the definition of the ordering in the concept lattice (by `simp_tac` given the definition of `ConceptLattice` function), we get the subgoals:

Step 3.1.1 $\widetilde{\gamma}g$ is a formal concept of (G, M, I).This is proved with lemma `gamma_fc` on page 50 which states that for an object g in G, $\widetilde{\gamma}g$ is a formal concept of (G, M, I).

Step 3.1.2 $\widetilde{\mu}m$ is a formal concept of (G, M, I).Similarly, this is proved with lemma `mu_fc` on page 50 which states that for an attribute m in M, $\widetilde{\mu}m$ is a formal concept of (G, M, I).

Step 3.1.3 $\{g\}'' \subseteq \{m\}'$.This is solved in one step automatically with set theory operations (by the `auto_tac` tactic) given the definitions of the functions `common_objects`, `common_attributes`, `gamma` and `mu`.

Step 3.2 Next we show the *if* direction of the equivalence.In this direction we need to show that $gIm \longleftarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$. Simplifying the assumption $\widetilde{\gamma}g \leq \widetilde{\mu}m$ with the definition of the ordering in the concept lattice (by `full_simp_tac` given the definitions of `ConceptLattice`, `gamma` and `mu` functions), we get $\{g\}'' \subseteq \{m\}'$. From this, we prove that gIm holds in one step with simplification of `common_objects common_attributes` functions and set theory operations (by `auto_tac` tactic).

Thus we showed that gIm is equivalent to $\widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$.

Having solved these 3 subgoals and intermediate subgoals resulting from them ,we showed that if a complete lattice $\mathbf{V}$ is isomorphic to a concept lattice $\underline{\mathfrak{B}}(G, M, I)$, then there are mappings $\widetilde{\gamma} : G \to V$ and $\widetilde{\mu} : M \to V$ such that $\widetilde{\gamma}(G)$ is supremum-dense

in $\mathbf{V}$, $\widetilde{\mu}(M)$ is infimum-dense in $\mathbf{V}$ and gIm is equivalent to $\widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$ for the special case $\mathbf{V} = \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$.

Now we go on with the proof in the book

Proof. More generally, if $\mathbf{V} \cong \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$ and $\varphi : \underline{\mathfrak{B}}(G, M, I) \to \mathbf{V}$ is an isomorphism, we define $\widetilde{\gamma}$ and $\widetilde{\mu}$ by

$$\widetilde{\gamma}g := \varphi(\{g\}'', \{g\}') \, for \, g \in G \text{ and } \widetilde{\mu}m := \varphi(\{m\}', \{m\}'') \, for \, m \in M$$

The properties claimed for these mappings are proved in a similar fashion. □

Now we formalize the general case.Unfolding the definition of the `isomorphic` function, we get an existentially quantified map from $\underline{\mathfrak{B}}(G, M, I)$ to $\mathbf{V}$. We rename it as φ and instantiate the existentially quantified maps $\widetilde{\gamma}$ and $\widetilde{\mu}$ as $\widetilde{\gamma}g := \varphi(\{g\}'', \{g\}') \, for \, g \in G$ and $\widetilde{\mu}m := \varphi(\{m\}', \{m\}'') \, for \, m \in M$ respectively.In this setting, we start the formalization.

Step 1 First, we prove that $\widetilde{\gamma}(G)$ is supremum-dense in $\mathbf{V}$. Here we can use the proof we made in the special case on page 22 by generalizing it. There we showed that the image of G under the map $\widetilde{\gamma}g := (\{g\}'', \{g\}') \, for \, g \in G$ is supremum-dense in $\underline{\mathfrak{B}}(G, M, I)$. This time, it will be enough to show that this property is preserved under the isomorphism φ. This is proved using the lemma `sup_dense_preserved` on page 51 which states that if a set is supremum-dense in a lattice, then the image of the set under an isomorphism is also supremum-dense in the isomorphic image of the lattice. We are given that $\mathbf{V}$ is the isomorphic image of $\underline{\mathfrak{B}}(G, M, I)$. Thus, we show that $\widetilde{\gamma}(G)$ is supremum-dense in $\mathbf{V}$.

Step 2 Now we need to show that $\widetilde{\mu}(M)$ is infimum-dense in $\mathbf{V}$. In the special case, we showed that the image of M under the map $\widetilde{\mu}m := (\{m\}', \{m\}'') \, for \, m \in M$ is infimum-dense in $\underline{\mathfrak{B}}(G, M, I)$ on page 23. Using this property and lemma `inf_dense_preserved`, on page 52, we show that $\widetilde{\mu}(M)$ is infimum-dense in $\mathbf{V}$. The lemma states that, if a set is infimum-dense in a lattice, then the image of the set under an isomorphism is also infimum-dense in the isomorphic image of the lattice under this isomorphism.

Step 3 Next, we show that gIm is equivalent to $\widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and all $m \in M$. To show the equivalence, we need to show both sides of it.

Step 3.1 First, we show the *only if* direction. Similar to the special case, we show that $gIm \longrightarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and for all $m \in M$ in this direction. In the special case, we showed that $gIm \longrightarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for $\widetilde{\gamma}g := (\{g\}'', \{g\}')\, for\, g \in G$ and $\widetilde{\mu}m := (\{m\}', \{m\}'')\, for\, m \in M$ on page 24. Now we need to show the same property for these maps composed with the isomorphism φ which are $\widetilde{\gamma}g := \varphi(\{g\}'', \{g\}')\, for\, g \in G$ and $\widetilde{\mu}m := \varphi(\{m\}', \{m\}'')\, for\, m \in M$ respectively in the generalized case. Since φ is an isomorphism, it is order-preserving. Thus we show the required property by simplification (using the `simp_tac` tactic).

Step 3.2 Next we show the *if* direction. Again, similar to the special case, we show that $gIm \longleftarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for all $g \in G$ and for all $m \in M$. In the special case, we showed that $gIm \longleftarrow \widetilde{\gamma}g \leq \widetilde{\mu}m$ for $\widetilde{\gamma}g := (\{g\}'', \{g\}')\, for\, g \in G$ and $\widetilde{\mu}m := (\{m\}', \{m\}'')\, for\, m \in M$ on page 24. Now we need to show the same property for these maps composed with the isomorphism φ which are $\widetilde{\gamma}g := \varphi(\{g\}'', \{g\}')\, for\, g \in G$ and $\widetilde{\mu}m := \varphi(\{m\}', \{m\}'')\, for\, m \in M$ respectively in the generalized case. Since φ is an isomorphism, the thesis holds.

This direction of the theorem is proved seperately as the lemma `basic_theorem_fwd` on page 57.

Now that both the proof in the book and commentary of the formalization of the *only if* direction of the theorem are given, we proceed with the proof of the *if* direction of the theorem. Following the convention, we first give the proof in the book.

For this direction, the fact that isomorphism implies the order-embedding property is implicitly used in the book.But we will also prove it as a separate lemma in the formalization.

The proof of the *if* direction of the theorem proceeds as follows in the book:

Proof. If, conversely, $\mathbf{V}$ is a complete lattice and $\widetilde{\gamma} : G \to \mathbf{V}$, $\widetilde{\mu} : \mathbf{M} \to \mathbf{V}$ are mappings with the properties stated above, then we define $\varphi : \underline{\mathfrak{B}}(G, M, I) \to \mathbf{V}$ by $\varphi(A, B) := \bigvee\{\widetilde{\gamma}(g) \,|\, g \in A\}$.

Evidently, φ is order-preserving. □

Now we formalize this much, and show how φ is order-preserving is proved in the

formalization.

Step 1 To show that φ is order-preserving, we need to show that for any two concepts (A_1, B_1) and (A_2, B_2) in the concept lattice $\underline{\mathfrak{B}}(G, M, I)$, if $(A_1, B_1) \leq (A_2, B_2)$ holds in the concept lattice, then $\varphi(A_1, B_1) \leq \varphi(A_2, B_2)$ should also hold in the isomorphic image $\mathbf{V}$ of the concept lattice (from unfolding the definition of the `order_preserving` function with the `simp_tac` tactic). More clearly, what is to be shown is $(A_1, B_1) \leq (A_2, B_2) \longrightarrow \bigvee \widetilde{\gamma}(A_1) \leq \bigvee \widetilde{\gamma}(A_2)$.

Step 1.1 By simplification with the definition of the ordering in the concept lattice (using the `full_simp_tac` given the definition of the `ConceptLattice` function), what we need to show becomes $A_1 \subseteq A_2 \longrightarrow \bigvee \widetilde{\gamma}(A_1) \leq \bigvee \widetilde{\gamma}(A_2)$.

Step 1.2 From $A_1 \subseteq A_2$, we get $\widetilde{\gamma}(A_1) \subseteq \widetilde{\gamma}(A_2)$ using the functional property of $\widetilde{\gamma}$. With this, now what we need to show becomes $\widetilde{\gamma}(A_1) \subseteq \widetilde{\gamma}(A_2) \longrightarrow \bigvee \widetilde{\gamma}(A_1) \leq \bigvee \widetilde{\gamma}(A_2)$. This is proved as lemma `sup_lt_ss` on page 55 of the formalization. The lemma states that if a set A is a subset of B, then, the supremum of A is smaller than the supremum of B.

Thus we show that φ is order-preserving for the generalized case. Now we go on with the proof of the theorem in the book.

Proof. In order to prove that φ is an isomorphism, we have to demonstrate that φ^{-1} exists and is also order-preserving. Therefore we define

$$\psi x := (\{g \in G \,|\, \widetilde{\gamma} g \leq x\}, \{m \in M \,|\, x \leq \widetilde{\mu} m\}),$$

for $x \in \mathbf{V}$ and demonstrate that ψx is a concept of (G, M, I):

$$\begin{aligned} h \in \{g \in G \,|\, \widetilde{\gamma} g \leq x\} &\Leftrightarrow \widetilde{\gamma} h \leq x \\ &\Leftrightarrow \widetilde{\gamma} h \leq \widetilde{\mu} n \text{ for all n} \in \{m \in M \,|\, x \leq \widetilde{\mu} m\} \\ &\Leftrightarrow hIn \text{ for all n} \in \{m \in M \,|\, x \leq \widetilde{\mu} m\} \\ &\Leftrightarrow h \in \{m \in M \,|\, x \leq \widetilde{\mu} m\}'. \end{aligned}$$

The second condition follows correspondingly. We have defined a map $\psi : \mathbf{V} \to \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$, and now can read of directly from the definition that ψ is order-preserving. □

Now we formalize the order-preserving property of ψ. First we instantiate the existentially quantified map with a map $\psi : \underline{\mathfrak{B}}(G, M, I) \to \mathbf{V}$ given as $\psi x := (\{g \in G \mid \widetilde{\gamma} g \leq x\}, \{m \in M \mid x \leq \widetilde{\mu} m\})$ in the book.

Step 1 To show that it is order-preserving, we need to show that whenever $x \leq y$ holds for $x \in \mathbf{V}$ and $y \in \mathbf{V}$, then $\psi(x) \leq \psi(y)$ must also hold in the concept lattice $\underline{\mathfrak{B}}(G, M, I)$. Simplifying with the definition of ordering in the concept lattice (using the `simp_tac` given the definition of the function `ConceptLattice`) we get the following subgoals:

Step 1.1 First we need to show that $\psi(x)$ is a formal concept of the context (G, M, I). Since the proof is too long to be given here, it is given as the lemma `psi_fc` on page 62.

Step 1.2 Next, we need to show that $\psi(y)$ is also a formal concept of the context. Similarly, the this is proved by the lemma stated above.

Step 1.3 Now we need to show how the ordering is preserved. Since $\psi(x)$ and $\psi(y)$ are concepts, we are done if we can show that the extent of $\psi(x)$ is a subset of the extent of $\psi(y)$; that is we need to show that $\{g \in G \mid \widetilde{\gamma} g \leq x\} \subseteq \{g \in G \mid \widetilde{\gamma} g \leq y\}$.

Step 1.3.1 Simplifying with the definition of subset relation, (using `simp_tac` given `subset_def`), what we need to show becomes $g \in \{g \in G \mid \widetilde{\gamma} g \leq x\} \longrightarrow g \in \{g \in G \mid \widetilde{\gamma} g \leq y\}$. This means, if $\widetilde{\gamma} g \leq x$, then $\widetilde{\gamma} g \leq y$. This is proved by the transitivity by simplification (using the `PartialOrder` and `trans` definitions) since we already know that $x \leq y$.

Thus we show that $x \leq y \longrightarrow \psi(x) \leq \psi(y)$ for all x and y in $\mathbf{V}$.

Thus we show that ψ is order-preserving.

Now that we are done with the formalization of ψ is order-preserving, we proceed with the proof in the book.

Proof. Now we prove that $\varphi = \psi^{-1}$. We have

$$\varphi\psi x = \bigvee \{\widetilde{\gamma} g \mid g \in G, \widetilde{\gamma} g \leq x\} = x,$$

since $\widetilde{\gamma}(G)$ is supremum-dense in $\mathbf{V}$. On the other hand, $\varphi(A,B) = \bigwedge\{\widetilde{\mu}m \,|\, m \in B\}$, since $\widetilde{\mu}(M)$ is infimum-dense in $\mu(\mathbf{M})$ $\mathbf{V}$, and consequently

$$\begin{aligned}
\varphi\psi(A,B) &= \psi \bigwedge\{\widetilde{\mu}m \,|\, m \in B\} \\
&= (\{g \in G \,|\, \widetilde{\gamma}g \leq \bigwedge\{\widetilde{\mu}m \,|\, m \in B\}\}, \{\ldots\}') \\
&= (\{g \in G \,|\, \widetilde{\gamma}g \leq \widetilde{\mu}m \text{ for all } m \in B\}, \{\ldots\}') \\
&= (\{g \in G \,|\, gIm \text{ for all } m \in B\}, \{\ldots\}') \\
&= (B', B'') = (A,B).
\end{aligned}$$

If we choose for a complete lattice $\mathbf{V}$ specifically $G := V$, $M := V$, $I := \leq$ and $\widetilde{\gamma}$ as well as $\widetilde{\mu}$ to be the identity of V, we obtain $\mathbf{V} \cong \underline{\mathfrak{B}}(\mathbf{G}, \mathbf{M}, \mathbf{I})$. □

The last part of the proof shows that φ and ψ are inverse functions. In the formalization, this is proved by showing that ψ is both right and left inverse of φ.

Step 1 We start with showing that ψ is the right inverse; that is for all x in $\mathbf{V}$, $\varphi\psi x = x$. More clearly, $\forall x \in \mathbf{V}.\ \bigvee\widetilde{\gamma}(\{\mathbf{g} \in \mathbf{G} \,|\, \widetilde{\gamma}\mathbf{g} \leq \mathbf{x}\}) = \mathbf{x}$ Unfolding the definition of supremum (by `simp_tac` tactic given the definition of the `islub` function), we get 3 subgoals:

Step 1.1 First we need to show that $x \in \mathbf{V}$; that is the supremum of the set is in the lattice $\mathbf{V}$. This is proved from the assumptions in one step (using the `atac` tactic).

Step 1.2 Next we need to show that x is an upper bound, that is; it is bigger than or equal to all other elements of the set $\widetilde{\gamma}\{g \in G \,|\, \widetilde{\gamma}g \leq x\}$. This is obvious from the definition of the set and proved in one step (with the `simp_tac` tactic).

Step 1.3 Next, we need to show that x is the least upper bound, that is; it is the smallest of all upper bounds. We know that $\widetilde{\gamma}(G)$ is supremum-dense in $\mathbf{V}$. So, all elements of $\mathbf{V}$ can be written as the supremum of a subset of $\widetilde{\gamma}(G)$. Unfolding the definition of `supremum_dense` function (with the `simp_tac` tactic) and picking the arbitrary element from $\mathbf{V}$ as x (with `dres_inst_tac` tactic), we have x as the supremum of a subset (say A) of $\widetilde{\gamma}(G)$. Since x is the least upper bound of A, it is also the least upper bound of $\widetilde{\gamma}\{g \in G \,|\, \widetilde{\gamma}g \leq x\}$. Because upper bounds of A and $\widetilde{\gamma}\{g \in G \,|\, \widetilde{\gamma}g \leq x\}$ are the

same sets. Finally this is proved automatically with set theoretic operations (using the `auto_tac` tactic).

Thus we show that ψ is the right inverse of φ.

Step 2 Next we show that ψ is the left inverse of φ, that is; for all (A, B) in $\underline{\mathfrak{B}}(G, M, I)$, $\psi\varphi(A, B) = (A, B)$.More clearly,

$$\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) \,.\, (\{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\}, \{m \in M \,|\, \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\}) = (A, B).$$

(by `simp_tac` with definitions of φ and ψ maps given above). Equality of concepts is proved by showing that their extents and intents are equal to each other's.

Step 2.1 So we start with showing that

$$\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) \,.\, \{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\} = A$$

Equality of two sets is proved by showing that they are subsets of each other.

Step 2.1.1 First, we show that $\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) \,.\, A \subseteq \{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\}$ Picking an arbitrary concept (A, B) from $\underline{\mathfrak{B}}(G, M, I)$ and simplifying with the definition of subset relation (using `simp_tac` given `subset` definition), what we need to show becomes $\forall g \in A \,.\, g \in \{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\}$, that is, we need to show that $\forall g \in A \,.\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)$. This is proved automatically by set theoretic operations from the definition of the set in one step (using `auto_tac` tactic).

Step 2.1.2 Second, we show that $\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) \,.\, \{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\} \subseteq A$. Picking an arbitrary concept (A, B) from $\underline{\mathfrak{B}}(G, M, I)$ and simplifying with the definition of subset relation (using `simp_tac` given `subset` definition), what we need to show becomes $\forall g \in \{g \in G \,|\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\} \,.\, g \in A$, that is, we need to show that $\forall g \in G \,.\, \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A) \longrightarrow g \in A$. For this, we prove that $\bigvee \widetilde{\gamma}(A) = \bigwedge \widetilde{\mu}(B)$ (This equality is used in the book without giving its proof explicitly). From this, our subgoal becomes $\forall g \in G \,.\, \widetilde{\gamma}g \leq \bigwedge \widetilde{\mu}(B) \longrightarrow g \in A$. From this, we get $\forall g \in G \,.\, \forall m \in B \,.\, \widetilde{\gamma}g \leq \widetilde{\mu}(m) \longrightarrow g \in A$ using the definition of infimum. From the assumptions, we know that, $\forall g \in G$ and $\forall m \in M \; gIm = \widetilde{\gamma}g \leq \widetilde{\mu}m$ holds.In our case

since $\widetilde{\gamma}g \leq \widetilde{\mu}(m)$ holds for all $g \in G$ and for all $m \in B$, we get $\forall g \in G . \forall m \in B . \widetilde{\gamma}g \leq \widetilde{\mu}(m) \longrightarrow gIm$. From which we get $\forall g \in G . \forall m \in B . \widetilde{\gamma}g \leq \widetilde{\mu}(m) \longrightarrow g \in B'$. And from the definition of `common_objects` we get $\forall g \in G . \forall m \in B . \widetilde{\gamma}g \leq \widetilde{\mu}(m) \longrightarrow g \in A$, which is what we want to show.

Step 2.2 Next we show the equality $\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) . \{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\} = B$. Since this is a set equality, again we show it in two steps picking an arbitrary concept (A, B) in $\underline{\mathfrak{B}}(G, M, I)$.

Step 2.2.1 First we show that $B \subseteq \{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\}$. Simplifying with the definition of the subset relation, (using the `simp_tac tactic` with `subset_def`), we get $\forall m \in \{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\} . m \in B$ which means $\forall m \in B . \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m$. In previous steps, we showed that $\bigvee \widetilde{\gamma}(A) = \bigwedge \widetilde{\mu}(B)$. So we prove the thesis using this and simplification with the definition of infimum.

Step 2.2.2 Next we show the other side of the set inclusion which is $\{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\} \subseteq B$. With simplification (by `simp_tac` with `subset_def`), our subgoal becomes $\forall m \in M . \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m \longrightarrow m \in M$. From the definition of supremum, we get $\forall m \in M . \forall g \in A . \widetilde{\gamma}g \leq \widetilde{\mu}m \longrightarrow m \in M$. From the assumptions we know that $\forall g \in G$ and $\forall m \in M\, gIm = \widetilde{\gamma}g \leq \widetilde{\mu}m$ holds. Using this assumption, our subgoal becomes, $\forall m \in M . \forall g \in A . gIm \longrightarrow m \in M$. From the definition of `common_attributes`, if an attribute m is in I relation with all objects in A, then we say that m is in A' which is equal to B, since (A, B) is a concept. Thus we prove our thesis.

Having proved both sides of the set inclusion, we proved that $\forall (A, B) \in \underline{\mathfrak{B}}(G, M, I) . \{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\} = B$ holds.

Having proved both the extents and intents are equal, we proved that the concept (A, B) is equal to the concept

$$(\{g \in G \mid \widetilde{\gamma}g \leq \bigvee \widetilde{\gamma}(A)\}, \{m \in M \mid \bigvee \widetilde{\gamma}(A) \leq \widetilde{\mu}m\})$$

which is $\psi\varphi(A, B) = (A, B)$. That is, ψ is the left inverse of φ.

Having proved that ψ is both the left and right inverse of φ, we proved that ψ and φ are inverse maps.

Having proved that φ is order-preserving, ψ is order-preserving, and ψ and φ are inverse maps, we proved that φ is a lattice isomorphism, which is our initial goal. The whole proof is given as lemma `basic_theorem_bwd` on page 72.

CHAPTER IV

CONCLUSION

IV.1 Discussions

Although mathematics texts usually do not give the proofs in whole detail, they are understandable by human. In an informal proof, some details of the proof may be skipped since they are intuitive to human. But for a proof to be machine-checkable, every single step of it has to be stated formally. There should not be any minor gap between proof steps since the machine does not have intuition like human beings and the theorems will not be proved unless each minor gap in the proof is filled.

During my formalization, I noticed some of these kinds of gaps in the informal proofs in the book Formal Concept Analysis. In the lemmata which take union or intersection over an index set, the case these index sets can be empty are not handled explicitly. But during the formalization I had to take care of this case because Isabelle insisted on the proof of the case the index set is empty. Also, in the first chapter, notions like common attributes of an empty object set and common objects of an empty attribute set are not introduced explicitly. For connecting the proofs of the *only if* and *if* parts of the basic theorem, a lemma from lattice theory is used but it is not mentioned clearly, since it is supposedly well-known to mathematicians.

For my formalization I also followed the proof of basic theorem in formal concept analysis chapter of the book **Introduction to Lattices and Order** [17]. There, in the proof of *only if* direction of the basic theorem on page 71, there is an overlooked part. It is written that the statement 'gIm if and only if $\widetilde{\gamma}(g) \leq \widetilde{\mu}(m)$ is in $\underline{\mathfrak{B}}(G, M, I)$,

for all g in G and for all m in M' is proved in 3.7. But the proof in 3.7 corresponds to the proof of the statement in our special case, so it does not give the proof in the generalized $\underline{\mathfrak{B}}(G, M, I)$ is isomorphic to **L** case. I think this part of the proof should be generalized to the isomorphism case. This is an example of the utility of formalization in revealing hidden gaps in published proofs.

IV.2 Future Work

Isar extension [15, 16] of Isabelle provides an interpreted language environment of its own, which has been specifically tailored for the needs of theory and proof developments. Compared to raw ML, the Isabelle/Isar top-level provides a more robust and comfortable development platform, with proper support for theory development graphs, single step transactions with limited undo, etc. The Isabelle/Isar version of the Proof General Interface provides an adequate front-end for interactive theory and proof development in this advanced theorem proving environment through the Emacs editor.

The main purpose of the Isar language is to provide a conceptually different view on machine-checked proofs. Isar stands for "Intelligible semi-automated reasoning". Drawing from both the traditions of informal mathematical proof texts and high level programming languages, Isar offers a versatile environment for structured formal proof documents. Thus properly written Isar proofs become accessible to a broader audience than unstructured tactic scripts (which typically only provide operational information for the machine).

As can be seen from the proof script in Appendix B, our formalization is in old-style tactic scripts format. At the time we started formalization Isabelle theory library did not contain the Isar version of the lattice theory we used in our formalization as the parent theory. Also, locales concept of Isabelle was not fully supported. But the Isabelle release Isabelle2003 announced in May 2003 provides both of these properties. Possible future work is to formalize Formal Concept Analysis using the Isar extension of Isabelle with structured human-readable proofs. Another possible future work may be to formalize chapters 2 and 3 of the book. Some sections like many-valued contexts in chapter 1 may be left out since they are not crucial for the theory. Chapter 2 of the book discusses topics like determination of all concepts of a context, and implications

and dependencies between attributes which are important parts of the theory. Chapter 3 discusses subcontexts, complete congruences and closed subrelations which are also important for the theory.

REFERENCES

[1] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411-440, 1993. Annotated version of the 1969 manuscript.

[2] Michael J. C. Gordon. From LCF to HOL: a short history. *Proof, Languages and Interaction*, Plotkin, Stirling, Tofte. MIT Press, 2000; ISBN 0262161885.

[3] R. Milner. Logic for computable functions descriptions of a mahine implementation. Technical Report STAN-CS-72-288, A.I. Memo 169, Stanford University, 1972.

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

[5] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994. LNCS 828. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361-386. Academic Press, 1990.

[6] de Bruijn, N. G. *The mathematical language AUTOMATH, its usage and some of its extensions*, 1970. Symposium on Automatic Demonstration, Volume 125 of Lecture Notes in Mathematics, Springer-Verlag.

[7] van Bentham Jutting, L. S. *Checking Landau's Grundlagen in the AUTOMATH System*, 1977 Ph. D. thesis, Eindhoven University of Technology.

[8] Nederpelt, R. P., Geuvers, J. H., and de Vrijer, R. C. *Selected Papers on Automath* 1994, Volume 133 of Studies in Logic and the Foundations of Mathematics. North-Holland.

[9] John Harrison *Formalized Mathematics*, 1996. `http://www.rbjones.com/rbjpub/logic/jrh0100.htm`

[10] Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. *A Proof Assistant for Higher-Order Logic*. Springer LNCS 2283, 2002.

[11] Lawrence C. Paulson *The Isabelle Reference Manual*. `http://isabelle.in.tum.de/doc/ref.pdf`

[12] Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. *Isabelle's Logics: HOL*. `http://isabelle.in.tum.de/doc/logics-HOL.pdf`

[13] Bernhard Ganter, Rudolf Wille. *Formal Concept Analysis*. Springer, 1999; ISBN 3540627715

[14] Bernhard Ganter, Rudolf Wille. *Applied Lattice Theory: Formal Concept Analysis*, 1997. http://www.math.tu-dresden.de/~ganter/concept.ps

[15] Markus Wenzel. *Isabelle/Isar Reference Manual* http://isabelle.in.tum.de/doc/isar-ref.pdf.

[16] Markus Wenzel. *Isabelle/Isar - a versatile environment for human-readable formal proof documents*. PhD thesis, Institut für Informatik, Technische Universität München, 2002. http://tumb1.biblio.tu-munchen.de/publ/dis/in/2002/wenzel.html

[17] B.A. Davey and H.A. Priestley.*Introduction to Lattices and Order* second edition, Cambridge University Press, 2002; ISBN 0521784514.

APPENDIX A

NOTATION INDEX

Table A.1: Notation Index

Math. Notation	Isabelle Notation	Definition
$'$ (*polymorphic*)	`common_attributes`	Common Attributes of an object set
$'$ (*polymorphic*)	`common_objects`	Common Objects of an attribute set
(G, M, I)	`K`	Context K
$\underline{\mathfrak{B}}(G, M, I)$	`(ConceptLattice K)`	Concept Lattice of the context K
$\bigwedge_{t \in T}(A_t, B_t)$	`(glb S K)`	Infimum of `S` in `K`
$\bigvee_{t \in T}(A_t, B_t)$	`(lub S K)`	Supremum of `S` in `K`
$\in$	`:`	In
$\wedge$	`&`	Conjunction
$\vee$	`|`	Disjunction
$\longrightarrow$	`-->`	Implication
$=$	`=`	Equality
$\forall t \in T. P(t)$	`! t : T . (P t)`	Universal Quantifier
$\exists t \in T. P(t)$	`? t : T . (P t)`	Existential Quantifier
$\subseteq$	`<=`	Subset or equal
$\bigcup_{t \in T} A_t$	`UN t : T . (F t)`	Indexed set union
$\bigcap_{t \in T} A_t$	`INT t : T . (F t)`	Indexed set intersection
λ	`%`	Lambda abstraction
$\cong$	`isomorphic`	Isomorphism

APPENDIX B

PROOF SCRIPT

```
Open_locale "concept_lattice";
Goal "[| A1 <= K.<OS> ; A2 <= K.<OS> ; A1 <= A2 |] ==>
 (common_attributes A2 K) <= (common_attributes A1 K)";
 by (auto_tac (claset(), simpset() addsimps [common_attributes_def]));
qed "proposition_10_1";


Goal "[| A <= K.<OS> |] ==> A <= (common_objects (common_attributes A K) K)";
 by (auto_tac (claset(), simpset() addsimps [common_objects_def,
 common_attributes_def]));
qed "proposition_10_2";
Goal "[| A <= K.<OS> |] ==> (common_attributes (common_objects (
 common_attributes A K) K) K) = (common_attributes A K)";
 by (auto_tac (claset(), simpset() addsimps [common_attributes_def,
 common_objects_def]));
qed "proposition_10_3";
Goal "[| B1 <= K.<AS> ; B2 <= K.<AS> ; B1 <= B2 |] ==> (common_objects B2 K)
 <= (common_objects B1 K)";
 by (auto_tac (claset(), simpset() addsimps [common_objects_def]));
qed "proposition_10_1_prime";
Goal "[| B <= K.<AS> |] ==> B <= (common_attributes (common_objects B K) K)";
 by (auto_tac (claset(), simpset() addsimps [common_attributes_def,
```

```
 common_objects_def]));
qed "proposition_10_2_prime";

Goal "[| B <= K.<AS> |] ==> (common_objects B K) = (common_objects (
 common_attributes (common_objects B K) K) K)";
 by (auto_tac (claset(), simpset() addsimps [common_objects_def,
 common_attributes_def]));
qed "proposition_10_3_prime";
Goal "[| ! t : T . (F t) <= K.<OS> |] ==> (common_attributes (
 UN t : T . (F t)) K) = (INT t : T . (common_attributes (F t) K))";
 by (case_tac "T = {}" 1);
 by (asm_simp_tac (simpset() addsimps [thm "univ_ax4"]) 1);
 by (auto_tac (claset(), simpset() addsimps [common_attributes_def]));
qed "proposition_11_1";

Goal "[| ! t : T . (H t) <= K.<AS> |] ==> (INT t : T .
 (common_objects (H t) K)) = (common_objects (UN t : T . (H t)) K)";
 by (case_tac "T = {}" 1);
 by (asm_simp_tac (simpset() addsimps [thm "univ_ax2"]) 1);
 by (auto_tac (claset(), simpset() addsimps [common_objects_def]));
qed "proposition_11_2";

Goal "[| FormalConcept x K |] ==> (x.<I>) = (common_attributes (x.<E>) K)";
 by (auto_tac (claset(), simpset() addsimps [FormalConcept_def]));
qed "aux_rwr1";
Goal "[| g : K.<OS> |] ==> (FormalConcept (| extent = common_objects (
 common_attributes {g} K) K, intent = common_attributes {g} K |) K)";
 by (simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
```

```
 [common_attributes_def])) 1);
 by (rtac proposition_10_3 1);
 by (asm_simp_tac (simpset()) 1);
qed "fc_dp_p";

Goal "[| m : K.<AS> |] ==> (FormalConcept (| extent = common_objects {m} K ,
 intent = common_attributes (common_objects {m} K) K |) K)";
 by (simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 by (rtac proposition_10_3_prime 1);
 by (asm_simp_tac (simpset()) 1);
qed "fc_p_dp";

Goal "[| ALL x : S . FormalConcept x K |] ==> FormalConcept (| extent =
 INTER S extent, intent = common_attributes (common_objects
 (UNION S intent) K) K |) K";
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (case_tac "S = {}" 1);
 by (asm_simp_tac (simpset() addsimps [thm "univ_ax1"]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 by (rtac conjI 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_2]) 1);
 by (rtac proposition_10_3_prime 1);
```

```
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_2]) 1);
qed "aux_lm1";

(*************************************************************************)
(*                  INFIMUM of CONCEPT LATTICE                           *)
(*************************************************************************)
Goal "[| S <= (ConceptLattice K).<A> |] ==> isglb S (ConceptLattice K)
 (| extent = (INT C : S . C.<E>) ,
 intent = (common_attributes (common_objects (UN C : S . C.<I>) K) K) |)";
 by (simp_tac (simpset() addsimps [isglb_def]) 1);
 by (rtac conjI 1);
 by (rewrite_goals_tac [subset_def]);
 by (ALLGOALS (full_simp_tac (simpset() addsimps [ConceptLattice_def])));
 (* 1st subgoal of isglb_def *)
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm1]) 1);
 (* 2nd subgoal of isglb_def, lower bound *)
 by (rtac conjI 1);
 by (clarify_tac (claset()) 1);
 by (rtac conjI 1);
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm1]) 1);
 by (rtac conjI 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 (* 3rd subgoal of isglb_def, greatest lower bound *)
 by (clarify_tac (claset()) 1);
 by (rtac conjI 1);
```

```
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm1]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (asm_simp_tac (simpset() addsimps [aux_rwr1]) 1);
 by (rtac proposition_10_1 1);
 by (ALLGOALS (full_simp_tac (simpset() addsimps [FormalConcept_def])));
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac proposition_10_1_prime 1);
 by (auto_tac (claset(), simpset()));
qed "inf_cl";

Goal "[| ALL x : S . FormalConcept x K |] ==> FormalConcept
 (| extent = common_objects (common_attributes (UNION S extent) K) K ,
 intent = INTER S intent |) K";
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (rtac conjI 1);
 by (case_tac "S = {}" 1);
 by (asm_simp_tac (simpset() addsimps [thm "univ_ax3"]) 1);
 by (force_tac (claset(),simpset()) 1);
 by (rtac conjI 1);
 by (full_simp_tac (simpset() addsimps [ball_conj_distrib]) 1);
 by (REPEAT (etac conjE 1));
 by (thin_tac "ALL x:S. x.<E> = common_objects (x.<I>) K" 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_1]) 1);
 by (full_simp_tac (simpset() addsimps [ball_conj_distrib]) 1);
 by (REPEAT (etac conjE 1));
```

```
 by (thin_tac "ALL x:S. x.<E> = common_objects (x.<I>) K" 1);
 by (subgoal_tac "ALL x : S . (x.<I>) = (common_attributes (x.<E>) K)" 1);
 by (thin_tac "ALL x : S . (common_attributes (x.<E>) K) = (x.<I>)" 1);
 by (ALLGOALS (asm_simp_tac (simpset() addsimps [proposition_11_1 RS sym])));
 by (rtac proposition_10_3 1);
 by (auto_tac (claset(), simpset()));
qed "aux_lm2";

(*************************************************************************)
(*                  SUPREMUM of CONCEPT LATTICE                          *)
(*************************************************************************)
Goal "[| S <= (ConceptLattice K).<A> |] ==> islub S (ConceptLattice K)
 (| extent = (common_objects (common_attributes (UN C : S . C.<E>) K) K) ,
 intent = (INT C : S . C.<I>) |)";
 by (simp_tac (simpset() addsimps [islub_def]) 1);
 by (rtac conjI 1);
 by (rewrite_goals_tac [subset_def]);
 by (ALLGOALS (full_simp_tac (simpset() addsimps [ConceptLattice_def])));
 (* 1st subgoal of islub_def *)
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm2]) 1);
 (* 2nd subgoal of islub_def, upper bound *)
 by (rtac conjI 1);
 by (clarify_tac (claset()) 1);
 by (rtac conjI 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac conjI 1);
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm2]) 1);
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
```

```
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 (* 3rd subgoal of islub_def, least upper bound *)
 by (clarify_tac (claset()) 1);
 by (rtac conjI 1);
 (* FormalConcept (|extent = ... , intent = ... |) *)
 by (asm_simp_tac (simpset() addsimps [aux_lm2]) 1);
 by (rtac conjI 1);
 by (ALLGOALS (full_simp_tac (simpset() addsimps [FormalConcept_def])));
 by (full_simp_tac (simpset() addsimps [ball_conj_distrib]) 1);
 by (REPEAT (etac conjE 1));
 by (thin_tac "ALL x:S. x.<E> = common_objects (x.<I>) K" 1);
 by (thin_tac "ALL y:S. y.<E> = common_objects (y.<I>) K" 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac proposition_10_1_prime 1);
 by (asm_simp_tac (simpset()) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_1]) 1);
 by (auto_tac (claset(), simpset()));
qed "sup_cl";

Goal "(ConceptLattice K) : CompleteLattice";
 by (simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
 by (rtac conjI 1);
 by (simp_tac (simpset() addsimps [PartialOrder_def]) 1);
 by (rtac conjI 1);
 (* reflexivity *)
 by (simp_tac (simpset() addsimps [refl_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def])) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def])) 1);
```

```
 by (rtac conjI 1);
 (* antisymmetry *)
 by (simp_tac (simpset() addsimps [antisym_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def,equalityI])) 1);
 (* transitivity *)
 by (simp_tac (simpset() addsimps [trans_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def])) 1);
 by (rtac conjI 1);
 (* supremum of arbitrary subset exists *)
 by (clarify_tac (claset()) 1);
 by (rtac exI 1);
 by (rtac sup_cl 1);
 by (asm_simp_tac (simpset()) 1);
 (* infimum of arbitrary subset exists *)
 by (clarify_tac (claset()) 1);
 by (rtac exI 1);
 by (rtac inf_cl 1);
 by (asm_simp_tac (simpset()) 1);
qed "cl_CL";
Open_locale "formal_context";
Goal "[| v : ConceptLattice K .<A> |] ==> (gamma ` (v.<E>)) <=
 (ConceptLattice K).<A>";
 by (simp_tac (simpset() addsimps [thm "gamma_def",subset_def]) 1);
 by (full_simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
 by (clarify_tac (claset()) 1);
 by (rtac fc_dp_p 1);
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (auto_tac (claset(), simpset()));
qed "aux_lm3";
Goal "[| v : ConceptLattice K .<A> |] ==> (mu ` (v.<I>)) <=
 (ConceptLattice K).<A>";
```

```
 by (simp_tac (simpset() addsimps [thm "mu_def",subset_def]) 1);
 by (full_simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
 by (clarify_tac (claset()) 1);
 by (rtac fc_p_dp 1);
 by (full_simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (auto_tac (claset(), simpset()));
qed "aux_lm3_prime";
Goal "[| C : (ConceptLattice K).<A> |] ==>
 (common_objects (common_attributes (C.<E>) K) K) = (C.<E>)";
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (asm_simp_tac (simpset() addsimps
 [proposition_10_3_prime RS sym]) 1);
qed "aux_lm4";
Goal "[| C : (ConceptLattice K).<A> |] ==>
 (common_attributes (common_objects (C.<I>) K) K) = (C.<I>)";
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (REPEAT (etac conjE 1));
 by (auto_tac (claset(), simpset() addsimps [proposition_10_3]));
qed "aux_lm4_prime";
Goal "[| v : (ConceptLattice K).<A> |] ==> v = (| extent =
 (common_objects (common_attributes (UNION (gamma ` (v.<E>)) extent) K) K) ,
 intent = (INTER (gamma ` (v.<E>)) intent) |)";
 by (subgoal_tac "! c : (gamma ` (v.<E>)) . (c.<E>) <= (K.<OS>)" 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_1]) 1);
 by (simp_tac (simpset() addsimps [thm "gamma_def"]) 1);
 by (subgoal_tac "! a : v.<E> . (common_attributes (common_objects
 (common_attributes {a} K) K) K) = (common_attributes {a} K)" 1);
 by (asm_simp_tac (simpset()) 1);
 by (subgoal_tac "! a : v.<E> . {a} <= K.<OS>" 1);
 by (SELECT_GOAL (fold_goals_tac [singleton_def]) 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_1 RS sym]) 1);
```

```
 by (simp_tac (simpset() addsimps [singleton_def]) 1);
 by (asm_simp_tac (simpset() addsimps [aux_lm4]) 1);
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (ALLGOALS (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def])));
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (clarify_tac (claset()) 1);
 by (rtac proposition_10_3 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (simp_tac (simpset() addsimps [thm "gamma_def"]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
qed "aux_lm5";

Goal "[| v : (ConceptLattice K).<A> |] ==> v = (| extent =
 (INTER (mu ‘ (v.<I>)) extent) , intent = (common_attributes
 (common_objects (UNION (mu ‘ (v.<I>)) intent) K) K) |)";
 by (subgoal_tac "! c : (mu ‘ (v.<I>)) . (c.<I>) <= (K.<AS>)" 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_2 RS sym]) 1);
 by (simp_tac (simpset() addsimps [thm "mu_def"]) 1);
 by (subgoal_tac "! a : v.<I> . (common_objects (common_attributes
 (common_objects {a} K) K) K) = (common_objects {a} K)" 1);
 by (asm_simp_tac (simpset()) 1);
 by (subgoal_tac "! a : v.<I> . {a} <= K.<AS>" 1);
 by (SELECT_GOAL (fold_goals_tac [singleton_def]) 1);
 by (asm_simp_tac (simpset() addsimps [proposition_11_2]) 1);
 by (simp_tac (simpset() addsimps [singleton_def]) 1);
 by (asm_simp_tac (simpset() addsimps [aux_lm4_prime]) 1);
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
```

```
 by (ALLGOALS (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def])));
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (clarify_tac (claset()) 1);
 by (rtac (proposition_10_3_prime RS sym) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (simp_tac (simpset() addsimps [thm "mu_def"]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
qed "aux_lm5_prime";
Goal "v : (ConceptLattice K).<A> ==> (UN a:v.<E>. (gamma a).<E>) = v.<E>";
 by (simp_tac (simpset() addsimps [thm "gamma_def"]) 1);
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def]));
qed "aux_lm6";

Goal "v : (ConceptLattice K).<A> ==> (UN a:v.<I>. (mu a).<I>) = v.<I>";
 by (simp_tac (simpset() addsimps [thm "mu_def"]) 1);
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (REPEAT (etac conjE 1));
 by (subgoal_tac "(common_objects (v.<I>) K) = v.<E>" 1);
 by (thin_tac "v.<E> = common_objects (v.<I>) K" 1);
 by (subgoal_tac "v.<I> = (common_attributes (v.<E>) K)" 1);
 by (thin_tac "(common_attributes (v.<E>) K) = v.<I>" 1);
 by (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def]));
qed "aux_lm6_prime";

Goal "[| g : K.<OS> |] ==> (FormalConcept (gamma g) K)";
 by (full_simp_tac (simpset() addsimps
```

```
 [ConceptLattice_def,FormalConcept_def,thm "gamma_def"]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_attributes_def])) 1);
 by (rtac proposition_10_3 1);
 by (asm_simp_tac (simpset()) 1);
qed "gamma_fc";

Goal "[| m : K.<AS> |] ==> (FormalConcept (mu m) K)";
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def,thm "mu_def"]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def])) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [common_objects_def,common_attributes_def])) 1);
 by (rtac proposition_10_3_prime 1);
 by (asm_simp_tac (simpset()) 1);
qed "mu_fc";

Goal " V = (| pset = V.<A> , order = V.<r> |)";
 by (simp_tac (simpset()) 1);
qed "aux_rwr2";
Goal "[| (isomorphic_thru V1 V2 phi) ; (supremum_dense B V1) |] ==>
 (supremum_dense (phi ` B) V2)";
 by (full_simp_tac (simpset() addsimps [isomorphic_thru_def,
 lattice_isomorphism_def,lattice_homomorphism_def,supremum_dense_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
```

```
 by (full_simp_tac (simpset() addsimps [supremum_preserving_def]) 1);
 by (REPEAT (etac conjE 1));
 by (full_simp_tac (simpset() addsimps [my_surj_def]) 1);
 by (rtac ballI 1);
 by (dres_inst_tac [("x","v")] bspec 1);
 by (atac 1);
 by (etac bexE 1);
 by (dres_inst_tac [("x","x")] bspec 1);
 by (atac 1);
 by (etac exE 1);
 by (res_inst_tac [("x","(phi ` A)")] exI 1);
 by (rtac conjI 1);
 by (thin_tac "phi ` (V1 .<A>) <= V2 .<A>" 1);
 by (thin_tac "B <= V1 .<A>" 1);
 by (thin_tac "phi ` (V1 .<A>) <= V2 .<A>" 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (thin_tac "phi ` (V1.<A>) <= V2.<A>" 1);
 by (asm_simp_tac (simpset()) 1);
 by (dres_inst_tac [("x","A")] spec 1);
 by (dtac mp 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (auto_tac (claset(), simpset()));
qed "sup_dense_preserved";

Goal "[| (isomorphic_thru V1 V2 phi) ; (infimum_dense B V1) |] ==>
 (infimum_dense (phi ` B) V2)";
 by (full_simp_tac (simpset() addsimps
 [isomorphic_thru_def,lattice_isomorphism_def,lattice_homomorphism_def,
 infimum_dense_def]) 1);
 by (rtac conjI 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (full_simp_tac (simpset() addsimps [infimum_preserving_def]) 1);
 by (REPEAT (etac conjE 1));
```

```
 by (full_simp_tac (simpset() addsimps [my_surj_def]) 1);
 by (rtac ballI 1);
 by (dres_inst_tac [("x","v")] bspec 1);
 by (atac 1);
 by (etac bexE 1);
 by (dres_inst_tac [("x","x")] bspec 1);
 by (atac 1);
 by (etac exE 1);
 by (res_inst_tac [("x","(phi ` A)")] exI 1);
 by (rtac conjI 1);
 by (thin_tac "phi ` (V1 .<A>) <= V2 .<A>" 1);
 by (thin_tac "B <= V1 .<A>" 1);
 by (thin_tac "phi ` (V1 .<A>) <= V2 .<A>" 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (thin_tac "phi ` (V1.<A>) <= V2.<A>" 1);
 by (asm_simp_tac (simpset()) 1);
 by (dres_inst_tac [("x","A")] spec 1);
 by (dtac mp 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (auto_tac (claset(), simpset()));
qed "inf_dense_preserved";

Goal "(supremum_dense (gamma ` (K.<OS>)) (ConceptLattice K))";
 by (simp_tac (simpset() addsimps [supremum_dense_def]) 1);
 by (rtac conjI 1);
 (* image of K.<OS> under gamma is in ConceptLattice K *)
 by (simp_tac (simpset() addsimps [thm "gamma_def"]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def])) 1);
 by (rtac fc_dp_p 1);
 by (asm_simp_tac (simpset()) 1);
 by (clarify_tac (claset()) 1);
 by (res_inst_tac [("x","(gamma ` (v.<E>))")] exI 1);
```

```
 by (rtac conjI 1);
 by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 (* existence of the supremum s.t ... *)
 by (subgoal_tac "islub (gamma ` (v.<E>)) (ConceptLattice K) (| extent =
 (common_objects (common_attributes (UNION (gamma ` (v.<E>)) extent) K) K) ,
 intent = (INTER (gamma ` (v.<E>)) intent) |)" 1);
 by (subgoal_tac "v = (| extent = (common_objects (common_attributes
 (UNION (gamma ` (v.<E>)) extent) K) K) , intent =
 (INTER (gamma ` (v.<E>)) intent) |)" 1);
 by (rtac (aux_lm5 RS ssubst) 1);
 by (asm_simp_tac (simpset()) 1);
 by (asm_full_simp_tac (simpset() addsimps [aux_lm6]) 1);
 by (rtac aux_lm5 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac sup_cl 1);
 by (asm_simp_tac (simpset() addsimps [aux_lm3]) 1);
qed "gamma_sup_dense";

Goal "(infimum_dense (mu ` (K.<AS>)) (ConceptLattice K))";
 by (simp_tac (simpset() addsimps [infimum_dense_def]) 1);
 by (rtac conjI 1);
 (* image of K.<AS> under mu is in ConceptLattice K *)
 by (simp_tac (simpset() addsimps [thm "mu_def"]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def])) 1);
 by (rtac fc_p_dp 1);
 by (asm_simp_tac (simpset()) 1);
 by (clarify_tac (claset()) 1);
 by (res_inst_tac [("x","(mu ` (v.<I>))")] exI 1);
 by (rtac conjI 1);
 by (full_simp_tac (simpset() addsimps
```

```
 [ConceptLattice_def,FormalConcept_def]) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 (* existence of the infimum s.t ... *)
 by (subgoal_tac "isglb (mu ` (v.<I>)) (ConceptLattice K) (| extent =
 (INTER (mu ` (v.<I>)) extent) , intent = (common_attributes (common_objects
 (UNION (mu ` (v.<I>)) intent) K) K) |)" 1);
 by (subgoal_tac "v = (| extent = INTER (mu ` (v.<I>)) extent ,
 intent = common_attributes (common_objects (UNION (mu ` (v.<I>))
 intent) K) K |)" 1);
 by (rtac (aux_lm5_prime RS ssubst) 1);
 by (asm_simp_tac (simpset()) 1);
 by (asm_full_simp_tac (simpset() addsimps [aux_lm6_prime]) 1);
 by (rtac aux_lm5_prime 1);
 by (asm_simp_tac (simpset()) 1);
 by (rtac inf_cl 1);
 by (asm_simp_tac (simpset() addsimps [aux_lm3_prime]) 1);
qed "mu_inf_dense";

Goal "[| V : CompleteLattice ; X <= V.<A> ; Y <= V.<A> ; X <= Y |] ==>
 ((lub X V),(lub Y V)) : V.<r>";
 by (subgoal_tac "! v : V.<A> . ((! x : X . (x,v) : V.<r>) -->
 ((lub X V),v) : V.<r>)" 1);
 by (subgoal_tac "! x : X . (x,(lub Y V)) : V.<r>" 1);
 by (dres_inst_tac [("x","(lub Y V)")] bspec 1);
 by (rtac (export lub_in_lattice) 1);
 by (atac 1);
 by (atac 1);
 by (etac impE 1);
 by (atac 1);
 by (atac 1);
 by (subgoal_tac "! y : Y . (y,(lub Y V)) : V.<r>" 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps [subset_def])) 1);
 by (rtac (export lubE1) 1);
```

```
 by (atac 1);
 by (atac 1);
 by (rtac ballI 1);
 by (clarify_tac (claset()) 1);
 by (rtac (export lubE2) 1);
 by (auto_tac (claset(), simpset()));
qed "sup_lt_ss";

Goal "[| V : CompleteLattice |] ==> (isomorphic (ConceptLattice K) V) -->
 (? gamma mu . (supremum_dense (gamma ` (K.<OS>)) V) &
 (infimum_dense (mu ` (K.<AS>)) V) & (! g : K.<OS> . ! m : K.<AS> . ((g,m) :
 K.<IR>) = (((gamma g),(mu m)) : V.<r>)))";
 by (rtac impI 1);
 by (full_simp_tac (simpset() addsimps [isomorphic_def]) 1);
 by (etac exE 1);
 by (rename_tac "phi" 1);
 by (res_inst_tac [("x","(phi o gamma)")] exI 1);
 by (rtac conjI 1);
 by (res_inst_tac [("x","(phi o mu)")] exI 2);
 (* supremum-dense proof *)
 by (simp_tac (simpset() addsimps [image_compose]) 1);
 by (res_inst_tac [("V1.0","(ConceptLattice K)")] sup_dense_preserved 1);
 by (atac 1);
 by (rtac gamma_sup_dense 1);
 by (rtac conjI 1);
 (* infimum-dense proof *)
 by (simp_tac (simpset() addsimps [image_compose]) 1);
 by (res_inst_tac [("V1.0","(ConceptLattice K)")] inf_dense_preserved 1);
 by (atac 1);
 by (rtac mu_inf_dense 1);
 by (rtac ballI 1);
 by (rtac ballI 1);
 by (rtac iffI 1);
```

```
(* property of the IR,  ==> direction *)
by (simp_tac (simpset() addsimps [image_compose]) 1);
by (full_simp_tac (simpset() addsimps [isomorphic_thru_def,
lattice_isomorphism_def,lattice_homomorphism_def,order_preserving_def]) 1);
by (REPEAT (etac conjE 1));
by (subgoal_tac "((gamma g),(mu m)) : (ConceptLattice K).<r>" 1);
by (dres_inst_tac [("x","(gamma g)")] bspec 1);
by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
by (rtac gamma_fc 1);
by (atac 1);
by (dres_inst_tac [("x","(mu m)")] bspec 1);
by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
by (rtac mu_fc 1);
by (atac 1);
by (dtac mp 1);
by (atac 1);
by (atac 1);
by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
by (rtac conjI 1);
by (rtac gamma_fc 1);
by (atac 1);
by (rtac conjI 1);
by (rtac mu_fc 1);
by (atac 1);
by (rtac conjI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps [common_objects_def,
common_attributes_def,thm "gamma_def",thm "mu_def"])) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps [common_objects_def,
common_attributes_def,thm "gamma_def",thm "mu_def"])) 1);
(* property of the IR,  <== direction *)
by (subgoal_tac "((gamma g),(mu m)) : (ConceptLattice K).<r>" 1);
by (subgoal_tac "gamma g.<E> <= mu m.<E>" 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps [common_objects_def,
```

```
 common_attributes_def,thm "gamma_def",thm "mu_def"])) 1);
 by (full_simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
 by (REPEAT (etac conjE 1));
 by (atac 1);
 by (res_inst_tac [("L","(ConceptLattice K)"),("a","(gamma g)"),
 ("b","(mu m)"),("phi","phi"),("V","V")] iso_imp_embd 1);
 by (atac 1);
 by (atac 1);
 by (full_simp_tac (simpset() addsimps [image_compose]) 1);
 by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
 by (rtac gamma_fc 1);
 by (atac 1);
 by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
 by (rtac mu_fc 1);
 by (atac 1);
qed "basic_theorem_fwd";

Goal "!! x y . [| V : CompleteLattice; supremum_dense (gamma ‘ (K.<OS>)) V;
 infimum_dense (mu ‘ (K.<AS>)) V; ! g : K.<OS>.  ! m : K.<AS>. ((g, m) :
 K.<IR>) = ((gamma g, mu m) : V .<r>); x : V .<A>; y : V .<A> |] ==>
 FormalConcept (| extent = {g. g : K.<OS> & (gamma g, x) : V .<r>},
 intent = {m. m : K.<AS> & (x, mu m) : V .<r>} |) K";
 (* proof of : {g. g : K.<OS> & (gamma g, x) : V .<r>}                *)
 (*            = common_objects {m. m : K.<AS> & (x, mu m) : V .<r>} K *)
 by (simp_tac (simpset() addsimps [FormalConcept_def]) 1);
 by (rtac conjI 1);
 by (simp_tac (simpset() addsimps [subset_def]) 1);
 by (rtac conjI 1);
 by (simp_tac (simpset() addsimps [subset_def]) 1);
 by (rtac conjI 1);
 (* proof of : {g. g : K.<OS> & (gamma g, x) : V .<r>}                *)
 (*         <= common_objects {m. m : K.<AS> & (x, mu m) : V .<r>} K   *)
 by (rtac equalityI 1);
```

```
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (clarify_tac (claset()) 1);
by (rename_tac "g" 1);
by (dres_inst_tac [("x","g")] bspec 1);
by (atac 1);
by (simp_tac (simpset() addsimps [common_objects_def]) 1);
by (rtac conjI 1);
by (atac 1);
by (rtac conjI 1);
by (clarify_tac (claset()) 1);
by (dres_inst_tac [("x","m")] bspec 1);
by (atac 1);
by (subgoal_tac "((gamma g, mu m) : V .<r>)" 1);
by (asm_simp_tac (simpset()) 1);
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (REPEAT (etac conjE 1));
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "ALL S. S <= V .<A> --> Ex (islub S V)" 1);
by (thin_tac "ALL S. S <= V .<A> --> Ex (isglb S V)" 1);
by (dres_inst_tac [("x","(gamma g)")] spec 1);
by (dres_inst_tac [("x","x")] spec 1);
by (etac impE 1);
by (atac 1);
by (dres_inst_tac [("x","(mu m)")] spec 1);
by (etac impE 1);
by (atac 1);
by (atac 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
(* proof of : common_objects {m. m : K.<AS> & (x, mu m) : V .<r>} K*)
(*         <= {g. g : K.<OS> & (gamma g, x) : V .<r>}              *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac ballI 1);
```

```
by (rename_tac "g" 1);
by (rtac conjI 1);
by (asm_full_simp_tac (simpset() addsimps [common_objects_def]) 1);
by (full_simp_tac (simpset() addsimps [infimum_dense_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","x")] bspec 1);
by (atac 1);
by (etac exE 1);
by (subgoal_tac " ! y : A . ? m : K.<AS> . (y = (mu m))" 1);
by (REPEAT (etac conjE 1));
by (full_simp_tac (simpset() addsimps [isglb_def]) 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 11 1);
by (dres_inst_tac [("x","(gamma g)")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
[supremum_dense_def,common_objects_def])) 1);
by (etac impE 1);
by (thin_tac "y : V .<A>" 1);
by (rtac ballI 1);
by (dres_inst_tac [("x","ya")] bspec 1);
by (atac 1);
by (etac bexE 1);
by (dres_inst_tac [("x","g")] bspec 1);
by (asm_full_simp_tac (simpset() addsimps [common_objects_def]) 1);
by (rotate_tac 9 1);
by (dres_inst_tac [("x","m")] bspec 1);
by (atac 1);
by (full_simp_tac (simpset() addsimps [common_objects_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","(mu m)")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (dres_inst_tac [("x","m")] spec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
```

```
by (atac 1);
by (rtac ballI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
(*proof of:common_attributes {g. g : K.<OS> & (gamma g, x) : V .<r>} K*)
(*        = {m. m : K.<AS> & (x, mu m) : V .<r>}                      *)
by (rtac equalityI 1);
(*proof of:common_attributes {g. g : K.<OS> & (gamma g, x) : V .<r>} K*)
(*         <= {m. m : K.<AS> & (x, mu m) : V .<r>}                    *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac ballI 1);
by (rename_tac "m" 1);
by (rtac conjI 1);
by (asm_full_simp_tac (simpset() addsimps [common_attributes_def]) 1);
by (full_simp_tac (simpset() addsimps [supremum_dense_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","x")] bspec 1);
by (atac 1);
by (etac exE 1);
by (subgoal_tac "! y : A . ? g : K.<OS> . (y = (gamma g))" 1);
by (REPEAT (etac conjE 1));
by (full_simp_tac (simpset() addsimps [islub_def]) 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 11 1);
by (dres_inst_tac [("x","(mu m)")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
[infimum_dense_def,common_attributes_def])) 1);
by (etac impE 1);
by (thin_tac "y : V .<A>" 1);
by (rtac ballI 1);
by (dres_inst_tac [("x","ya")] bspec 1);
by (atac 1);
by (etac bexE 1);
by (dres_inst_tac [("x","g")] bspec 1);
```

```
by (atac 1);
by (rotate_tac 9 1);
by (dres_inst_tac [("x","m")] bspec 1);
by (asm_full_simp_tac (simpset() addsimps [common_attributes_def]) 1);
by (full_simp_tac (simpset() addsimps [common_attributes_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","(gamma g)")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (dres_inst_tac [("x","g")] spec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (atac 1);
by (rtac ballI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
(*proof of:{m. m : K.<AS> & (x, mu m) : V .<r>}                        *)
(*    <= common_attributes {g. g : K.<OS> & (gamma g, x) : V .<r>} K *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (clarify_tac (claset()) 1);
by (rename_tac "m" 1);
by (simp_tac (simpset() addsimps [common_attributes_def]) 1);
by (rtac conjI 1);
by (atac 1);
by (rtac conjI 1);
by (rtac allI 1);
by (rtac impI 1);
by (dres_inst_tac [("x","g")] bspec 1);
by (REPEAT (etac conjE 1));
by (atac 1);
by (dres_inst_tac [("x","m")] bspec 1);
by (atac 1);
by (REPEAT (etac conjE 1));
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (REPEAT (etac conjE 1));
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
```

```
 by (thin_tac "ALL S. S <= V .<A> --> Ex (islub S V)" 1);
 by (thin_tac "ALL S. S <= V .<A> --> Ex (isglb S V)" 1);
 by (REPEAT (etac conjE 1));
 by (dres_inst_tac [("x","(gamma g)")] spec 1);
 by (dres_inst_tac [("x","x")] spec 1);
 by (etac impE 1);
 by (atac 1);
 by (dres_inst_tac [("x","(mu m)")] spec 1);
 by (etac impE 1);
 by (atac 1);
 by (asm_simp_tac (simpset()) 1);
 by (auto_tac (claset(), simpset()));
qed "psi_fc";

Open_locale "formal_context";
Goal "[| V : CompleteLattice |] ==> ? phi psi . ? gamma mu .
 (supremum_dense (gamma ` (K.<OS>)) V) & (infimum_dense (mu ` (K.<AS>)) V) &
 (! g : K.<OS> . ! m : K.<AS> . ((g,m) : K.<IR>) =
 (((gamma g),(mu m)) : V.<r>)) -->
 (order_preserving phi (ConceptLattice K) V) &
 (order_preserving psi V (ConceptLattice K)) &
 (my_inv phi psi ((ConceptLattice K).<A>) (V.<A>))";
 by (simp_tac (simpset() addsimps [imp_ex]) 1);
 by (rtac impI 1);
 (* phi is order-preserving *)
 by (dres_inst_tac [("x","gamma")] spec 1);
 by (REPEAT (etac conjE 1));
 by (dres_inst_tac [("x","mu")] spec 1);
 by (REPEAT (etac conjE 1));
 by (res_inst_tac [("x","(% c . (lub (gamma ` (c.<E>)) V))")] exI 1);
 by (rtac conjI 1);
 by (simp_tac (simpset() addsimps [order_preserving_def]) 1);
 by (rtac ballI 1);
```

```
by (rtac ballI 1);
by (rtac impI 1);
by (rtac sup_lt_ss 1);
by (atac 1);
by (subgoal_tac "x.<E> <= K.<OS>" 1);
by (thin_tac "ALL g:K.<OS>. ALL m:K.<AS>. ((g, m) : K.<IR>) =
 ((gamma g, mu m) : V .<r>)" 1);
by (full_simp_tac (simpset() addsimps [supremum_dense_def]) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (atac 1);
by (subgoal_tac "y.<E> <= K.<OS>" 1);
by (thin_tac "ALL g:K.<OS>. ALL m:K.<AS>. ((g, m) : K.<IR>) =
 ((gamma g, mu m) : V .<r>)" 1);
by (full_simp_tac (simpset() addsimps [supremum_dense_def]) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (atac 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def,FormalConcept_def])) 1);
(*  psi is order-preserving *)
by (res_inst_tac [("x","(% x . (| extent = { g . g : K.<OS> &
 ((gamma g),x) : V.<r> } , intent = { m . m : K.<AS> &
 (x,(mu m)) : V.<r> } |))")] exI 1);
by (rtac conjI 1);
by (simp_tac (simpset() addsimps [order_preserving_def]) 1);
by (rtac ballI 1);
by (rtac ballI 1);
by (rtac impI 1);
```

```
by (simp_tac (simpset() addsimps [ConceptLattice_def]) 1);
by (rtac conjI 1);
by (asm_simp_tac (simpset() addsimps [psi_fc]) 1);
by (rtac conjI 1);
by (asm_simp_tac (simpset() addsimps [psi_fc]) 1);
by (rtac conjI 1);
(* proof of : {g. g : K.<OS> & (gamma g, x) : V .<r>}            *)
(*            <= {g. g : K.<OS> & (gamma g, y) : V .<r>}         *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac allI 1);
by (rtac impI 1);
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","(gamma xa)")] spec 1);
by (dres_inst_tac [("x","x")] spec 1);
by (etac impE 1);
by (atac 1);
by (dres_inst_tac [("x","y")] spec 1);
by (asm_simp_tac (simpset()) 1);
(* proof of : {m. m : K.<AS> & (y, mu m) : V .<r>}               *)
(*            <= {m. m : K.<AS> & (x, mu m) : V .<r>}            *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac allI 1);
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","x")] spec 1);
by (dres_inst_tac [("x","y")] spec 1);
by (etac impE 1);
by (atac 1);
by (dres_inst_tac [("x","(mu xa)")] spec 1);
by (asm_simp_tac (simpset()) 1);
```

```
(* phi and psi are inverse functions *)
by (simp_tac (simpset() addsimps [my_inv_def]) 1);
by (rtac conjI 1);
(* psi is right inverse of phi *)
by (rtac ballI 1);
by (rtac (export lubIa RS sym) 1);
by (atac 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [supremum_dense_def])) 1);
by (simp_tac (simpset() addsimps [islub_def]) 1);
by (rtac conjI 1);
by (atac 1);
by (rtac ballI 1);
by (rtac impI 1);
by (full_simp_tac (simpset() addsimps [supremum_dense_def]) 1);
by (etac conjE 1);
by (dres_inst_tac [("x","x")] bspec 1);
by (atac 1);
by (etac exE 1);
by (etac conjE 1);
by (full_simp_tac (simpset() addsimps [islub_def]) 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 10 1);
by (dres_inst_tac [("x","z")] bspec 1);
by (atac 1);
by (etac impE 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (atac 1);
(* psi is left inverse of phi *)
by (rtac ballI 1);
by (subgoal_tac "x.<E> = {g. g : K.<OS> & (gamma g,
 lub (gamma ` (x.<E>)) V) : V .<r>}" 1);
by (subgoal_tac "x.<I> = {m. m : K.<AS> &
```

```
 (lub (gamma ` (x.<E>)) V, mu m) : V .<r>}" 1);
by (asm_simp_tac (simpset()) 1);
(* proof of:                                                              *)
(* x.<I> = {m. m : K.<AS> & (lub (gamma ` (x.<E>)) V, mu m) : V .<r>} *)
by (rtac equalityI 1);
(* x.<I> <= {m. m : K.<AS> & (lub (gamma ` (x.<E>)) V, mu m) : V .<r>} *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac ballI 1);
by (rtac conjI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def,FormalConcept_def])) 1);
by (rtac (export lubE2) 1);
by (atac 1);
by (full_simp_tac (simpset() addsimps
 [supremum_dense_def,ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "ALL g:K.<OS>.ALL m:K.<AS>. ((g, m) : K.<IR>) =
 ((gamma g, mu m) : V .<r>)" 1);
by (thin_tac "x.<E> = common_objects (x.<I>) K" 1);
by (thin_tac "x.<I> <= K.<AS>" 1);
by (thin_tac "common_attributes (x.<E>) K = x.<I>" 1);
by (thin_tac "ALL v:V .<A>. EX A. A <= gamma ` (K.<OS>) & islub A V v" 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (full_simp_tac (simpset() addsimps
 [infimum_dense_def,ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "common_attributes (x.<E>) K = x.<I>" 1);
by (thin_tac "x.<E> = common_objects (x.<I>) K" 1);
by (thin_tac "x.<E> <= K.<OS>" 1);
by (thin_tac "ALL v:V .<A>. EX A. A <= mu ` (K.<AS>) & isglb A V v" 1);
by (thin_tac "x.<E> = {g. g : K.<OS> & (gamma g,
 lub (gamma ` (x.<E>)) V) : V .<r>}" 1);
by (thin_tac "ALL g:K.<OS>.  ALL m:K.<AS>. ((g, m) : K.<IR>) =
```

```
 ((gamma g, mu m) : V .<r>)" 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (full_simp_tac (simpset() addsimps [supremum_dense_def]) 1);
by (rtac ballI 1);
by (dres_inst_tac [("x","xb")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (dres_inst_tac [("x","xa")] bspec 1);
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def,FormalConcept_def,common_objects_def,
 common_attributes_def])) 1);
(* {m. m : K.<AS> & (lub (gamma ` (x.<E>)) V, mu m) : V .<r>} <= x.<I> *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac allI 1);
by (rtac impI 1);
by (etac conjE 1);
by (subgoal_tac "x.<E> = {} | x.<E> ~= {}" 1);
by (etac disjE 1);
by (thin_tac "x.<E> = {g. g : K.<OS> &
 (gamma g, lub (gamma ` (x.<E>)) V) : V .<r>}" 1);
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "x.<E> = common_objects (x.<I>) K" 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (SELECT_GOAL (fold_goals_tac [thm "univ_ax4"]) 1);
by (SELECT_GOAL (rewrite_goals_tac [thm "univ_ax3"]) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (subgoal_tac "? g . g : x.<E>" 1);
by (etac exE 1);
by (full_simp_tac (simpset() addsimps
```

```
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 12 1);
by (etac equalityE 1);
by (full_simp_tac (simpset() addsimps [subset_def,common_attributes_def]) 1);
by (dres_inst_tac [("x","xa")] spec 1);
by (etac impE 1);
by (rtac conjI 1);
by (atac 1);
by (rtac conjI 1);
by (rtac ballI 1);
by (dres_inst_tac [("x","ga")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (rotate_tac 12 1);
by (dres_inst_tac [("x","xa")] bspec 1);
by (atac 1);
by (subgoal_tac "! g : x.<E> . ((gamma g),
 (lub (gamma ` (x.<E>)) V)) : V.<r>" 1);
by (rotate_tac 14 1);
by (dres_inst_tac [("x","ga")] bspec 1);
by (atac 1);
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","(gamma ga)")] spec 1);
by (dres_inst_tac [("x","(lub (gamma ` (x.<E>)) V)")] spec 1);
by (etac impE 1);
by (atac 1);
by (dres_inst_tac [("x","(mu xa)")] spec 1);
by (etac impE 1);
by (atac 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
```

```
by (atac 1);
by (atac 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (simp_tac (simpset()) 1);
(* proof of:                                                              *)
(* x.<E> = {g. g : K.<OS> & (gamma g, lub (gamma ‘ x.<E>) V) : V .<r>} *)
(* x.<E> <= {g. g : K.<OS> & (gamma g, lub (gamma ‘ x.<E>) V) : V .<r>} *)
by (rtac equalityI 1);
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac ballI 1);
by (rtac conjI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
 [ConceptLattice_def,FormalConcept_def])) 1);
by (res_inst_tac [("x","(gamma xa)"),("A","(gamma ‘ (x.<E>))")] bspec 1);
by (rtac (export lubE1) 1);
by (atac 1);
by (full_simp_tac (simpset() addsimps
 [supremum_dense_def,ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "x.<E> = common_objects (x.<I>) K" 1);
by (thin_tac "x.<I> <= K.<AS>" 1);
by (thin_tac "common_attributes (x.<E>) K = x.<I>" 1);
by (thin_tac " ALL v:V .<A>. EX A. A <= gamma ‘ (K.<OS>) & islub A V v" 1);
by (thin_tac "ALL g:K.<OS>.  ALL m:K.<AS>. ((g, m) : K.<IR>) =
 ((gamma g, mu m) : V .<r>)" 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (asm_simp_tac (simpset()) 1);
(* {g. g : K.<OS> & (gamma g, lub (gamma ‘ x.<E>) V) : V .<r>} <= x.<E> *)
by (simp_tac (simpset() addsimps [subset_def]) 1);
by (rtac allI 1);
by (rtac impI 1);
by (etac conjE 1);
by (case_tac "x.<I> = {}" 1);
```

```
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (thin_tac "common_attributes (x.<E>) K = x.<I>" 1);
by (asm_full_simp_tac (simpset()) 1);
by (SELECT_GOAL (fold_goals_tac [thm "univ_ax2"]) 1);
by (asm_simp_tac (simpset()) 1);
by (subgoal_tac "? m . m : x.<I>" 1);
by (etac exE 1);
by (full_simp_tac (simpset() addsimps
 [ConceptLattice_def,FormalConcept_def]) 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 11 1);
by (etac equalityE 1);
by (full_simp_tac (simpset() addsimps
 [subset_def,common_attributes_def]) 1);
by (dres_inst_tac [("x","m")] spec 1);
by (etac impE 1);
by (rtac conjI 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (rtac conjI 1);
by (rtac ballI 1);
by (rotate_tac 11 1);
by (dres_inst_tac [("x","m")] bspec 1);
by (atac 1);
by (REPEAT (etac conjE 1));
by (rotate_tac 13 1);
by (dres_inst_tac [("x","g")] bspec 1);
by (atac 1);
by (atac 1);
by (atac 1);
by (etac equalityE 1);
by (full_simp_tac (simpset() addsimps [common_objects_def,subset_def]) 1);
```

```
by (dres_inst_tac [("x","xa")] spec 1);
by (rotate_tac 13 1);
by (etac impE 1);
by (rtac conjI 1);
by (atac 1);
by (rtac conjI 1);
by (rtac ballI 1);
by (subgoal_tac "! m : x.<I> .
 ((lub (gamma ` (x.<E>)) V),(mu m)) : V.<r>" 1);
by (dres_inst_tac [("x","xa")] bspec 1);
by (atac 1);
by (rotate_tac 14 1);
by (dres_inst_tac [("x","ma")] bspec 1);
by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
by (rotate_tac 13 1);
by (dres_inst_tac [("x","ma")] bspec 1);
by (atac 1);
by (full_simp_tac (simpset() addsimps [CompleteLattice_def]) 1);
by (full_simp_tac (simpset() addsimps [PartialOrder_def,trans_def]) 1);
by (REPEAT (etac conjE 1));
by (dres_inst_tac [("x","(gamma xa)")] spec 1);
by (dres_inst_tac [("x","(lub (gamma ` (x.<E>)) V)")] spec 1);
by (rotate_tac 18 1);
by (etac impE 1);
by (atac 1);
by (dres_inst_tac [("x","(mu ma)")] spec 1);
by (rotate_tac 18 1);
by (etac impE 1);
by (atac 1);
by (etac iffE 1);
by (rotate_tac 19 1);
by (etac impE 1);
by (atac 1);
```

```
 by (atac 1);
 by (atac 2);
 by (atac 2);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 2);
 by (rtac ballI 1);
 by (rtac (export lubE2) 1);
 by (atac 1);
 by (thin_tac "ALL g:K.<OS>. ALL m:K.<AS>. ((g, m) : K.<IR>) =
  ((gamma g, mu m) : V .<r>)" 1);
 by (thin_tac "ALL xa:x.<E>.xa : K.<OS> &(ALL m:x.<I>. (xa, m) :
  K.<IR>) & (ALL x:x.<I>. x : K.<AS>)" 1);
 by (thin_tac "ALL xa:x.<I>.xa : K.<AS> &(ALL g:x.<E>. (g, xa) :
  K.<IR>) & (ALL x:x.<E>. x : K.<OS>)" 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
  [supremum_dense_def])) 1);
 by (thin_tac "ALL xa:x.<I>. xa : K.<AS> & (ALL g:x.<E>. (g, xa) :
  K.<IR>) & (ALL x:x.<E>. x : K.<OS>)" 1);
 by (thin_tac "ALL g:K.<OS>. ALL m:K.<AS>. ((g, m) : K.<IR>) =
  ((gamma g, mu m) : V .<r>)" 1);
 by (thin_tac "ALL xa:x.<E>. xa : K.<OS> &(ALL m:x.<I>. (xa, m) :
  K.<IR>) & (ALL x:x.<I>. x : K.<AS>)" 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset() addsimps
  [infimum_dense_def])) 1);
 by (clarify_tac (claset()) 1);
 by (dres_inst_tac [("x","xc")] bspec 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (rotate_tac 14 1);
 by (dres_inst_tac [("x","mb")] bspec 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
 by (SELECT_GOAL (auto_tac (claset(), simpset())) 1);
qed "basic_theorem_bwd";
```