FORMAL CONCEPT ANALYSIS AND FORMAL METHODS

Thomas Tilley
t.tilley@mailbox.gu.edu.au

GRIFFITH UNIVERSITY
GOLD COAST CAMPUS

FACULTY OF ENGINEERING AND INFORMATION AND TECHNOLOGY

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR PHD CANDIDATURE.

December, 2000

Abstract

Formal Concept Analysis (FCA) is a data analysis technique based on ordered lattice theory. It provides graph-based visualisations of tabular data and has successfully been applied to a number of fields including Text Data Mining, Psychology, Social Science and Software Engineering. This research proposal sets out a framework for the application of FCA to Formal Methods.

Formal Methods can be broadly defined as tools and notations that support the unambiguous specification of computer systems and software. While there can be significant advantages obtained by integrating Formal Methods into the production of software artifacts there is an associated cost. The complexity of Formal Methods means they can be difficult to use and have a non-trivial learning curve. The aim of the research described in this proposal is to increase the accessibility of Formal Methods by providing alternative visual representations of specifications through the application of Formal Concept Analysis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Formal Concept Analysis (FCA) is a data analysis technique based on ordered lattice theory. It provides graph-based visualisations of tabular data and has successfully been applied to a number of fields including Text Data Mining [3, 4, 5], Psychology [6, 7], Social Science [8] and Software Engineering [9, 10, 11, 12]. Although the interpretation of FCA relies on the established meaning of the word "concept" the term "formal" is included because it does not attempt to explain conceptual thinking.

Formal Methods can be broadly defined as tools and notations that support the unambiguous specification of computer systems and software. They provide a means by which the completeness and consistency of a specification be explored, and can be integrated at all stages of the software development life-cycle. Formal Methods can be of benefit to specifiers, implementers and testers by providing verification, validation, and in some cases mechanized code generation. The "formal" in Formal Methods denotes the ordered and deliberate application of mathematically rigorous processes to the act of specification.

1.1 Aim

While there can be significant advantages obtained by integrating Formal Methods into the production of software artifacts there is an associated cost. The complexity of Formal Methods means they can be difficult to use and have a non-trivial learning curve. This research proposal sets out a framework for the application of FCA to Formal Methods. The aim is to increase the accessibility of formal methods by providing alternative visual representations of specifications that may be more intuitive and require less training to use than the formal methods themselves. These alternative views are not intended to replace the original formal specifications but rather there should be a mapping from one to the other so they can be used in conjunction with the original specification.

1.2 Scope

The scope of this proposal requires an exploration of not only FCA and Formal Methods but also Conceptual Graphs [13]. Conceptual Graphs are a knowledge representation technique based on lexical, hierarchically structured ontologies of semantically related terms. Rather than attempting to map Formal specifications directly into FCA the specifications will be transformed into Conceptual Graphs. Existing Conceptual Graph inference engines could then be used to provide leverage in the analysis of specifications and some exploration of the mapping between Conceptual Graphs and FCA has already been done [3, 14]. This work can be exploited to transform the intermediate representation into the desired FCA representation. An abstract view of the process is presented in Figure 1.1.

There are a large number of formal method notations, however, this proposal will focus an a specification language called "Z" [15] and some related derivatives [1] While there may be potential applications to other parts of the software life-cycle such as test-case generation this work will concern itself initially with the visualisation of specifications.

1.3 Overview

This initial Chapter has provided a broad overview of the research proposal and its main aim. Chapters 2 and 3 provide introductions to FCA and Conceptual Graphs

[1] Conceptual Graphs have been shown to represent first order logic and are therefore closer to Z than FCA which is a propositional framework.

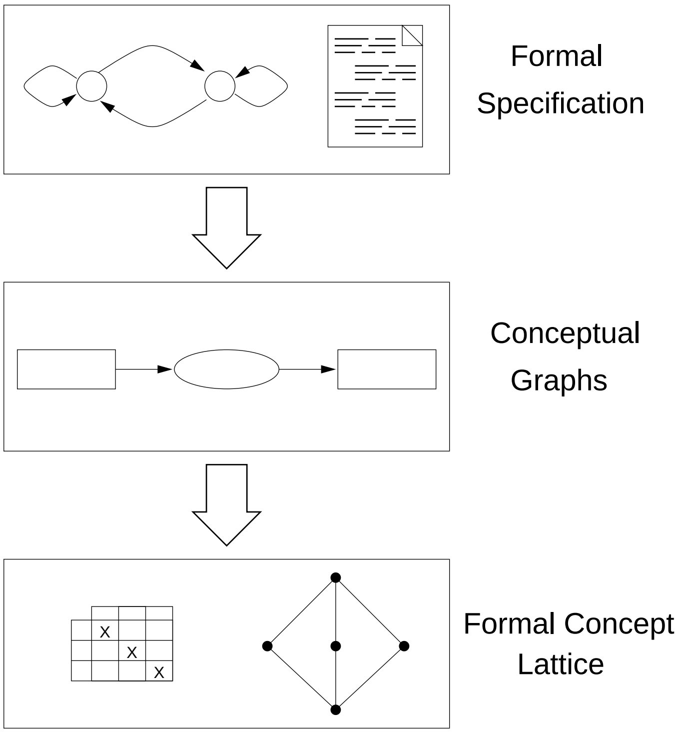

Figure 1.1: Abstract representation of the translation from a Formal Specification into a Formal Concept Lattice via Conceptual Graphs.

respectively while Formal Methods are described in Chapter 4. A framework and schedule for the integration of FCA and Formal Methods via Conceptual Graphs is then presented in Chapter 5.

Chapter 2

Formal Concept Analysis

Formal Concept Analysis (FCA) is a data analysis technique that was introduced by Wille as a way to restructure order theory [16]. It provides an alternative graphical representation of tabular data that is relatively intuitive to use and navigate. This chapter provides a brief introduction to FCA based on Ganter and Wille's book describing the mathematical foundations of FCA [17]. Some parts of the text are also drawn from [18] and [19].

2.1 Formal Contexts

FCA is the process of describing the world in terms of a number of objects and a number of attributes which may be possessed by those objects. The fundamental construct of FCA is the formal context. A *formal context* is a triple (G, M, I) where G is the set of objects, M is the set of attributes and I is a relation defined between G and M. A relation is understood to be a subset of the cross product between the sets it relates. So $I \subseteq G \times M$. If an object object g has attribute m then $g \in G$ is related by I to m and we write $(g, m) \in I$ or gIm. The notion of a *concept* is then introduced as a pair (A, B) where $A \subseteq G$ is a subset of the objects, $B \subseteq M$ is a subset of attributes, and both A and B are maximal with respect to I. In order to define what maximal means we need to introduce the derivation operator, denoted A' when applied to the set A. The derivation operator may be applied to either a set of objects, or a set of attributes. Its two definitions are:

$$A' = \{m \in M \mid \forall g \in A : gIm\} \quad (2.1)$$

$$B' = \{g \in G \mid \forall m \in B : gIm\} \quad (2.2)$$

The derivation of a set of objects, denoted A', contains all attributes that are possessed by all the objects in A. Similarly, the derivation of a set of attributes, denoted B', is simply all the objects that have all that attributes in B. Now for a pair (A, B) to be a concept of the given formal context it must be the case that $A = B'$ and $B = A'$. So a pair (A, B) is a concept of the formal context (G, M, I) if and only if $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. The set A is called the *extent* of the concept and the set B is called the *intent* of the concept. The set of concepts of a formal context forms a lattice which is called the *concept lattice* and for any complete lattice there is an isomorphic concept lattice.

	Lives in Water	Has Hair	Has Gills
Frog	×		
Fish	×		×
Dog		×	

Table 2.1: A representation of a formal context as a cross table. An '×' indicates that an object has that attribute.

Table 2.1 is a representation of a formal context called a cross-table. The context has three attributes (lives in water, has hair, and has gills) and three objects (frog, fish, and dog). An '×' at a particular location or cell in the cross table indicates that an object has an attribute, so for example the object "frog" has the attribute "lives in water". Conversely a blank location indicates that the object does not have that attribute.

2.2 A Lattice of Concepts

Figure 2.1 shows the concept lattice formed from the concepts of the formal context in Table 2.1. The concept lattice contains five concepts the most general of which is marked *Everything*. This concept is the pair ($\{$frog,fish,dog$\}, \{\}$). The bottom concept is marked *Nothing* because it is the concept of objects that have all attributes. In many applications there are objects that possess all the attributes in the formal context and so the labelling of *Nothing* cannot be generally applied to the bottom concept. The remaining concepts are marked with their intent and their extent. The diagram shows that there is an ordering over the concepts. This ordering is a specialisation ordering denoted by $\leq$. The concept $(A_1, B_1) \leq (A_2, B_2)$ if $A_1 \subseteq A_2$ or equivalently $B_2 \subseteq B_1$.

Concept lattices can be labelled in an economical way. For each attribute m there is a maximal (with respect to the specialisation ordering) concept that has that attribute in its intent. All more specific concepts will also have that attribute in their intents so if one attaches attributes to their maximal concepts in the diagram then the intent of a concept may be determined by collecting attribute labels from the more general concepts. If one applies this idea to Figure 2.1 then the concept for frog would have the label "lives in water" and the concept for fish would have the label "has gills". To determine the intent of the concept for fish one would collect "has gills" and "lives in water" because there is an upwards path from the concept for fish to the concept for frog. Similarly, for objects one may attach object labels to their minimal concepts (also known as their most specific concepts) and determine the extent of a concept by collecting object labels attached to concepts on downward paths from the concept in question.

2.3 Multi-valued Contexts

FCA may be applied to data in which objects are interpreted as having attributes with values. A common example may be an analysis of cars in which the attributes

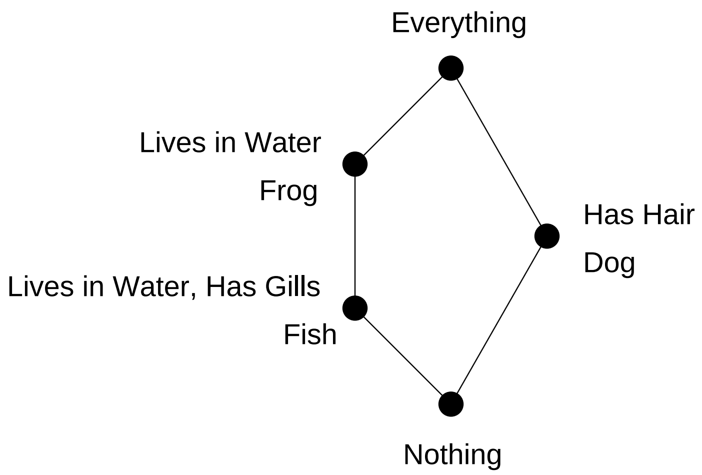

Figure 2.1: Concept lattice showing the five concepts of the formal context in Table 2.1.

are colors, maximum speed, etc. In this case the basic data is stored in a multi-valued context. A *multi-valued context* is a tuple (G, M, W, I) where G is a set of objects, M is a set of attributes, W is a set of attribute values and $I \subseteq G \times M \times W$ is a relation saying which object has which attribute value. The relation I is restricted such that for any $(g, m, w_1) \in I$, and $(g, m, w_2) \in I$, it is the case that $w_1 = w_2$. In other words every object has only one value for a particular attribute. The set of all values taken on by objects for a given attribute m, is denoted W_m. In other words $W_m = \{w \in W \mid \exists g \in G : (w, m, g) \in I\}$. An object, g has at most one attribute value for each attribute, m. A partial function to produce this attribute value is introduced, $m(g) := w$ such that $(g, m, w) \in I$.

	Color	Max. Speed
Car 1	Red	100
Car 2	Green	120
Car 3	Black	160

	Light Color	Dark Color
Red	×	
Green	×	
Black		×

Table 2.2: Multi-valued context and a conceptual scale for the color attribute

For the purpose of analysis, a multi-valued context is transformed into a single valued formal context via a conceptual scale. The new single valued formal context is called the derived context, and presents a summary of the data. A conceptual scale is normally applied to a single attribute m in which case the scale is itself a formal context, $S_m = (G_m, M_m, I_m)$ where $W_m \subseteq G_m$, M_m is a new set of attributes, and I_m is a relation connecting attribute values in G_m with the new attributes in M_m. To make this situation clearer consider the example in Table 2.2. On the left a multi-valued context has been drawn and on the right is a scale for the color attribute. The scale translates Red and Green to be a Light Color and Black to be Dark Color. Table 2.3 shows the result of applying this scale to the multi-valued context in Table 2.2. The objects, Car 1 and Car 2 have the new attribute Light Color and the object Car 3 has the new attribute Dark Color. The conceptual scale is used to generate a new context called the *context derived with respect to plain scaling.* This new context is derived from the scale and the multi-valued context, and is defined as: (G, M_m, J) where for $g \in G$ and $n \in M_m$

$$(g, n) \in J :\Leftrightarrow \quad \exists w \in W_m : \ (g, m, w) \in I \quad \text{and} \quad (w, n) \in I_m$$

	Light Color	Dark Color
Car 1	×	
Car 2	×	
Car 3		×

Table 2.3: The derived context resulting from applying the scale in Table 2.2 to the multi-valued context in Table 2.2

Often it is inconvenient to enumerate all attribute values of an attribute in a scale. In such a case we define a scale to be a context (G_m, M_m, I_m) where the scale objects G_m partition the attribute values W_m via a function $\alpha : W_m \rightarrow G_m$ called the *composition operator* and assigning to each attribute value a single scale object.

Then the derived context is given by:

$$(g, n) \in J :\Leftrightarrow \quad \exists w \in W_m : (g, m, w) \in I \quad \text{and} \quad (\alpha(w), n) \in I_m$$

2.4 TOSCANA and Anaconda

The process of conceptual scaling has been successfully captured in the tool set comprised by the computer programs TOSCANA and Anaconda [20, 21] . Anaconda allows the user to construct conceptual scales by entering crosses in a cross table such as the one drawn in Table 2.2. The objects, rather than enumerating all the possible attribute values, are expressed as fragments of an SQL statement. So for example an object in a scale for Maximum Speed might be `Cars.speed` $\geq$ `100`. When using the Anaconda and TOSCANA systems it is important that the SQL queries attached to the objects partition the set of values of the attribute for which the scale is being constructed. The TOSCANA program is used to derive concept lattices from the conceptual scales produced using Anaconda and the multi-valued context stored in a relational database. TOSCANA draws the concept lattice of the conceptual scale and replaces the scale objects with objects retrieved from the database via the SQL query attached to the scale objects.

Replacing the scale objects with objects from the multi-valued context is possible because there is an $\bigvee$-preserving order embedding from the derived concept lattice into the scale concept lattice given by:

$$(A, B) \mapsto (B', B)$$

where B' is derivation of B with respect to scale context. The order embedding means that we can use the diagram of the scale concept lattice to display the derived concept lattice. Any concepts that are not in the image of the embedding are displayed as grey nodes indicating that no concept exists there. The lines passing through the grey nodes are still used for determining the ordering relation between other concepts

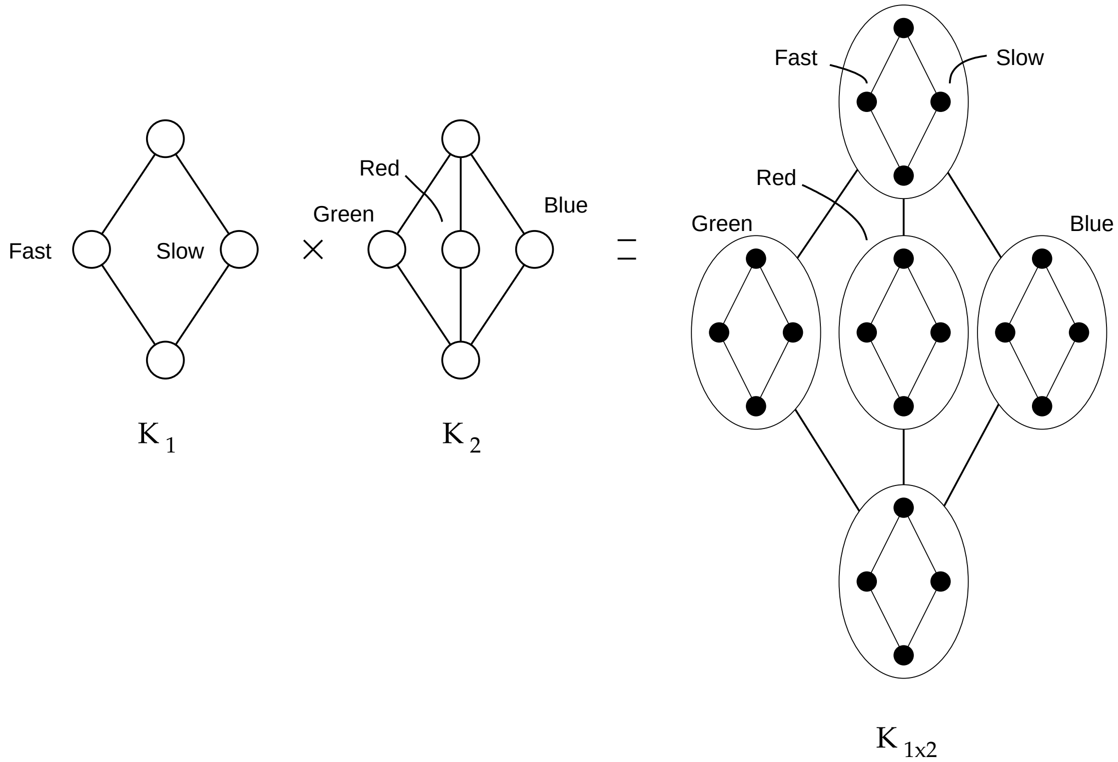

Figure 2.2: Example of a nested line diagram, shown as the product of two scale lattices.

and also help to give a clue to the presence of attribute implications in the concept lattice.

Two scales, acting on the same multi-valued context (G, M, W, I), can be combined and the resulting concept lattice drawn using a technique called nested scaling. An example of nesting two scales is shown in Figure 2.2. The two constituent scales are shown on the left and the lattice product is shown on the right. The product is the direct product of the concept lattices. There is an order embedding from the concept lattice of the semi-product of the scales into the direct product, and an order embedding from the derived concept lattice into the concept lattice of the semi-product. Concepts which are not mapped to in the direct product, from the derived concept lattice, by the composition of the two order embeddings, are normally shown as faded grey circles.

2.5 Applications

As mentioned in Chapter 1, FCA has been successfully applied to a number of domains including Psychology [6, 7] and Social Science [8]. In the field of Software Engineering FCA has been used to re-engineer class hierarchies for the object-oriented programming language C++ [9, 12], for configuration management [11], and for indexing and retrieving software components [10].

The analysis of email collections using FCA also demonstrates applicability to text data mining and information retrieval [4]. Cole and Eklund's work also addresses the issue of scalability for large datasets partly through the application of conceptual scales.

Chapter 3

Conceptual Graphs

This chapter provides a brief introduction to Conceptual Graphs (CGs). An example that illustrates a number of the representational forms of CGs is presented in Section 3.1. Section 3.2 describes some previous work linking CGs with Formal Concept Analysis, while Section 3.3 introduces the concept of CGs as interlingua.

3.1 Background

CGs are a knowledge representation technique based on Pierce's Existential Graphs [13, 1]. They can be represented in a number of notations including a graphical *display form* (DF), and the textual *linear form* (LF), *conceptual graph interchange form* (CGIF), *Knowledge Interchange Format* (KIF) and typed predicate calculus. A statement expressed in any of these forms can be translated into a logically equivalent statement in any of the other forms. Both DF and LF are designed to be human readable as well as facilitating communication between humans and machines. A draft proposed American National Standards Institute (ANSI) standard for conceptual graphs has been developed [1]. Several other variants have also been developed including Formalised English (FE) and Frame Conceptual Graphs (FCGs) [22].

[1]see http://www.bestweb.net/~sowa/cg/cgdpans.htm

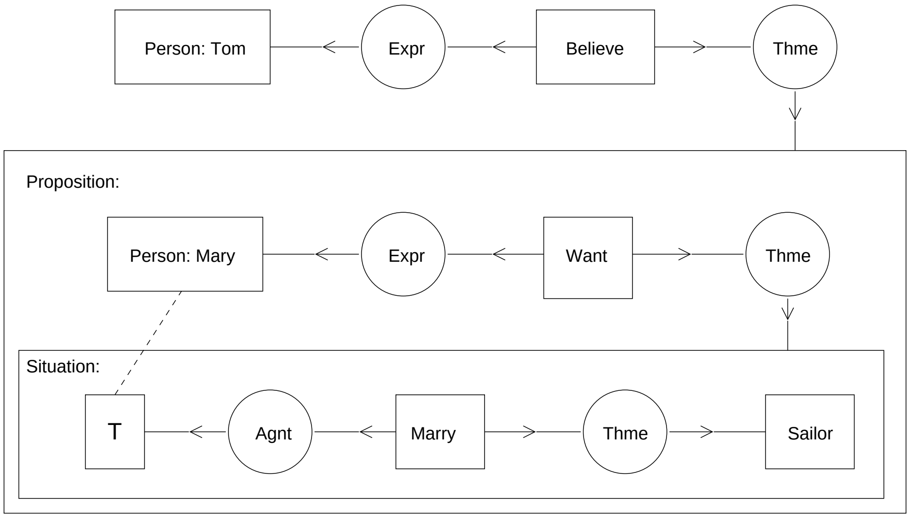

Figure 3.1: CG Display Form of the sentence *Tom believes that Mary wants to marry a sailor* taken from [1].

Figure 3.1 presents the display form of a CG representation of the English sentence "Tom believes that Mary wants to marry a sailor". This example and some of the corresponding text is drawn from [1]. The equivalent linear and CGIF forms are shown in Figures 3.2 and 3.3 respectively.

```
[Person:  Tom]<-(Expr)<-[Believe]->(Thme)-
   [Proposition:  [Person:  Mary *x]<-(Expr)<-[Want]->(Thme)-
      [Situation:  [?x]<-(Agnt)<-[Marry]->(Thme)->[Sailor] ]].
```

Figure 3.2: Linear form of the conceptual graph shown in Figure 3.1.

CGs are bipartite with rectangles representing *concepts* and circles or ovals representing the *relations* between concepts. Relations and concepts are linked by directed arcs and if a relation has more than two arcs then the arcs are numbered. The outer boxes labelled "proposition" and "situation" in Figure 3.1 are special concepts known as *contexts*. That part of the graph outside the contexts is assumed to be true in the real world. In this example Tom is the experiencer of the concept "Believe" which is linked by the theme relation to a proposition believed by Tom. A CG is nested inside the "proposition" context which says Mary is the experiencer of "want" which has as its theme a situation that Mary hopes will come to pass. A *coreference link* shown by the dotted line between the situation and the proposition indicate that Mary is the same individual referred to inside the box. The innermost CG therefore says Mary marries a sailor.

```
[Person:  *x1 'Tom'] [Believe *x2] (Expr ?x2 ?x1)
   (Thme ?x2 [Proposition:
      [Person:  *x3 'Mary'] [Want *x4] (Expr ?x4 ?x3)
         (Thme ?x4 [Situation:
            [Marry *x5] (Agnt ?x5 ?x3) (Thme ?x5 [Sailor]) ]) ])
```

Figure 3.3: Conceptual Graph Interchange Format (CGIF) form of the conceptual graph shown in Figure 3.1.

A complete CG definition also includes; an ontology describing the sub-type/super-type relationships between the concepts and relations in the graph, conformity relations (or signatures) that describe the types of individuals in the graphs and formation rules. Other graph operators can be composed from the basic operations which are *restrict, copy, simplify and join.* Editing and reasoning tool support for CGs is available and several extensions have been proposed including an executable form [23].

3.2 Conceptual Graphs and Power Context Families

Of particular relevance to this proposal is some existing work that links CGs and Formal Concept Analysis [14, 3, 24]. Groh and Eklund have implemented Wille's algorithm for transforming CGs into a set of formal contexts. Their implementation takes a CG as input and creates a set of formal contexts which are stored as tables in a relational database. This set of contexts is called the *Power Context Family.*

Having stored the Power Context Family as a series of tables another CG can then be used to generate a query to retrieve objects from the database.

3.3 Interlingua

The example from Section 3.1 illustrates how CGs can represent natural language sentences. With their direct mapping to language CGs can serve as an intermediate language for natural language processing or natural language generation sytems. They can also be used as an intermediate language or "interlingua" for other applications.

RDF is the W3C's Resource Description Framework for describing meta-data. A mapping from RDF to CGs has been implemented so queries to a knowledge base can be specified in RDF but the retrieval is performed using CGs [25]. The results from these queries are returned as RDF so it appears to users of the system that the reasoning was done in RDF itself. The results, however, were made available by the application of CG tools to this intermediate representation. The CGs in the system were implemented using the NOTIO JAVA API [26].

Within our own research laboratory Martin and Eklund have been developing a knowledge base server named *WebKB-2* that permits Web users to add knowledge in a single knowledge base [27]. The knowledge base forms what the authors called a rudimentary "Domain ontology Server" or DOS. The DOS is analogous to the DNS mechanism used on the Web. A DNS resolves domain names to IP addresses, it is distributed with a well understood propagation mechanism for update. Similarly, the DOS is designed to resolve textual terms to their semantic context and its meta-mechanics to distributed propagation of updates.

The present prototype ontology server, WebKB-2, is initialised with the natural language ontology WordNet and a top-level universal ontology. WebKB-2 can be used for representing the content of documents and therefore indexing them, although the creation and retrieval of knowledge is finer-grained permiting greater precision than keyword-based document retrieval. Eklund and Martin claim that the feature set of WebKB-2 represents the minimum requirement for an ontology-based semantic Web, such as that described by Tim Berners-Lee [2].

Applications of the semantic Web have been slow to realise but are easy to imagine. One example is a subscriber constructed "Yellow-Pages"-like catalogue. Such a catalogue would express deep semantic structure, by interrelating terms: express the term types and contexts with interconnections to related meanings and terms. Consumers of the Yellow-pages query it for goods and services while producers generate knowledge markup that profiles the semantic meaning and context of their product.

One semantic Web application, developed by Bruza, Eklund and Martin, is called *HiBKB* [28, 29]. HiBKB is contract research for defense analysis, and demonstrates the use of the DOS elements in a practical system. The HiBKB is a groupware knowledge management system that performs precision-based knowledge acquisition. Analysis tasks are supported by the creation of an ontology or re-usable terms. These are ordered hierchically and augmented by WordNet. The ontology is used as a structured dictionary of re-usable types and relations.

Documents relevant to the analysis task are identified using information retrieval, in particular a query by refinement engine developed at the Distributed Systems Technology Centre [30] called the *Hyper-Index Browser*. Once the target documents are identified, text within documents is highlighted only if it contains the search terms and one or more ontology terms. The level of inheritance in the ontology, and therefore level of generality of the text high-lighted, can be adjusted by the user. Text (as a sentence or a paragraph) once highlighted, is parsed and output as a structured knowledge representation called a conceptual graph. The conceptual graphs are stored within a knowledge base and can be used to search and navigate the original documents and their context. The main reusable architectural feature of the HiBKB is a scalable ontology server. This allows the centralised control of access, update and the service of concepts, relations and their interconnections.

[2]WebKB-2 can be queried at
http://tempus.int.gu.edu.au/ phmartin/WebKB/webKBshared.html

Chapter 4

Formal Methods

This chapter provides a brief introduction to Formal Methods and overviews of some specific systems. Section 4.2 introduces Estelle, LOTOS and SDL - the Formal Description Techniques that were developed as part of the Open Systems Interconnection initiative. Sections 4.3 and 4.4 provide an overview of the Z specification language and the proof checker HOL which has been applied to Z.

A less rigorous approach to applying formal methods is described in Section 4.5. These "lightweight" approaches include a derivative of Z called "Alloy". Alloy provides a simple mapping into UML, the Universal Modelling Language, and it represents one possible path for the convergence of Formal Methods and UML. This convergence is discussed in Section 4.6. Finally, Section 4.7 discusses some of the limitations and problems associated with the use of Formal Methods.

4.1 Background

Formal Methods can be broadly defined as tools and notations with formal semantics that support the unambiguous specification of the requirements for a computer system. They provide a means by which the completeness and consistency of a specification can be explored as well as proofs of correctness for implementations of the specification [31, 32]. The application of Formal Methods can be of benefit to specifiers, implementers, and testers by providing unambiguous communication, verification, validation, and in some cases mechanized code generation [33].

The National Aeronautic and Space Administration (NASA) have produced two formal methods guidebooks which are used in-house for the specification and verification of software and computer systems [34, 35]. NASA advocate the use of formal methods at all stages of the software life-cycle but in particular during the specification of requirements. Gathering complete requirements and defining exactly what a system is supposed to do is one of the major problems faced by software engineers. The NASA guidebooks cite a number of case studies where defects in the final system implementations can be traced back to incomplete or insufficient specification of the initial system requirements. As with other software engineering processes there is a certain amount of organisational maturity that is required before Formal Methods can be effectively integrated and applied however there are potential cost savings both in terms of more reliable software and in early defect removal or avoidance. The earlier a defect is detected in the software development process, the cheaper it is to fix [36].

The integration of formal methods into a development project is not necessarily an “all or nothing” approach. Existing formal methods do not necessarily scale well when applied to large systems. There is also an associated cost in terms of time and expertise in their application. For this reason formal techniques are often applied only to the “critical” parts of a system in an effort to prove their correctness or improve reliability. They can also be used alongside or in conjunction with existing processes to improve software [32, 37, 38].

One categorisation of Formal Methods is based on whether they are descriptive or analytic in nature. Descriptive formal methods focus on specification as a tool for review and discussion. They generally use notations based on set theory and do not readily support automation [34]. The specification language Z is briefly introduced in Section 4.3 and could be considered aa an example of a descriptive formal methods.

Analytic formal methods focus on mathematical specifications that can be used for analysing and predicting the behaviour of systems. This can include specifications or prototypes which can themselves be executed to prove that a system holds certain properties. The process of running or executing a specification is referred to as "animation". Analytic methods are also more suited to automation and mechanised deduction with varying amounts of user direction being required. The HOL theorem proving system is an example of an analytic method that requires significant user input. HOL is described briefly in Section 4.4.

There are a large number of techniques that could be classified as formal methods depending on their application to the software engineering process. In some sense all programming languages defined in Backus-Naur Form (BNF) are using formal methods. Their syntax is defined using formal notation however their semantics are not usually formally specified [31]. A comprehensive list of Formal Methods notations and tools is available on the World Wide Web (WWW) at the Formal Methods Virtual Library [1] and at the site maintained by Formal Methods Europe [2]. A selection of the methods presented there are summarised in the following sections.

4.2 Formal Description Techniques

Open Systems Interconnection (OSI) is a standardisation effort by the International Standardisation Organisation (ISO) to facilitate the exchange of information between data processing equipment. The OSI required ways to unambiguously define their standards in an implementation independent manner. They needed to describe what to do but not how to do it. In response to the ISO's need a work group was set up to standardise formal specification languages which resulted in three Formal Description Techniques (FDTs): Estelle, LOTOS and SDL [33]. These FDTs were used to specify services and protocols within the fabled OSI reference model [39]. Their shared basis is behaviour specification via labelled transition systems [40]. These systems are tabular or graphical representations of automata where transitions between states occur in response to actions.

[1] see http://www.comlab.ox.ac.uk/archive/formal-methods.html

[2] see http://csr.ncl.ac.uk/projects/FME/InfRes/tools/name.html

4.2.1 Estelle

Estelle is the Extended State Transition Language or Extended Finite Sate-Machine Language that was approved as an ISO international standard in 1989. Estelle is suitable For describing distributed or concurrent processing systems and in particular was used to describe OSI services and protocols. Outside the OSI it has been used to specify military mobile combat network radio protocols [41] and a repository of Estelle protocol specifications is maintained by the University of Delaware [3].

Systems are specified in Estelle as a hierarchy of communicating non-deterministic state machines [40]. This hierarchy represents different levels of abstraction where the lower levels represent refinements of the higher levels. A PASCAL derivative is used to specify actions, and both synchronous and asynchronous parallelism can be modelled between state machines.

4.2.2 LOTOS

The Language of Temporal Ordering Specification (LOTOS) was the first piece of mathematics to be standardised internationally [4]. It is based on the process algebras CCS (the Calculus of Communicating Systems) and CSP (Communicating Sequential Processes). Implementation independent specifications of Abstract Data Types (ADTs) are also incorporated using the ACT ONE language. ACT ONE is used to specify the "allowed" behaviour for a conforming implementation of a LOTOS specification.

LOTOS models both synchronous and asynchronous communication and it has been used to specify both connection-less and connection-oriented services and protocols in six of the seven layers of the OSI reference model. E-LOTOS (Enhancements to Language of Temporal Ordering Specification) makes a number of extensions to increase the modularity of LOTOS and it introduces a temporal semantic [42]. A number of examples are available from the University of Madrid [5].

[3] available by anonymous FTP from ftp://ftp.udel.edu/pub/grope/estelle-specs
[4] ISO/IEC 8807 available at http://www.iso.ch/cate/d16258.html
[5] available by anonymous FTP from ftp://ftp.dit.upm.es/pub/lotos/specifications/

4.2.3 SDL

SDL is the Specification and Description Language standardised by the International Telecommunication Union-Telecommunication (ITU-T). It is based on an extended Finite State Machine [40] model that is supplemented with the ACT ONE ADT specification language used in LOTOS.

SDL provides constructs for representing structures, behaviours, interfaces, and communications links within a specification. It supports abstraction, module encapsulation, and refinement and a number of commercial and shareware tools are available [6].

MSC

Message Sequence Charts (MSC) are a graphical and textual language used to describe interactions between components in a system [43]. MSC is often used in conjunction with SDL to provide an overview of the communication within a system being modelled. They are particularly suited for real-time systems and can be used on their own for specification, simulation, validation and documentation. Their use in combination with SDL is analogous to that of use cases in UML.

4.3 Z

Z is a widely used specification notation based on set theory and first order predicate logic that has been under development since the late 1970s [15, 44]. Writing Z specifications requires a number of non-ASCII mathematical symbols and as a result most Z tools are comprised of at least a formatting package for LaTeX and a type-checker.

Operations in Z are specified by their input/output behaviour and are represented diagrammatically by named *schema* boxes. The schema are divided into an upper region called the *declaration-part* of the schema and a lower region called the *predicate-part* or more correctly the *formula-part*. Models are constructed by specifying a series

[6] see http://www.sdl-forum.org/Tools

of schemas.

Z is a non-executable specification language that aims to increase the understandability of the system being specified, however, some mechanised proof support is available by embedding Z in HOL (see Section 4.4). A number of extensions to Z for modelling object oriented systems have also been proposed.

4.4 HOL

Higher Order Logic (HOL) is an interactive mechanised proof checker that supports both forward and backward proofs [34]. Forward proofs apply inference rules to existing theorems to derive new theorems. Rules are applied until the desired theorem is derived. Backward or goal-oriented proofs start with a desired theorem and then attempt to decompose it into simpler existing theorems.

4.5 Lightweight Formal Methods

Lightweight formal methods are "lightweight" in that they offer "less than completely formal" or "partial" approaches to specification, validation and testing [45, 46]. Typically they trade off completeness or language functionality for efficiency. While they don't offer the same certainty that more formal theorem proving provides they can still be used to refute theories or to detect defects early in the development life-cycle.

4.5.1 Alloy

Alloy is an example of a lightweight formal method [46]. It provides a language that has both graphical and textual representations that are equivalent to each other. While many formal methods have both a graphical and textual component they are generally not equivalent. Typically one representation includes additional properties or annotations that are not easily representable in the other.

The Alloy language is based on Z and is tailored for handling object models. According to Jackson:

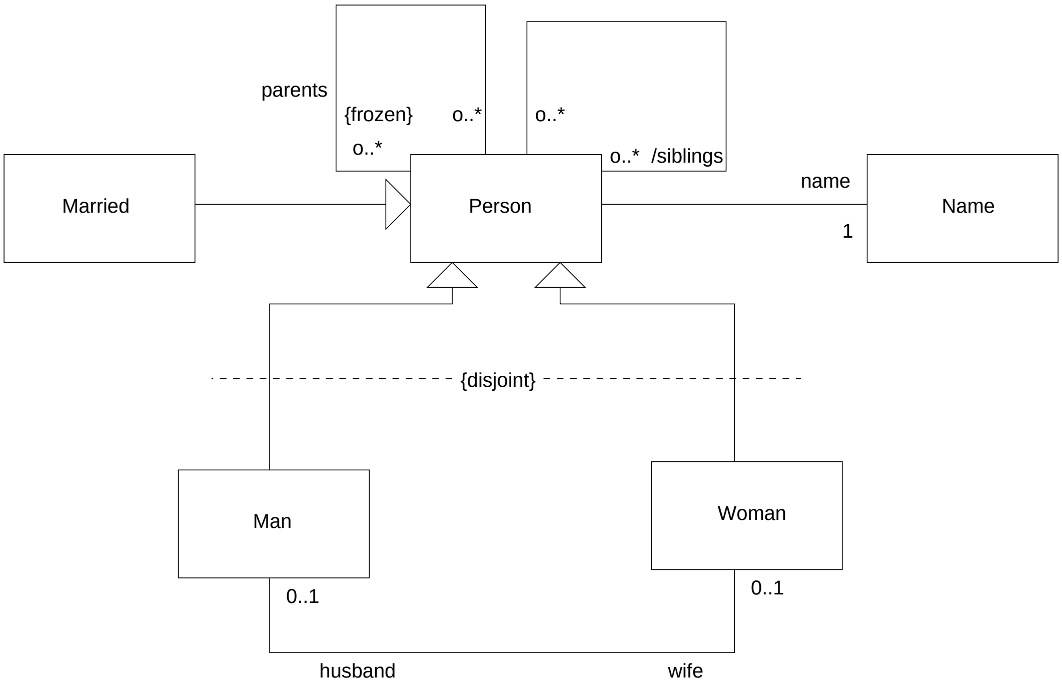

Figure 4.1: Graphical part of a UML model of a family tree taken from [2].

> Alloy is close enough to UML to make transcription of an object model diagram into Alloy a trivial task…

Figure 4.1 shows the graphical part of a UML model for a family tree. The corresponding Alloy diagram is presented in Figure 4.2.

Unlike Z the language can be represented using standard ASCII characters but is still close enough that existing Z and Vienna Development Method (VDM) specifications can be analysed using Alcoa - the Alloy Constraint Analyser. Figure 4.3 presents the textual part of the Alloy family tree model. The *domain* and *state* paragraphs correspond directly to the diagram in Figure 4.2. A number of sample Alloy specifications are available on the WWW [7].

[7]see http://sdg.lcs.mit.edu/alloy/docs/examples.html

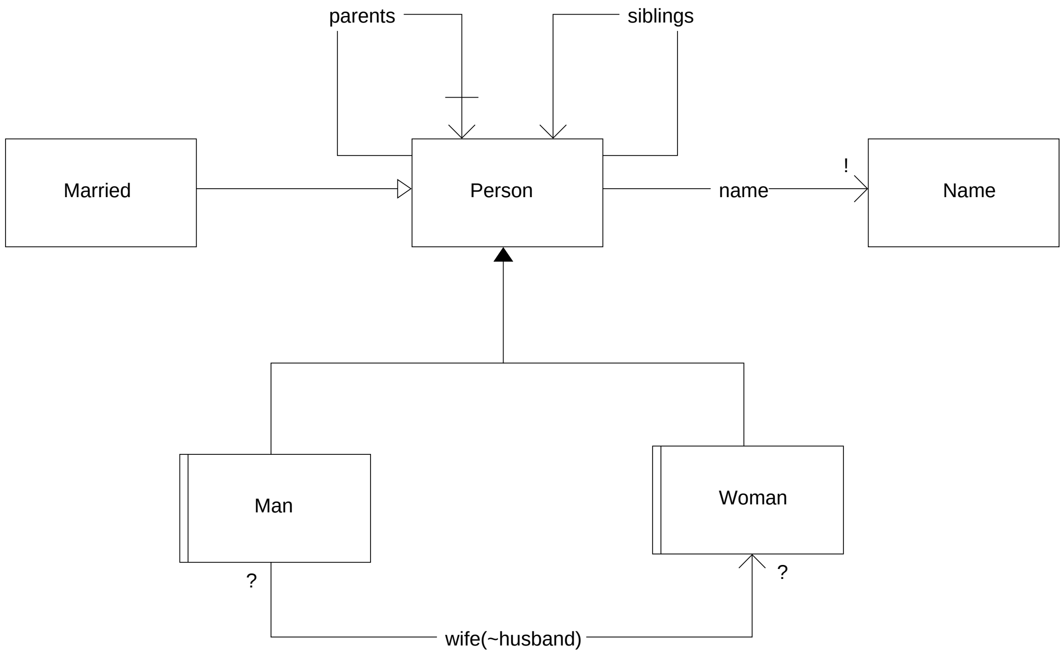

Figure 4.2: Partial Alloy model of the *domain* and *state* paragraphs from Figure 4.3. The example is taken from [2].

4.6 Convergence with UML

UML [47] has become the de facto industry language for modelling systems. While it provides a means of specification it does not have the mathematical rigour of formal methods. On the other hand it enjoys a popularity that formal methods do not (see Section 4.7). UML is also implementation oriented which may be helpful to the ultimate implementers of a particular model, however, it is inconsistent with the aims of a conceptual modelling language [48].

There are a number of possible paths for convergence between UML and formal methods. The first is for UML to adopt those ideas from formal methods that can provide implementation abstraction and verification capabilities. This appears to be the case with UML 2.0 request for information input arising from the SDL and MSC community [8].

[8]see http://www.irisa.fr/manifestations/2000/sam2000/SAM2000

An alternative path is for formal methods to become more UML like. This also appears to be happening with lightweight formal methods like Alloy that offer straightforward mappings from UML into a formal notation [46, 2].

4.7 Limitations

A paper by Hall in 1990 attempted to dispel a number of myths surrounding the use of formal methods [49]. These seven myths were:

1. Formal methods can guarantee perfect software and eliminate the need for testing.
2. Formal methods are all about proving systems correct.
3. Formal methods are only useful in safety-critical systems.
4. Formal methods requires highly trained mathematicians.
5. Formal methods increases development costs.
6. Formal methods are unacceptable to users.
7. Formal methods are not used on large-scale systems.

A number of case studies and applications were presented as counter examples to demonstrate that the myths were untrue however some of them live on.

While formal methods may be useful outside of safety-critical systems (Myth 3) it is here and in the domain of real-time systems where they have found their niche. NASA argue that this is not because they have nowhere else to turn to increase the reliability of their systems but rather the very nature of the safety-critical and real-time domains makes them well suited to the adoption and integration of formal methods [34, 50].

Bowen and Hinchey produced a "follow-up" paper in 1995 which introduced seven more myths. The second myth they sought to dispell was that "Formal Methods are not supported by tools" [51]. There is certainly support for tools as evidenced by the formal methods tools databases on the WWW, however, in papers published since 1995 there continues to be a call for new tools. These calls cite a need not only for tools that have matured from research prototypes into robust, commercial quality software [52, 53], but also for functionality that is not currently supported. There is a need for Tools that can present comprehensible specifications and proofs for large systems at different levels of abstraction:

> One important problem in current formal methods is that in practice it is difficult to relate formal views of the same system at different levels of abstraction. If we had better practical solutions to this problem, it might be easier to apply formal methods at many stages during the development of a large system [54].

In support of this Clarke and Wing in their paper on the current state of formal methods and future directions list abstraction as a fundamental concept that requires further work:

> Real systems are difficult to specify and verify without abstractions. We need to identify different kinds of abstractions, perhaps tailored for certain kinds of systems or problem domains, and we need to develop ways to justify them formally, perhaps using mechanical help [55].

With reference to formal methods based on Abstract State Machines a similar request is made for:

> More advanced and industrially satisfactory tool support... for defining, simulating and visualizing... ASMs [56].

Clarke and Wing go on to list a number of criteria that methods and tools should attempt to address including ease of use, efficiency, and focused analysis. They argue that tools and their output should be as easy to use as compilers. The time taken for analysis should be comparable to that of compilation and individual tools need not be good at analysing all aspects of a system, but they should analyse one aspect well.

While future tools may make the integration of formal methods easier there is currently a certain level of organisational maturity that must exist before the benefits afforded by formal methods can be obtained. As with the application of other software engineering processes like the Capability Maturity Model (CMM) [36], formal methods are not suitable for all development environments.

In addition to the above there are two fundamental limitations of formal methods. Firstly, formal methods can prove an implementation satisfies a given formal specification but they can't prove that the specification captured the users actual understanding of a system. A system can only ever be proved correct with respect to its specification [51]. Secondly, while a formal method may show that a specification correctly implements a program, the compiler the program is fed to may not correctly implement the language. If we verify the compiler as being correct then there could still potentially be a problem with the hardware itself. At some point it must be accepted that a physical system satisfies the axioms used in a proof [31].

Chapter 5 presents an approach to integrating Formal Concept Analysis with Formal methods that seeks to address some of these limitations.

```
model Family {
domain {Person,Name}
state {
partition Man,Woman : static Person
    Married : Person
    parents : Person -> static Person
    siblings : Person -> Person
    wife (~husband) : Man ? -> Woman ?
    name : Person -> Name !
    }
def siblings {
    all a,b|a in b.siblings <-> (a.parents = b.parents)
    }
inv Basics {
    all p|some p.Wife <-> p in Man & Married
    no p|p.Wife / in p.siblings
    all p|(sole p.parents & Man) && (sole p.parents & Woman)
    no p|p in p.+parents
    all p,q|p.name = q.name -> no (p.parents & q.parents)
    }
op Marry (m:Man!,w:Woman!) {
    m not in Married && w not in Married
    m.wife' = p.wife
    all p:Man - m|p.wife' = p.wife
    all p|p.name' = p.name
    all p|p.parents' = p.parents
    Person' = Person
    }
assert HusbandsWife {
    all p:Married & Woman|p.husband.wife = p
    }
}
```

Figure 4.3: Textual part of an Alloy model of a family tree taken from [2]. The *domain* and *state* paragraphs correspond to Figure 4.2

Chapter 5

Integrating Formal Concept Analysis and Formal Methods

Having introduced the required background in the previous chapters, this final chapter now describes a proposed technique to integrate FCA with Formal Methods by using CGs as an interlingual representation. Section 5.1 summarises the motivation for the work which is then described in Section 5.2. Finally, Section 5.3 presents a schedule for the conduct and completion of this research.

5.1 Motivation

Formal Methods have a number of benefits to offer software engineers however these have generally only been taken up by the safety-critical and real-time software communities. While there are initial costs involved with integrating a formal method into the software development process there are also potential cost savings throughout the lifecycle and ultimately in more reliable software. The two main obstacles to wider formal methods use appears to be the perceived difficulty in using formal methods and the lack of commercial quality CASE tools with the required functionality (see Section 4.7).

Existing work has already looked at increasing the usability of formal methods by using tabular notations. An example is the SC(R)^3 system which is based on the Naval Research Laboratories SCR (Software Cost Reduction) method [57]. Another approach has been to provide alternate textual representations of specifications with simple mappings or links for navigation between the two [38]. These alternatives are not meant to replace the existing specifications but rather to supplement them with different views of a system. The fifth commandment of formal methods is "Thou shalt not abandon thy traditional development methods" [37]. This commandment applies equally well to not abandoning formal specifications but using alternative representations in conjunction with the specification itself.

One of the early applications of the TOSCANA system was as a navigation tool for laws and regulations in civil engineering [19]. If we consider formal methods to be a kind of software regulation then there is a parallel here. FCA is a data-analysis technique that provides an appropriate visual representation for tabular data [58]. It supports scalability and information hiding through conceptual scaling and lattice folding/unfolding. The line diagram representations are intuitive to use and exploit the power of human visual processing while providing a mechanism for retaining global context. FCA has also already been successfully applied to other software engineering tasks.

The application of FCA to Formal Methods described here is an exploration of the meeting between these two areas. The aim is to provide a framework, not in the software component re-use sense, but rather a theoretical framework that can be demonstrated in software. The aim is not to produce commercial grade CASE tools based on FCA but rather to provide an exploration of the foundations on which such tools may be built in the future.

5.2 Methodology

The proposed integration of FCA and Formal methods via CGs is illustrated on an abstract level by Figure 1.1 that was presented in Chapter 1. The initial aim is to provide alternate representations based on FCA for existing specifications. A number of specification repositories are available on the web for methods including Estelle, LOTOS, Z, and Alloy. These specifications vary in scale and complexity and provide a basis for implementation and tool benchmarking.

The proposed approach is to exploit the scalable work described in Section 3.2 for mapping CGs into Formal Contexts via the Power Context Family. A specification can then be stored in the tables of a relational database and existing tools like TOSCANA can be used for data analysis, exploration and visualisation. Before this can be done, however, a mapping of a suitable formal method into CGs must be found. This will be the focus of the research for the start of 2001.

Z and its dialects [1] have proven popular for embedding in other languages. An example is the use of the HOL interactive theorem prover with Z described in Section 4.4. Simple mappings of Z into CGs have been attempted and a reverse mapping of Constraint Graphs into an ASCII based dialect of Z called ZSL is presented in [59].

Alloy is another ASCII based Z derivative that is both relational and declarative. The underlying data structures in Alloy are sets and relations. These characteristics make Alloy an ideal candidate for embedding in CGs. The close lineage from Z means many existing Z and VDM specifications can be recast in Alloy and the mapping to UML could facilitate an alternate analysis of UML via FCA.

The process of embedding Alloy in CGs requires both the application of theory and software engineering. While the resulting mapping implementation will be new the approach provides maximal reuse of existing languages (Alloy), formalisms (CGs, FCA, PCF) and tools (TOSCANA and WebKB [2]). This contribution of this research is in the mapping from a Z-like language to CGs and in the application of existing tools and techniques to provide alternate representations of formal methods.

By embedding a derivative of Z the approach is open for ready comparison against existing systems such as HOL. This will be the focus of research for the remainder of 2001.

[1] see http://archive.comlab.ox.ac.uk/z.html
[2] see http://meganesia.int.gu.edu.au/ phmartin/WebKB/

5.3 Schedule

A proposed schedule for conducting and completing the proposed research is shown in Table 5.3.

	2001			**2002**		
Task	**Jan-Apr**	**May-Aug**	**Sep-Dec**	**Jan-Apr**	**May-Aug**	**Sep-Dec**
FM Specification to CG Mapping	$\times$					
Tools Comparison		$\times$	$\times$			
Software Implementation	$\times$	$\times$	$\times$	$\times$	$\times$	
Conference Paper Submission			$\times$	$\times$	$\times$	
Journal Paper Submission			$\times$	$\times$	$\times$	
Thesis Preparation				$\times$	$\times$	$\times$

Table 5.1: Proposed schedule for conducting and completing the research.

Bibliography

[1] J. Sowa, "A brief introduction to conceptual graphs." `http://www.cs.uah.edu/ delugach/CG/Sowa-intro.html`, 1999.

[2] D. Jackson, "A comparison of object modelling notations: Alloy, UML and Z." Unpublished Manuscript, August 1999.

[3] B. Groh and P. Eklund, "Algorithms for creating relational power context families from conceptual graphs," in *Conceptual Structures: Standards and Practices* (W. Tepfenhart and W. Cyre, eds.), Lecture Notes in Artificial Intelligence, (Berlin), pp. 389–400, Springer Verlag, 1999.

[4] R. Cole and P. Eklund, "Scalability in formal concept analysis," *Computational Intelligence,*, vol. 15, no. 1, pp. 11–27, 1999.

[5] R. Cole, P. Eklund, and G. Stumme, "CEM - a program for visualization and discovery in email," in *4th European conference on principles and practice of knowledge discovery in databases,*, Springer Verlag,, September 2000.

[6] N. Spangenberg and K. E. Wolff, "Comparison between biplot analysis and formal concept analysis of repertory grids," in *Classification, data analysis, and knowledge organization.*, pp. 104–11, Berlin-Heidelberg: Springer, 1991.

[7] N. Spangenberg, "The conceptual structure of countertransference associations: an examination of diagnostic associations through an analysis of their semantic features.," in *Psychoanalytic research by means of formal concept analysis.*, Special des Sigmund-Freud-Instituts, Münster: Verlag, 1999.

[8] B. Ganter and R. Wille, "Conceptual scaling," in *Applications of combinatorics and graph theory to the biological and social sciences* (F. Roberts, ed.), pp. 139–167, New York: Springer-Verlag, 1989.

[9] P. Funk, A. Lewien, and G. Snelting, "Algorithms for concept lattice decomposition and their applications," Tech. Rep. 95-09, TU Braunschweig, December 1995.

[10] C. Lindig, "Concept-based component retrieval," in *Proceedings of IJCAI-95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, August 1995.

[11] G. Snelting, "Reengineering of configurations based on mathematical concept analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 5, pp. 146–189, April 1996.

[12] G. Snelting and F. Tip, "Reengineering class hierarchies using concept analysis," in *Proceedings of ACMSIGSOFT Symposium on the Foundations of Software Engineering*, pp. 99–110, November 1998.

[13] J. Sowa, *Conceptual Structures: Information Processing in Miind and Machine.* Addison-Wesley systems programming series, Reading, Massachusetts: Addison-Wesley, 1984.

[14] H. Dulugach, M. Keeler, D. Lukose, L. Searle, and J. Sowa, eds., *Conceptual Graphs and Formal Concept Analysis*, no. 1257 in Lecture Notes on Computer Science, (Berlin), Springer Verlag, 1997.

[15] A. Diller, *Z: An Introduction to Formal Methods.* Chichester: John Wiley and Sons, 2nd ed., 1994.

[16] R. Wille, "Restructuring lattice theory: An approach based on hierarchies of concepts," *Ordered Sets*, pp. 445–470, 1982.

[17] B. Ganter and R. Wille, *Formal Concept Analysis - Mathematical Foundations.* Berlin: Springer-Verlag, 1999.

[18] R. Cole, *The Management and Visualisation of Document Collections Using Formal Concept Analysis.* PhD thesis, Griffith University, School of Information and Communication Technology, 2000. in press.

[19] G. Stumme, "Local scaling in conceptual data systems," in *Conceptual Structures: Knowledge Representation as Interlingua* (P. Eklund, G. Ellis, and G. Mann, eds.), LNAI 1115, (Berlin), pp. 308–320, Springer Verlag, August 1996.

[20] B. Groh, S. Strahringer, and R. Wille, "Toscana-systems based on thesauri," in *Proceedings of the 6th International Conference on Conceptual Structures*, LNAI 1453, (Berlin), pp. 127–138, Springer Verlag, 1998.

[21] P. Eklund, B. Groh, G. Stumme, and R. Wille, "A contextual-logic extension of TOSCANA," in *7th International Conference on Conceptual Graphs,*, (Berlin), pp. 453–467, Springer Verlag, August 2000.

[22] P. Martin, "Conventions and notations for knowledge representation and retrieval," in *Proceedings of ICCS 2000, 8th International Conference on Conceptual Structures*, LNAI 1867, pp. 41–54, Springer Verlag,, August 2000.

[23] D. Lukose, "MODEL-ECS: Executable conceptual modelling language," in *KAW96*, 1996.

[24] G. Mineau, G. Stumme, and R. Wille, "Conceptual structures represented by conceptual graphs and formal concept analysis," in *Conceptual Structures: Standards and Practices* (W. Tepfenhart and W. Cyre, eds.), Lecture Notes in Artificial Intelligence, (Berlin), pp. 423–441, Springer Verlag, 1999.

[25] O. Corby, R. Dieng, and C. Hébert, "A conceptual graph model for W3C resource description framework," in *Conceptual Structures: Logical, Linguistic, and Computational Issues* (B. Ganter and G. Mineau, eds.), Lecture Notes in Artificial Intelligence, (Berlin-Heidelberg), pp. 465–478, Springer, 2000.

[26] F. Southey and J. Linders, "NOTIO - a Java API for developing CG tools," in *Conceptual Structures: Standards and Practices* (W. Tepfenhart and W. Cyre, eds.), Lecture Notes in Artificial Intelligence, (Berlin), pp. 262–271, Springer, 1999.

[27] P. Martin and P. Eklund, "Embedding knowledge in web documents," in *Proceedings of WWW8, 8th International World Wide Web Conference*, pp. 324–341, Elsevier, 1999.

[28] P. Bruza and S. Dennis, “Query reformulation on the internet: Empirical data and hyperindex search engine,” in *Proceedings of the RIAO97 Conference - Computer-Assisted Information Searching on Internet*, Centre de Hautes Etudes Internationales dÍnformatique Documentaires, 1997.

[29] P. Bruza, R. McArthur, and S. Dennis, “Interactive internet search: Keyword, directory and query formulation mechanisms compared,” in *Proceedings of the SIGIR Conference, 2000*, 2000.

[30] “DSTC Pty Ltd.” `http://www.dstc.edu.au`.

[31] R. Vienneau, “A review of formal methods,” in *Software Engineering* (M. Dorfman and R. Thayer, eds.), Computer Society Press, 1996.

[32] J. Bowen and M. Hinchey, eds., *Applications of Formal Methods*. London: Prentice Hall, 1996.

[33] K. Turner, ed., *Using Formal Description Techniques (An Introduction to Estelle, LOTOS and SDL)*. Wiley series in Communication and Distributed Systems, Chichester: John Wiley and Sons Ltd., 1993.

[34] NASA, *Formal Methods Specification and Verification Guidebook for the Software and Computer Systems*, vol. 1. Washington, DC: National Aeronautics and Space Administration, December 1998. ”NASA/TP-98-208193”.

[35] NASA, *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems*, vol. 2. Washington, DC: National Aeronautics and Space Administration, May 1997. NASA-GB-001-97.

[36] R. Pressman, *Software Engineering: a practitioner’s approach*. McGraw-Hill, third ed., 1992.

[37] J. Bowen and M. Hinchey, “Ten commandments of formal methods,” *IEEE Computer*, vol. 28, pp. 56–63, April 1995.

[38] M. Feather, “Low-cost pathways towards formal methods use,” in *Proceedings of the second workshop on Formal Methods in software practice*, pp. 85–91, ACM, 1998.

[39] A. Tanenbaum, *Computer Networks*, ch. 1, pp. 28–43. New Jersey: Prentice-Hall, 3rd ed., 1996.

[40] G. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall Software Series, New Jersey: Prentice-Hall, 1991.

[41] M. Fecko, M. Uyar, P. Amer, A. Sethi, T. Dzik, R. Menell, and M. McMahon, “A success story of formal description techniques: Estelle specification and test generation for mil-std 188-220,” *Computer Communications (special issue)*, vol. 23, Spring 2000.

[42] ISO/IEC JTC1/SC21/WG7, “Final commitee draft on enhancements to LOTOS,” tech. rep., ISO/IEC, May 1998. Project WI 1.21.20.2.3.

[43] ITU-T, “Recommendation Z.120: Message sequence chart (MSC),” tech. rep., ITU-T, Geneva, 1993.

[44] J. Spivey, *Understanding Z: A specifciation language and its formal semantics*. Cambridge University Press, 1988.

[45] S. Agerholm and P. Larsen, “A lightweight approach to formal methods,” in *Applied Formal Methods – FM–Trends 98* (D. Hutter, W. Stephan, P. Traverso,

and M. Ullman, eds.), LNAI 1641, (Berlin), pp. 168–183, Springer Verlag, October 1998.

[46] D. Jackson, I. Schechter, and I. Shlyakhter, "Alcoa: the alloy constraint analyzer," in *Proceedings of the International Conference on Software Engineering*, (Limerick, Ireland,), June 2000.

[47] G. B. amd I. Jacobson and J. Rumbaugh, *The unified modeling language user guide*. Addison-Wesley object technology series, Reading, Massachusetts: Addison-Wesley, 1999.

[48] D. Jackson and M. Vaziri, "Some shortcomings of OCL, the object constraint language of UML." `http://sdg.lcs.mit.edu/~dnj/publications.html`, December 1999.

[49] A. Hall, "Seven myths of formal methods," *IEEE Software*, pp. 11–19, September 1990.

[50] K. Abernethy, J. Kelly, A. Sobel, J. Powell, and J. D. Kiper, "Technology transfer issues for formal methods of software specification," in *Thirteenth Conference on Software Engineering Education and Training*, IEEE, March 2000.

[51] J. Bowen and M. Hinchey, "Seven more myths of formal methods," *IEEE Software*, vol. 12, pp. 34–41, July 1995.

[52] R. Butler and C. Holloway, "Impediments to industrial use of formal methods," *IEEE Computer*, pp. 25–26, April 1996.

[53] University of Delaware Protocol Engineering Laboratory, "Protocol specification in Estelle at univ of delaware." `http://www.eecis.udel.edu/~amer/PEL/estelle/index.html`, April 2000.

[54] S. German, "Research goals for formal methods," *ACM Computing Surveys*, vol. 28, December 1996.

[55] E. Clarke and J. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, vol. 28, pp. 626–643, December 1996.

[56] E. Börger, "High level system design using abstract state machines," in *Applied Formal Methods – FM–Trends 98* (D. Hutter, W. Stephan, P. Traverso, and M. Ullman, eds.), LNAI 1641, (Berlin), pp. 1–43, Springer Verlag, October 1998.

[57] M. Chechik, "SC(R)^3: Towards usability of formal methods," in *Proceedings of CASCON'98*, pp. 177–191, November 1998.

[58] I. Vessey, "Cognitive fit: A theory-based analysis of the graphs versus tables literature," *Decision Sciences*, vol. 22, no. 2, pp. 219–240, 1991.

[59] R. Kremer, *Constraint Graphs: A Concept Map Meta-Language*. PhD thesis, Department of Computer Science, University of Calgary, 1997. `http://www.cpsc.ucalgary.ca/~kremer/dissertation`.