A COMPARISON OF TWO TERMINOLOGICAL KNOWLEDGE REPRESENTATION SYSTEMS

A thesis submitted to the University of Manchester for the degree of
Master of Science (Computer Science)
in the Faculty of Science

by
Ian R. Horrocks
Department of Computer Science
July 1995

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

GRAIL and LOOM, two terminological knowledge representation systems, are compared both qualitatively and quantitatively. The objective is to achieve a better understanding of GRAIL and its relation to other KL-ONE languages, to empirically compare the performance of the two terminological classifiers, and to contrast GRAIL's custom design with LOOM's general purpose design.

GRAIL has been specially developed to represent knowledge about medical terminology; it has some powerful and original features but a restricted terminological language. LOOM has been designed to satisfy a wide range of knowledge representation requirements by combining a highly expressive terminological language with an efficient classifier.

The quantitative comparison tests scaleability by measuring the rate of performance degradation with increasing knowledge base size. A large GRAIL knowledge base (2,000 concepts) from a real application (the GALEN project) was used for the experiment and translated into LOOM for the purpose. Both systems performed well in view of the tractability problems associated with KL-ONE languages and showed no sign of an exponential explosion in classification time.

The qualitative comparison is based on a set theoretic account of the semantics of the two languages and on the experience of attempting to translate a GRAIL knowledge base into LOOM. Although LOOM's terminological expressiveness allows it to represent concepts which are difficult or impossible to represent in GRAIL, it proved impossible to satisfactorily translate GRAIL's special features into LOOM. However, a detailed study of GRAIL revealed some serious problems with the classifier which, in its current form, is shown to be both incomplete and unsound.

DECLARATION

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

ACKNOWLEDGEMENTS

I gratefully acknowledge the help and support of all the members of the Medical Informatics Group, University of Manchester, without whose assistance the work embodied in this thesis would not have been possible. I particularly wish to thank Carole Goble, Alan Rector, Sean Bechhofer, Peter Crowther and Delma McDermott.

I also wish to thank Tom Russ and Robert MacGregor of USC/Information Sciences Institute for their help with LOOM.

This work was supported by a grant from the Engineering and Physical Sciences Research Council.

THE AUTHOR

Ian Horrocks obtained a B.Sc. in Computer Science from Manchester University in 1981. After graduation he stayed in the Computer Science Department working as a Research Assistant first in the Barclays microprocessor laboratory and later with the dataflow parallel architecture group.

In 1983 he joined Scientex Limited as Technical Director and was responsible for the development of a number of word processing and desk-top publishing products.

His research interests are knowledge representation and, in particular, terminological logics.

CHAPTER 1 INTRODUCTION

1.1 The purpose of this thesis

As part of the European GALEN[1] project, researchers in Manchester University's Medical Informatics Group are building a large concept model representing knowledge about medical terminology. GRAIL[2], a terminological knowledge representation system (TKRS) in the KL-ONE tradition, has been developed specifically for the task of building the concept model. This thesis compares GRAIL with LOOM, a state of the art general purpose TKRS with the aim of:

- better understanding GRAIL's semantics and expressive power through a formal comparison using set theoretic interpretations;
- comparing scaleability by measuring the rate of performance degradation with increasing knowledge base size;
- evaluating both systems with particular reference to their ability to satisfy the requirements of the medical terminology application;
- enhancing GRAIL's future development through the detailed study of an alternative system.

This chapter describes the motivation for GRAIL's development, explains why LOOM was chosen for the comparison and details the bases of the comparison. After a brief note on nomenclature the chapter concludes with an outline of the remainder of the thesis.

1.2 Motivation for the development of GRAIL

The development of GRAIL [GBS+94], and its predecessor SMK[3] [NR91], has been motivated by the requirements of two medical informatics research projects: PEN&PAD [NRK+90] which is developing clinical information systems with predictive data entry and GALEN which "aims to develop language independent concept representation systems as the foundations for the next generation of multilingual coding systems" [RNG93]. Both projects make use of knowledge about medical terminology represented in the GRAIL concept model.

The GALEN project uses the concept model and GRAIL classifier to drive a networked terminology server (TeS) [RSNR94]. It is intended that such servers will facilitate the integration of medical informatics applications and the sharing of medical data by providing sophisticated terminology and coding services. PEN&PAD's design incorporates an advanced user interface which uses knowledge from the model to predict what extra detail a clinician might want to add to a concept description.

[1] Generalised Architecture for Languages Encyclopaedias and Nomenclatures in Medicine.

[2] The GALEN Representation And Integration Language.

[3] Structured Meta Knowledge.

1.3 Why use a Terminological Knowledge Representation System?

The requirement for a standard coding system for medical data has long been recognised and attempts to solve the problem using enumerative coding schemes[4] date back over 200 years [Now93]. The size, complexity and diversity of medicine make the development of a comprehensive system a difficult and probably infeasible task – increasing expressiveness causes a combinatorial explosion in the number of codes needed [RNG93]. Table 1.1 shows how adding information about location, degree, aetiology and cause increases the number of codes required to represent a burn from 1 to 12,000.

Table 1.1 – Combinatorial explosion in static coding schemes

Description	Additional Detail	Number of Codes
Burn	none	**1**
Burn+Location	≈200 anatomical locations	200 × 1 = **200**
Burn+Location+Degree	4 degrees – 1st/2nd/3rd/not-known	4 × 200 = **800**
Burn+Location+Degree +Aetiology	3 aetiologies – chemical/thermal/ not-known	3 × 800 = **2,400**
Burn+Location+Degree +Aetiology+Cause	5 (say) causes – home/work/ traffic-accident/other/not-known	5 × 2,400 = **12,000**

Multiaxial schemes allow terms from a number of broad axes, such as topography, morphology and aetiology, to be combined to form complex codes. While representing an advance over simple enumerative schemes multiaxial systems still impose a rigid structure, have limited expressiveness, and rely on enumeration within the axes. Multiaxial systems also introduce new problems of their own including vague semantics and the possibility of creating nonsensical terms [Now93].

The GALEN project aims to improve on static and multiaxial coding schemes by using the GRAIL TKRS to build a concept model of medical terminology. TKRSs are designed to support “the definition of complex concepts and the discovery of their interrelationships” [BMPSR91]. The features of TKRSs directly address many of the problems associated with static and multiaxial coding schemes:

- unlimited expressiveness – there are no pre-defined axes and concepts can be combined and specialised in an arbitrary manner and to an arbitrary extent;
- unambiguous semantics – the relations between concepts and the meaning of complex terms have a clearly defined “criterial semantics” [WS92];
- facilitates extension and maintenance – automatic classification, inheritance and the detection of inconsistencies aid knowledge acquisition.

TKRSs have also been shown to be useful in the integration of heterogeneous databases [NSA+94] and in knowledge sharing [PFPS+92]. Database integration is important given the number of different coding schemes already in use; sharing and re-usability are central to the aims of the GALEN project.

[4] Schemes which list medical concepts and assign each concept a unique code.

1.4 Why compare GRAIL with another TKRS?

A wide range of TKRSs, based on the KL-ONE paradigm [BS85], have now been developed by research groups in America and Europe. These include BACK [Pel91], CLASSIC [PS91], CycL [LG91], K-Rep [MDW91], KRIS [BH91], LOOM [Mac91b] and SB-ONE [Kob91]. The range of systems now available has been said to make it "reasonable to build upon an existing terminological system instead of building one from scratch" [HKNP94].

GRAIL is a special purpose system tailored to the requirements of a specific application. Comparing GRAIL with a modern general purpose TKRS is intended to show whether GRAIL's custom design offers significant advantages in meeting the application requirements. It is also hoped that a more formal analysis of GRAIL's semantics and a better understanding of another TKRS will prove useful in GRAIL's future development.

1.4.1 Why compare GRAIL with LOOM?

LOOM was chosen for the comparison due to its combination of expressiveness and efficiency [HKNP94]. The power and flexibility of LOOM's terminological language made it likely to provide a stringent and revealing test for GRAIL. The efficiency of LOOM's classifier would also provide an interesting comparison. If LOOM is unable to satisfy GRAIL's design goals it is unlikely that this could be achieved by less expressive and less efficient systems.

LOOM has the added advantage of being a relatively mature system with an established user base. LOOM is made available to approved researchers via an ftp site and the LOOM development team provide technical support via a mailing list; the mailing list also acts as a forum for the LOOM user community to exchange ideas and information.

1.5 The bases of comparison.

The two systems are compared both qualitatively and quantitatively on the basis of their features, the performance of their classifiers and their ability to satisfy the requirements of the medical terminology application.

1.5.1 Features

The features and expressive capabilities of the two languages are compared by giving a formal account of their semantics using a slightly extended version of the terminological logic proposed by Baader *et al.* [BHH+91]. An informal extensional semantics is also provided based on the operational descriptions in the LOOM Reference Manual [Bri93].

1.5.2 Performance

Theoretical complexity analyses of subsumption and classification have produced discouraging results [LB85], [Neb88], [PS89], [SS89]. Nebel has shown that even the least expressive languages must have worst case complexity which is at least co-NP-complete

[Neb90]. Fortunately Nebel goes on to observe that the pathological cases which give rise to these results are rarely encountered in applications.

This thesis empirically compares the performances of GRAIL and LOOM using a large knowledge base from a real application – the CORE[5] model from the GALEN project [RGG+94]. The CORE model consists of 2,128 concepts and 416 relations providing basic terms and definitions intended to act as a foundation for the extension and expansion of the medical terminology knowledge base. The scaleability of performance is measured by gradually increasing the size of the knowledge base up to a maximum of 3,987 concepts[6].

1.5.3 Satisfying the application requirements

The capabilities of the two systems are examined, with particular reference to the requirements of the medical terminology server application, in order to ascertain:

- if a highly expressive system like LOOM can emulate those features of GRAIL designed to meet specific application requirements;
- if LOOM's extra expressiveness is applicable to types of knowledge known to be difficult to represent using GRAIL;
- how easy they are to use and what tools are provided to help with knowledge acquisition;
- how stable, robust and reliable they are.

1.6 A note on nomenclature

There is an unfortunate degree of confusion and disagreement surrounding the vocabulary used to talk about TKRSs. In particular the terms relation, role and attribute have been given a number of different interpretations. To promote clarity a brief interpretation is given here and used consistently throughout regardless of the system being discussed:

concept	an intensional description which represents a set of objects in the domain. For example: **Person** or **Female**.
individual	a unique object in the domain[7]. For example: **John** or **Mary**.
relation	an intensional description which represents a set of binary tuples relating pairs of objects in the domain[8]. For example: **has-child** or **is-sibling-of**.

[5] COmmon REference model.

[6] This figure has no special significance – sections 4.4.2 and 4.4.3 explain how it comes about.

[7] It may be more accurate to consider individuals as representing disjoint sets of objects in the domain [BPS94].

[8] LOOM is capable of supporting n-ary relations.

instance	an individual member of a concept or relation’s extension.
	For example: **Mary** is an instance of both Person and Female; **(Mary, John)** is an instance of has-child.
role	a concept forming expression which involves a relation.
	For example: (some **has-child** Person) forms a concept which represents the set of objects which are related to an instance of Person via the has-child relation.
filler	the filler of a role is an object in the domain which is a value of a role. Range restricted roles can only be filled by instances of the range restricting concept; unrestricted roles can be filled with arbitrary objects (such as atoms, lists or numbers) which may or may not be instances of some concept.
	For example: **John** is a filler of the role has-child on Mary.
attribute	a role which must have exactly 1 filler.
	For example: (Person $\wedge$ (**has-spouse** Person)) might be used as the definition of a concept called Married-Person. All instances of Married-Person must be instances of Person and must be related to exactly one other instance of Person via the **has-spouse** attribute[9] .
criterion	the GRAIL name for a role or attribute. Criteria constructed from many valued relations are equivalent to roles while those constructed from single valued relations are equivalent to attributes.
	For example: **<has-child Person>** $\equiv$ (some has-child Person); **<has-spouse Person>** $\equiv$ (has-spouse Person).

1.7 Outline of the remainder of this thesis

Chapter 2 describes the rationale behind the design of GRAIL and explains how its features were determined by the application requirements.

Chapter 3 compares the terminological services provided by the two systems through a formal analysis of their semantics.

Chapter 4 describes the translation of the CORE model into LOOM and the design of the performance comparison experiments.

Chapter 5 presents and analyses the results of the performance comparison experiments.

Chapter 6 compares the two systems and assesses their performance in meeting the design goals set by the medical terminology application.

Chapter 7 is the discussion and conclusions – it summarises what has been learned and suggests directions for future work.

[9] Internally LOOM translates attributes into roles, adding a cardinality restriction of exactly 1.

CHAPTER 2 THE DESIGN OF GRAIL

This chapter briefly describes the rationale behind the design of SMK and GRAIL and explains how their features were determined by the requirements of the terminology server application. A detailed study of the work which led to the identification of these requirements is beyond the scope of this thesis; the interested reader is referred to [Now93].

2.1 GRAIL design goals and solutions

GRAIL has been designed as a tool for a single specific task: to build a concept model of medical terminology[10] which can be used in both the PEN&PAD and GALEN projects. The primary requirements for the model are that it should be reusable and extensible while still being computationally tractable.

2.1.1 Concept only terminological model

Brachman has stated that a key element in the design of a terminological knowledge base is determining the correct type (concept, relation or individual) for objects in the domain [BMPSR91]. GRAIL aims to promote re-usability by providing terminological services at the concept and relation level. Implementors are free to make their own design decisions – based on the requirements of a particular application – about the level of detail which is appropriate for individuals. By using the concept model as a classification schema applications can be independently developed while still guaranteeing data interoperability.

2.1.2 Restricted expressiveness

Brachman and Levesque have shown that there is a “fundamental trade-off” between expressiveness and computational tractability in knowledge representation [LB85]. TKRSs are no exception to this rule – Brachman and Levesque go on to demonstrate that even a very small increase in the expressive power of a terminological language can drastically affect tractability[11] .

If GRAIL is to meet the requirements of the GALEN project it will have to be capable of supporting a very large knowledge base and classifying new concepts in real time. Computational tractability is therefore of crucial importance and in order to minimise the complexity of classification, the expressiveness of GRAIL’s concept and relation forming operators is severely restricted – the constructs supported are intended to be just those which are necessary for the modelling of medical terminology [RNK92].

[10] GRAIL may be a useful tool in other application areas but this has yet to be clearly demonstrated.

[11] Addition of the *restr* operator to a simple frame description language changes subsumption from polynomial to co-NP-hard.

2.1.3 *Necessary* statements – terminologically significant assertions

Most KL-ONE derived TKRSs provide for extra non-definitional characteristics to be asserted about concepts. Such assertions represent characteristics which, while true of a concept, are not essential for the recognition of an individual as a member of the concept's extension. In systems which follow the KRYPTON [BFL83] philosophy of strictly separating terminological and assertional knowledge (such as KANDOR [PS84], CLASSIC, LOOM and BACK), assertions do not affect the classification of concepts by the terminological classifier (T-box); in these systems assertions only affect inferences about individuals made by the assertional reasoner (A-box).

The strict separation of terminological and assertional knowledge has been questioned by Doyle and Patil, with particular reference to medical knowledge representation, on the grounds that it severely restricts the usefulness of classification based reasoning [DP91]; they suggest that the correct classification of some concepts is dependant on assertional knowledge. Woods has also shown that the rejection of assertional knowledge restricts the kinds of facts that can be represented by a terminological system [Woo91].

It has also been pointed out by Doyle and Patil [DP91], and by MacGregor [Mac91a], that the extreme stance taken by KRYPTON has been softened in almost all subsequent systems so as to allow some assertional knowledge to be represented in the T-box. Subsumption relations between primitive concepts and the definition of disjoint concepts are examples of assertional knowledge which is visible to the classifier in many TKRSs.

GRAIL takes this process one stage further by making certain knowledge which would be treated as assertional in other TKRSs visible to the classifier through ***necessary*** statements. This enhances the utility of classification based reasoning and GRAIL's ability to reduce concepts to a canonical form. In GRAIL the assertion that "all cancers are necessarily severe" allows "cancer" and "severe cancer" to be recognised as the same concept while still classifying all "cancers" as kinds of "severe conditions". A classifier which ignored assertional knowledge would give a very different result: asserting "severe" as a characteristic of "cancer" would not result in "severe cancer" being recognised as the same concept as "cancer" nor in the classification of all "cancers" as kinds of "severe conditions"; including the "severe" characteristic in the definition of "cancer" would require that a condition be explicitly described as "severe" before it could be classified as a kind of "cancer".

2.1.4 Sanctioning – controlled genericity

In most TKRSs there is no restriction on the generation of new concepts. The classifier will detect concepts whose definitions are logically inconsistent and classify them as incoherent[12] but there is nothing to prevent the creation of concepts whose definitions, while logically consistent, are nonsensical. Examples such as "fractured lung" and "severe aspirin" are easily generated in unconstrained systems.

[12] An incoherent concept is one who's extension can be proved to be empty, for example as the result of conflicting role cardinality restrictions.

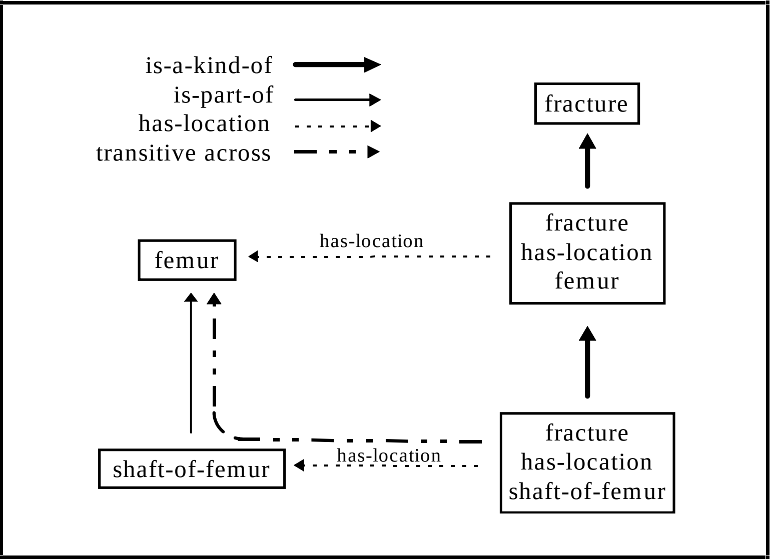
As well as detecting incoherent concepts GRAIL provides a hierarchical sanctioning mechanism which constrains the generation of new concepts. New specialisations – non-primitive concepts created by adding criteria to existing concepts – are checked by GRAIL and rejected if any criterion is not sanctioned. Sanctioned specialisations do not add any new knowledge but are a kind of lazy evaluation of the model – the existence of such concepts can be inferred from sanctioning knowledge but they are only defined and installed in the model when required by an application. Sanctioning supports arbitrary expressiveness and tractability by representing a potentially very large or even infinite number of concepts in a sparse model which can be dynamically extended in response to application demand.

Sanctioning also addresses some of the problems associated with knowledge acquisition. The construction and maintenance of a concept model of medical terminology sufficiently comprehensive to meet the goals of the GALEN project will be a large and difficult task; multi-level sanctioning provides a mechanism for the guidance and control of this task.

2.1.5 Refinement and transitivity – co-ordinating taxonomies

Subsumption or is-a-kind-of relations form the backbone of the taxonomic hierarchy in a TKRS. GRAIL is designed to support the construction of multiple taxonomies of basic medical concepts which can be combined to form more complex terminological concepts; some of these taxonomies are naturally based on the subsumption relation but others are not. In particular taxonomies which deal with physically composed objects, for example in anatomy, are more naturally described using the transitive part-whole relation [PL94].

Figure 2.1 – has-location transitive across part-of



In a subsumption based taxonomy all relations are transitive across the special is-a-kind-of relation. GRAIL's refinement mechanism supports the co-ordination of taxonomies by allowing the definition of relations which are transitive across other user-defined relations. Figure 2.1 shows how refinement enables the classifier to recognise that a "fracture which has location shaft of the femur" is a kind of "fracture which has location femur" in spite of the "shaft of the femur" being a part of and not a kind of "femur". In this example the co-ordination of the anatomical and process taxonomies requires that the “has-location” relation be transitive across the “is-part-of” relation.

In some TKRSs, including LOOM, refinement can be represented using assertions; CycL, the TKRS used in the Cyc project, recognises the need to represent this kind of knowledge and provides the ***TransfersThro*** statement which has been optimised for efficient truth maintenance [LG89]. However these representations only effect assertional inferences and not T-box classification. While some TKRSs do support T-box transitivity [HKQ+93] – the special case of refinement where a relation is transitive across itself – support for refinement in the T-box is unique to GRAIL.

2.2 Summary

Table 2.1 summarises GRAIL’s design goals and the solutions adopted. The result is a system which, while clearly in the KL-ONE tradition, has a distinctly different set of features to those normally found in a KL-ONE derived TKRSs.

Table 2.1 – GRAIL design goals and solutions

reusable	• **concept only** terminological model acts as an application independent classification schema; • ***necessary*** **statements** enhance the reduction of concepts to a canonical form and the recognition of trivial variants; • ***necessary*** **statements** support varying requirements for descriptive detail by minimising concept definitions while providing full classification based on asserted characteristics; • **refinement and transitivity** co-ordinate taxonomies based on relations other than is-a.
extensible	• **restricted expressiveness** and a simple and transparent terminological language which is easily understood by model builders; • **sanctioning** and constrained genericity allows large numbers of concepts to be created from a sparse model; • **sanctioning** helps guide model builders by controlling how the model can be extended.
tractable	• **restricted expressiveness** supporting only those features which are necessary for modelling medical terminology. • **sanctioning** and constrained genericity allow the size of the basic model to be kept to a minimum.

CHAPTER 3 FEATURE COMPARISON

This chapter compares the features of the two languages and the terminological services they provide. Their expressive capabilities are interpreted using a combination of set theoretic and informal descriptive semantics adapted from [BHH+91], [WS92], [HKNP94] and [Brill93].

3.1 Descriptive semantics

The symbols used in concept and relation definitions are summarised in table 3.1. Concept, relation, attribute and individual names are assumed to be unique. Concept expressions define a new concept in terms of existing concepts; for example the concept Man could be defined as the conjunction of the concepts Person and Male. Relation expressions define a new relation in terms of existing relations; for example the relation daughter could be described as the relation child with its range restricted to the concept Female.

Table 3.1 – Symbols

Symbol	Meaning	Examples
CN	Concept Name	Person, Male, Female
RN	Relation Name	child, sibling
AN	Attribute Name	mother, spouse
IN	Individual Name	John, Mary
C	Concept expression	Person $\wedge$ Male *(i.e. Man)*
R	Relation expression	child$\vert_{\text{Female}}$ *(i.e. daughter)*
A	Attribute expression	spouse$\vert_{\text{Female}}$ *(i.e. wife)*

Concept and relation forming operators are described using a generalised LISP-like syntax (concrete form), logic symbols (abstract form), and an informal extensional semantics.

- A concept C is defined by the set of individuals I_n which form its extension in the domain Δ.
- Concept forming operators are defined in terms of the extensions of the new concept and those of the composing concepts. For example if an individual I is in the extension of concept C which is formed from the conjunction $C_1 \wedge ... \wedge C_n$, I must be in the extensions of each of $C_1 ... C_n$.
- A relation R is defined by the set of tuples $(I_1..I_n)$ which form its extension. In the case of binary relations, this is the set of (I_1,I_2) which form a subset of $\Delta \times \Delta$.
- Relation forming operators are defined in terms of the extensions of the new relation and those of the composing relations. For example if a tuple (I_1,I_2) is in the extension of R

which is formed from the disjunction $R_1 \vee ... \vee R_n$, (I_1,I_2) must be in the extension of one of $R_1...R_n$.

3.2 LOOM overview

LOOM offers a complete high level programming environment for the development of knowledge based systems and applications [LOOM93]. As well as a term classifier (T-Box) and assertional reasoner (A-Box) LOOM provides integrated support for two object oriented programming paradigms: pattern-directed programming (methods) and data-driven programming (production rules). LOOM's terminological language is highly expressive and forms a superset of most other languages [HKNP94], [WS92]. Assertion and retrieval is based on a query language which embodies the full first order predicate calculus. The LOOM system is under continued development with new features regularly being added including, in version 2.1, a context mechanism, negation and temporal extensions [LOOM94].

Some systems, most notably CLASSIC, restrict expressiveness in order to guarantee tractability and completeness[13] [BPS94]. LOOM provides greater expressiveness by supporting terminological constructions which are known to be intractable or even undecidable[14]. LOOM's designers acknowledge that the classifier must therefore be incomplete; this is justified on the grounds that complete systems, while of theoretical interest, are too restrictive to be of use in most applications [Mac94]. Users are said to consistently demand more functionality and it is suggested that additional features are likely to be better designed and controlled as part of the system rather than being implemented by users on an ad hoc basis [Mac91a].

Borgida has pointed out that one problem with this approach is the difficulty of characterising to users the exact circumstances which will result in incomplete reasoning or intractability [Bor92]. LOOM tackles the intractability aspect of this problem by providing a ***power-level*** function which allows the user to limit the computational effort expended in seeking "expensive types of inferencing" [Bri93]; however it is admitted to be "difficult to precisely characterise the types of inferencing affected" – users still have no precise indication as to when reasoning is incomplete.

LOOM is written in COMMON LISP and requires a full native CLOS. LOOM consists of a set of pre-defined concepts and relations along with an extensive library of functions, macros and methods numbering approximately 250. These provide the user with facilities for creating, manipulating and querying knowledge bases both interactively and from within applications. However no tools are provided to assist in these tasks beyond a few basic macros which provide textual listings detailing various aspects of the current state of the knowledge base.

3.3 LOOM semantics

Most of LOOM's pre-defined functions are designed to facilitate application programming by supporting the retrieval of information about concepts, relations and individuals. The

[13] A system is complete if it is guaranteed to find all valid inferences.

[14] An undecidable inference is one which no algorithm can be guaranteed to find regardless of the computational effort expended.

characteristics and functionality of the system are illustrated by describing the definition of concepts and relations. The definition of individuals and the formation of query expressions is also described.

3.3.1 Term classifier (T-box)

Concepts and Relations are defined using the ***defconcept*** and ***defrelation*** macros or their functional equivalents ***define-concept*** and ***define-relation***; the available concept and relation forming operators are described in tables 3.2 and 3.3 respectively while table 3.4 describes the available terminological axioms. LOOM does not allow incremental changes to the terminological definitions of concepts or relations but deletion and re-definition are supported.

Table 3.2 – LOOM concept forming operators

Concrete Form	Abstract Form	Semantics
top	$\top$	Δ – every I is an instance of top
bottom	$\bot$	$\varnothing$ – bottom has no instances
(and $C_1...C_n$)	$C_1\wedge...\wedge C_n$	I_c is an instance of all of $C_1...C_n$
(or $C_1...C_n$)	$C_1\vee...\vee C_n$	I_c is an instance of at least one of $C_1...C_n$
(not C)	$\neg C$	I_c is not an instance of C[15]
(one-of $I_1...I_n$)	$\{I_1...I_n\}$	I_c is one of $I_1...I_n$
(one-of $N_1...N_n$)	$\{N_1...N_n\}$	I_c is one of $N_1...N_n$
(through N_1 N_2)	$\{N_1–N_2\}$	I_c is in range N_1 to N_2
(at-least n R)	$\geq nR$	I_c:R has at least n fillers
(at-most n R)	$\leq nR$	I_c:R has at most n fillers
(exactly n R)	nR	I_c:R has exactly n fillers
(all R C)	$\forall R:C$	all fillers of I_c:R are instances of C
(some R C)	$\exists R:C$	some filler of I_c:R is an instance of C
(in A C)	A:C	the filler of I_c:A is an instance of C
(filled-by R $v_1...v_n$)	$R:v_1\wedge...\wedge R:v_n$	I_c:R is filled by all of $v_1...v_n$
(not-filled-by R $v_1...v_n$)	$\neg R:v_1\wedge...\wedge\neg R:v_n$	I_c:R is not filled by any of $v_1...v_n$
(eq R_1 R_2)	$R_1 = R_2$	$I_c:R_1$ and $I_c:R_2$ have the same fillers
(subset R_1 R_2)	$R_1 \subseteq R_2$	$I_c:R_1$'s fillers are a subset of $I_c:R_2$'s
($\{<,>,=,\neq\}$ A_1 A_2)	A_1 $\{<,>,=,\neq\}$ A_2	fillers of $I_c:A_1$ & $I_c:A_2$ are $\{<,>,=,\neq\}$
(relates R A_1 A_2)	A_1 R A_2	fillers if $I_c:A_1$ & $I_c:A_2$ are related by R
(satisfies (?X) Q)	$\forall I:Q(I)$	$Q(I_c)$ = True
(predicate (X) $f_1..f_j$)	$\forall I:P(I)$	(lambda (I_c) $f_1...f_j$) $\neq$ nil
(function () $f_1...f_j$)	{F()}	$I_c \in$ (lambda () $f_1...f_j$)
I_c is an instance of the concept formed by the described operator; $v_1...v_n$ are role values (LOOM or LISP objects); N_n is a member of the built-in concept *number* (can be integer or real); Q is a LOOM query expression; ($f_1...f_j$) are LISP forms.		

[15] LOOM's open-world assumption means that an individual can only be recognised as a member of (not C) if it is provably not a member of C, and vice versa.

Constructs can be combined to form definitions of arbitrary complexity. Inconsistent definitions are not treated as illegal but are classified under the built-in concept *incoherent*. It is also possible to use these constructs to attach assertions[16] to concepts and relations either at definition time or incrementally. Assertions can be made either strict, using the *:implies* keyword, or default, using the *:default* keyword. Strict assertions always apply whereas default assertions only apply if they do not result in incoherence. Assertions are ignored by the LOOM classifier.

Table 3.3 – LOOM relation forming operators

Concrete Form	Abstract Form	Semantics
(and $R_1 ... R_n$)	$R_1 \wedge ... \wedge R_n$	(I_1, I_2) is an instance of all of $R_1 ... R_n$
(domain C)	$_C\|R$	I_1 is an instance of C
(restrict C)	$R\|_C$	I_2 is an instance of C
(domains $C_1 ... C_{n-1}$)	$_{C_1 ... C_{n-1}}\|R$	if $(I_1, ..., I_n)$ is an instance of an n-ary R, $I_1 ... I_{n-1}$ are instances of $C_1 ... C_{n-1}$
(inverse R)	R^{-1}	(I_2, I_1) is an instance of R
(compose $R_1 ... R_n$)	R_1 o...o R_n	if (I_1, I_{n+1}) is an instance of R, then for j from 1 to n, there is an (I_j, I_{j+1}) which is an instance of R_j
(satisfies ($?X_1 ... ?X_n$) Q)	$\forall(I_1 ... I_n):Q(I_1 ... I_n)$	$Q(I_1 ... I_n)$ = True
(predicate ($X_1 ... X_n$) $f_1 ... f_j$)	$\forall(I_1 ... I_n):P(I_1 ... I_n)$	(lambda $(I_1 ... I_n)$ $f_1 ... f_j$) $\neq$ nil
(function ($X_1 ... X_{n-1}$) $f_1 ... f_j$)	{F()}	$I_n \in$ (lambda $(I_1 ... I_{n-1})$ $f_1 ... f_j$)

(I_1, I_2) is an instance of a binary relation formed by the described operator; Q is a LOOM query expression; $f_1 ... f_j$ are LISP forms

Additional characteristics can be attached to concepts and relations using key words. Among the more interesting of these are: *:backward-chaining* which instructs LOOM only to classify individuals in response to a query; *:closed-world* which allows LOOM to draw additional inferences by making the assumption that current information about individuals is complete; and *:monotonic* which tells LOOM that the recognition of individuals is indefeasible. The keywords *:partitions*, *:exhaustive-partitions* and *:in-partition* can be used to assert explicit disjunctions and disjoint coverings.

Table 3.4 – LOOM terminological axioms

Concrete Form	Abstract Form	Semantics
(defconcept CN C)	CN = C	$I \in CN \Leftrightarrow I \in C$
(defrelation RN R)	RN = R	$(I_1, ..., I_n) \in RN \Leftrightarrow (I_1, ..., I_n) \in R$
(defprimconcept CN C)	$CN \subseteq C$	$I \in CN \Rightarrow I \in C$
(defprimrelation RN R)	$RN \subseteq R$	$(I_1, ..., I_n) \in RN \Rightarrow (I_1, ..., I_n) \in R$
(defdisjoint $CN_1 ... CN_n$)	$CN_1 \|\| ... \|\| CN_n$	$I \in CN_j \Rightarrow I \notin CN_1 \vee ... \vee CN_{j-1} \vee CN_{j+1} \vee ... \vee CN_n$

[16] Rules which are implied by membership of a concept but which do not form part of its definition.

In spite of its complexity LOOM's classifier is comparatively efficient, although many classes of possible inference are not supported [HKNP94]. Reasoning about role cardinality restrictions, role value restrictions, role value maps, equality of attribute chains and inverse roles is known to be incomplete. The powerful satisfies, predicate and function constructs are completely opaque to the classifier: even identical definitions are not recognised as being equivalent. Role fillers are also ignored by the classifier except in so far as they impose minimum cardinality restrictions[17] .

3.3.2 Assertional reasoner (A-box)

Table 3.5 summarises the available assertional axioms. Recognition[18] and truth maintenance do not take place until the LOOM matcher is invoked by a call to the ***createm***, ***tellm*** or ***destroym*** macros. LOOM will then 'seal' the network, generating an error if there are any incompletely defined objects. The matcher re-computes the types of any modified instances and propagates changes throughout the knowledge base using a forward chaining algorithm [MB92]. The A-Box uses full concept and relation definitions for recognition and may succeed in finding some of the more difficult inferences which are missed or not implemented in the classifier [HKNP94].

Table 3.5 – LOOM assertional axioms

Concrete Form	Abstract Form	Semantics
(C IN)	IN $\in$ C	IN is an instance of C
(R $IN_1,...,IN_n$)	($IN_1,...,IN_n$) $\in$ R	($IN_1,...,IN_n$) is an instance of R
(same-as IN_1 IN_2)	$IN_1 = IN_2$	IN_1 and IN_2 are merged

3.3.3 Query language

Much of LOOM's assertional reasoning power is vested in its query and retrieval mechanism. Query expressions of arbitrary complexity can be formed using the operators summarised in table 3.6 and can return either a truth value or a set of matching objects from the knowledge base. Query expressions can also be included in concept and relation definitions using the ***satisfies*** construct. Individuals and tuples which satisfy the query expression will then be recognised by the A-Box as instances of the concept or relation.

[17] It has been shown to be necessary to ignore individuals as role fillers in order to maintain the monotonicity of classification when assertional retractions and redefinitions are permitted [Bor92].

[18] The classification of individuals in terms of the concepts they instantiate.

Table 3.6 – LOOM query expressions

Expression	Returns
(same-as v_1 v_2)	true if v_1 and v_2 are equivalent
(subset v_1 v_2)	true if v_1 & v_2 are sets & v_1 is a subset of v_2
(about I C)	true if I is an instance of C
(about I (R v))	true if I:R is filled with v
(about I (filled-by R v_1...v_n)	true if I:R is filled by all of v_1...v_n
(about I (at-least/most n R))	true if I:R has at least/most n fillers
(about I (exactly n R))	true if I:R has exactly n fillers
(about I (all/some R C))	true if all/some fillers of I:R are instances of C
(about I (the R C))	true if I:R is filled by exactly one of C
(predcall P v)	true if (P v) $\neq$ nil
(and Q_1...Q_n)	true if all of Q_1...Q_n are true
(or Q_1...Q_n)	true if any of Q_1...Q_n are true
(not Q)	true if Q is not true
(fail Q)	true if Q is not provably true
(for-some (?x_1...?x_n) Q)	true if $\exists$ a binding for ?x_1...?x_n such that Q is true
(for-all (?x_1...?x_n) (Q_1 Q_2))	true if all ?x_1...?x_n that satisfy Q_1 also satisfy Q_2
(collect ?x Q)	returns v_1...vn for which Q(?x) is satisfied
v is a LISP value which can be a LOOM concept, relation or instance; ?x is a variable which can be bound to a value; Q is a query expression; P is a LISP predicate.	

3.4 GRAIL overview

Unlike LOOM, GRAIL is not designed to be directly accessible to applications programmers; GRAIL is encapsulated within the TeS which provides a high level application interface and a sophisticated graphical environment for the development of terminological models. The GRAIL language has restricted expressiveness in terms of concept and relation forming operators but provides powerful additional features in the form of sanctioning, ***necessary*** statements and refinement.

The current version of GRAIL is written in Smalltalk. C++ implementations of the classifier have been developed for the IBM-PC and a C++ implementation for parallel platforms is under development [GGJ94].

3.5 GRAIL semantics

The current version of GRAIL has no assertional component and consists entirely of a term classifier (T-box). GRAIL's simple syntax makes it necessary to use combinations of statements to define concepts and relations incrementally. There is for example no explicit conjunction operator but a semantically equivalent result can be achieved by using ***addSuper*** statements to assert additional subsumers. Table 3.7 summarises the mapping between GRAIL statements and their equivalent concrete forms.

Table 3.7 – GRAIL compound statements and equivalencies	
GRAIL Statements	Equivalent Concrete Form
TopThing ***which*** *R C* – where R is many valued	(some R C)
TopThing ***which*** *R C* – where R is single valued	(in A C)
(C_1 ***newSub*** *CN)* ***addSuper*** *[C_2...C_n]*	(defprimconcept CN (and C_1...C_n))
C_1 ***whichG*** *<C_2...C_n>* – where $(C_1 \wedge C_2)$...$(C_1 \wedge C_n)$ are grammatically sanctioned	(and C_1...C_n)) – where C_2...C_n are role restriction concepts of the form (∃R:C) or (A:C)
C_1 ***which*** *<C_2...C_n>* – where $(C_1 \wedge C_2)$...$(C_1 \wedge C_n)$ are sensibly sanctioned	(and C_1...C_n) – where C_2...C_n are role restriction concepts of the form (∃R:C) or (A:C)
(R_1 ***newAttribute*** *RN)* ***addSuper*** *[R_2...R_n]*	(and R_1...R_n)

Care must be taken when using statements which add to definitions as this can introduce inconsistencies which are not detected by the current classifier; concepts and relations should be fully defined before being used in further definitions. Section 6.4.3 describes this problem in more detail.

3.5.1 Term classifier

Concepts are defined using a combination of ***newSub***, ***addSub***, ***addSuper***, ***which*** and ***whichG*** statements – table 3.8 summarises the available concept forming operators while table 3.10 summarises the available terminological axioms. The form of the ***which*** and ***whichG*** statements restricts non-primitive concepts to the special case where a single primitive base is conjoined with one or more role restriction concepts (see table 3.7).

Table 3.8 – GRAIL concept forming operators		
Concrete Form	Abstract Form	Descriptive Semantics
top	$\top$	Δ – every I is an instance of top
bottom	$\perp$	∅ – bottom has no instances[19]
(and C_1...C_n)	$C_1 \wedge ... \wedge C_n$	I_c is an instance of all of C_1...C_n
(some R C)	∃R:C	some filler of I_c:R is an instance of C
(in A C)	A:C	the filler of I_c:A is an instance of C
I_c is an instance of the concept formed by the described operator.		

GRAIL has an unusual syntax for role cardinality restrictions: the cardinality of a role, which can only be specified as single or multiple, is fixed by the relation definition. All roles formed from a given relation therefore have the same cardinality restriction.

Role restrictions can be introduced in one of two ways in GRAIL: either by the ***which*** and ***whichG*** operators or by ***necessary*** statements. All role restrictions are of the range restriction types (∃R:C) and (A:C), referred to in GRAIL as criteria. The form (∃R:C) describes criteria

[19] The current implementation of GRAIL does not maintain bottom in the sense of a concept which is subsumed by all other concepts but it still theoretically exists.

where R is a many valued relation while (A:C) describes criteria where R is a single valued relation.

Table 3.9 – GRAIL relation forming operators		
Concrete Form	Abstract Form	Descriptive Semantics
(and $R_1...R_n$)	$R_1 \wedge ... \wedge R_n$	(I_1,I_2) is an instance of all of $R_1...R_n$
(trans R)	R^+	$(I_1,I_2) \in R \wedge (I_2,I_3) \in R \Rightarrow (I_1,I_3) \in R$
(trans-across R R1)	R^{+R1} *– see note*[20]	$(I_1,I_2) \in R \wedge (I_2,I_3) \in R1 \Rightarrow (I_1,I_3) \in R$
(I_1,I_2) is an instance of a binary relation formed by the described operator.		

Relations are defined using the ***newAttribute*** statement which can be combined with ***addSub*** and ***addSuper*** statements to form conjunctions; all GRAIL relations are primitive (see table 3.10). Relation definitions can be extended by adding transitivity and refinement characteristics. The ***newAttribute*** statement defines a relation–inverse pair; inverse relations are used by sanctioning and ***necessary*** statements, which are bi-directional, and by the refinement/transitivity mechanism. Table 3.9 summarises the available relation forming operators.

Table 3.10 – GRAIL terminological axioms		
Concrete Form	Abstract Form	Semantics
(defconcept CN C)	CN = C	$I \in CN \Leftrightarrow I \in C$
(defprimconcept CN C)	$CN \subseteq C$	$I \in CN \Rightarrow I \in C$
(defprimrelation RN R)	$RN \subseteq R$	$(I_1,I_2) \in RN \Rightarrow (I_1,I_2) \in R$

3.5.2 *Necessary* statements

The process of classification has two distinct phases. A concept is located in the hierarchy by first finding those more general concepts which are 'above' it – its subsumers – and then finding those more specialised concepts which are 'below' it – its subsumees. Unlike LOOM implications GRAIL ***necessary*** statements are visible to the classifier – assertional knowledge is used during upwards classification by treating asserted characteristics as part of a concept's description when evaluating its subsumers. However as assertions represent characteristics which are not essential for the recognition of members of a concept they are not considered when evaluating subsumees in the downwards classification phase.

GRAIL provides three statements for adding asserted characteristics to concepts: ***necessarily***, ***topicNecessarily*** and ***valueNecessarily***. All three have identical syntax to the ***which*** statement but are not concept forming; instead they attach strict implications to existing concepts. The statement:

C_1 ***topicNecessarily*** *R* C_2

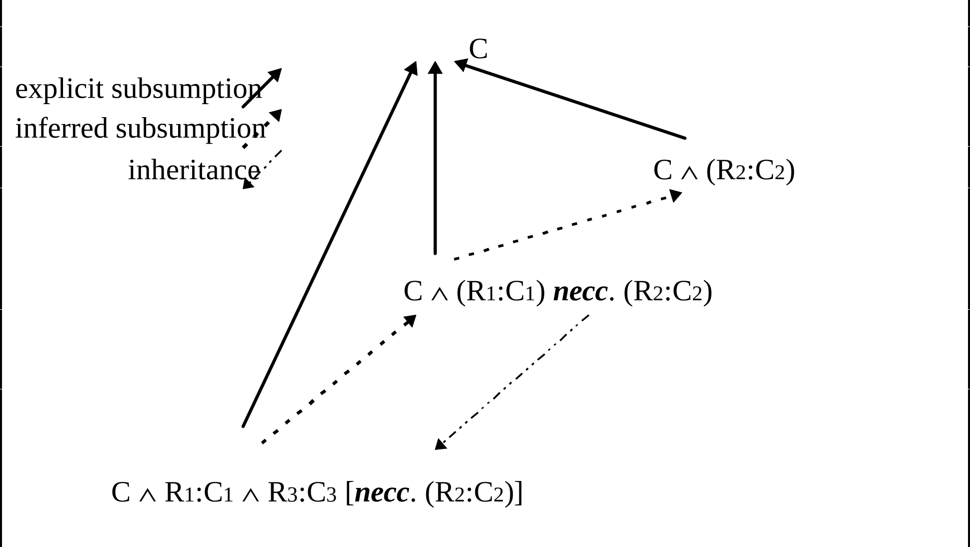
[20] The R^+ notation for transitive relations is extended so that R^{+R1} describes the relation R which is transitive across the relation R1.

allows the classifier to infer that all instances of the concept C_1 are also instances of the concept $(R:C_2)$[21] ; a similar ***valueNecessarily*** statement would allow the classifier to infer that all instances of the concept C_2 are also instances of the concept $(R^{-1}:C_1)$; a ***necessarily*** statement allows both inferences to be made and is equivalent to a combination of ***topicNecessarily*** and ***valueNecessarily*** statements:

$$C_1 \textbf{ valueNecessarily } R\ C_2 \equiv C_2 \textbf{ topicNecessarily } R^{-1}\ C_1$$

$$\begin{array}{c} C1 \textbf{ topicNecessarily } R\ C2 \\ + \\ C1 \textbf{ valueNecessarily } R\ C2 \end{array} \equiv C1 \textbf{ necessarily } R\ C2$$

Figure 3.1 – *Necessary* statements



If concept $C \wedge (R_1:C_1)$ has an attached ***necessary*** statement $(R_2:C_2)$ it will be subsumed by $C \wedge (R_2:C_2)$ but will still subsume $C \wedge (R_1:C_1) \wedge (R_3:C_3)$. The ***necessary*** criterion $(R_2:C_2)$ would then be inherited by $C \wedge (R_1:C_1) \wedge (R_3:C_3)$.

3.5.3 Sanctioning

GRAIL provides a sanctioning mechanism which constrains both the formation of new non-primitive concepts, using the ***which*** and ***whichG*** statements, and the addition of asserted characteristics using ***necessary*** statements.

Before a base–criterion conjunction $C_1 \wedge (R:C_2)$ can be used in a ***which***, ***whichG*** or ***necessary*** statement it must be sanctioned at the appropriate level. Sanctioning is hierarchical and has 3 levels: 'conceivable', 'grammatical' and 'sensible'. A conjunction is sanctioned at the conceivable level if the base concept C_1, the relation R and the range restriction concept C_2

[21] The implication is of attribute form $(\exists R:C_2 \wedge 1R)$ for a single valued R and existential form $(\exists R:C_2)$ for a many valued R.

all exist; grammatical and sensible sanctions are explicitly applied to the base concept using sanctioning statements. Grammatical and sensible sanctions must themselves be sanctioned by conceivable and grammatical sanctions respectively. Sanctioning statements cause sanctions to be applied symmetrically: if $C_1 \wedge (R:C_2)$ is sanctioned the same sanction will be applied to $C_2 \wedge (R^{-1}:C_1)$.

Table 3.11 – GRAIL sanctions

Sanction	Example	Semantics
Conceivable	C_1, R and C_2 all exist	$C_1 \wedge (R:C_2)$ or $C_2 \wedge (R^{-1}:C_1)$ may be grammatically sanctioned
Grammatical	*C_1* ***grammatically*** *R C_2*	$C_1 \wedge (R:C_2)$ or $C_2 \wedge (R^{-1}:C_1)$ may be sensibly sanctioned or conjoined to form a new concept using the ***whichG*** statement
Sensible	*C_1* ***sensibly*** *R C_2*	$C_1 \wedge (R:C_2)$ or $C_2 \wedge (R^{-1}:C_1)$ may be conjoined to form a new concept using the ***which*** statement or conjoined assertively using a ***necessary*** statement

Sanctions are treated as assertions which are not terminologically significant: they do not affect the evaluation of the subsumption relation but they are inherited down the subsumption hierarchy. If a conjunction is sanctioned the same sanction applies to all its subsumees. When a conjunction is used in a ***which***, ***whichG*** or ***necessary*** statement the resulting concept(s)[22] must inherit a sanction at the appropriate level in respect of itself or of a subsuming conjunction. Conjunctions used in ***which*** and ***necessary*** statements must be sanctioned at the sensible level; conjunctions used in ***whichG*** statements need only be sanctioned at the grammatical level. The classifier will reject any statement which would result in a concept being inadequately sanctioned.

Sanctioning highlights a fundamental difference in the GRAIL and LOOM design philosophies: GRAIL makes a default closed-world assumption whereas LOOM assumes an open world. In GRAIL all relations are restricted until explicitly relaxed by sanctioning; in LOOM all relations are unrestricted until explicitly tightened by role restrictions. GRAIL starts out with the implicit assumption that for every relation R the concept top has a restriction of the form $(\forall R:\text{bottom})$; these restrictions are relaxed as sanctions are applied. A sensible sanction $(R:C_1)$ applied to a concept has the effect of relaxing the inherited role restriction so that it becomes $(\forall R:C_1)$. Another sensible sanction $(R:C_2)$ would further relax the restriction so that it becomes $(\forall R:(C_1 \vee C_2))$. This makes it impossible to transpose sanctions into role restriction assertions[23] as, in KL-ONE based systems like LOOM, restrictions can only be tightened as the concept hierarchy is descended.

[22] The result may be a new concept, in the case of ***which*** and ***whichG*** statements, an incrementally redefined concept in the case of ***topicNecessarilly*** and ***valueNecessarily*** statements or a pair of incrementally redefined concepts in the case of the symmetrical ***necessarily*** statement.
[23] They would have to be assertions as they are not terminologically significant.

3.5.4 Refinement and transitivity

In GRAIL relations can be made transitive across other user-defined relations using the ***specialisedBy*** operator:

> R_1 ***specialisedBy*** R_2 ≡ (trans-across R_1 R_2)

In the example in figure 2.1 refinement is used to make the "hasLocation" relation transitive across the "isPartOf" relation:

> *hasLocation* ***specialisedBy*** *isPartOf*

This allows (hasLocation:shaftOfFemur) to be recognised by the classifier as a "kind-of" (hasLocation:femur) which in turn allows the classifier to find the subsumption relation:

> fracture ∧ (hasLocation:femur)
>
> *subsumes*
>
> fracture ∧ (hasLocation:shaftOfFemur)

Transitivity has the special case where a relation is transitive across itself – the syntactic form:

> *R* ***transitiveDown***

is provided as a convenience. This is equivalent to:

> *R* ***specialisedBy*** *R*

3.6 Summary

As can be seen from the preceding sections the terminological services provided by the two languages are quite diverse – GRAIL's ***necessary*** statements, sanctioning and refinement have no direct terminological equivalent in LOOM while GRAIL's concept and relation forming operators are only a small subset of those available in LOOM. This diversity is reflected in the difficulties encountered in translating the CORE model from GRAIL into LOOM, as described in chapter 4.

CHAPTER 4 EXPERIMENTAL METHOD

This chapter describes how the GRAIL CORE model was translated into LOOM, how the performance comparison experiments were designed and how they were carried out.

4.1 Related work

Given that all TKRSs are theoretically intractable, performance analyses using real knowledge bases are of interest. Heinsohn *et al.* empirically tested the classifiers of 6 different TKRSs including CLASSIC and LOOM [HKNP94]. Baader *et al.* used the same data to compare the effect of various optimisation techniques on the KRIS classifier, an early version of which had performed poorly in Heinsohn's tests [BHNP92]. A problem common to both these studies is that the "realistic" knowledge bases used for the tests were relatively small (the largest having 435 concepts and 10 relations) and of questionable realism – most were exemplars provided by the developers of the TKRSs being tested. Both studies also used larger randomly generated knowledge bases but the structure of these knowledge bases was very different from the kind of structure anticipated in a TeS application: 80% of the concepts were primitive, there were only 10 different relations and range restriction roles were formed from a random combination of relations and concepts.

This thesis compares the performances of GRAIL and LOOM using the CORE model from the GALEN project. The timing experiments test classifier performance when expanding the CORE model by adding sanctioned specialisations and when querying the model by re-classifying concepts which have already been added. This should represent a realistic pattern of use in a TeS application.

4.2 Translating the CORE model

In order to conduct performance comparison experiments the GRAIL CORE model was directly translated into LOOM. This may not have made for a completely fair comparison – a different modelling methodology would probably have been chosen if building the model from scratch using LOOM – but it still produced useful and interesting results. Translating the model also highlighted the differences between the two systems.

4.2.1 Separation of parsing and translating

The translation process is separated into two operations, parsing/pre-processing and code generation. The parsing/pre-processing phase is performed by a stand-alone program which converts GRAIL statements into an intermediate LISP readable form. A translation module is loaded along with LOOM and generates LOOM code from the parser's output. Separating the two phases allows the overhead of parsing GRAIL syntax to be eliminated from the performance experiments. The parser would also be reusable for the translation of GRAIL into other LISP based systems such as CLASSIC.

4.3 Translating GRAIL into LOOM

This section describes the process of translating GRAIL into LOOM. Emphasis is given to the problems encountered and the solutions adopted when translating GRAIL's unusual features and syntax.

4.3.1 Relations – cardinality and inverses

Mapping GRAIL's primitive relation hierarchy to LOOM's much richer representation would be straightforward except for GRAIL's unusual treatment of role cardinalities and inverses.

LOOM provides no mechanism for specifying cardinality restrictions as part of relation definitions. A *:single-valued* characteristic can be attached to relations but this is effectively an A-box assertion – it causes the A-box to retract previous role filler assertions when a new filler is asserted for a *:single-valued* role (if the global *:clip-roles* characteristic has been specified). To emulate GRAIL's behaviour the translation module notes relation cardinalities (in LISP property lists) and adds appropriate role cardinality restrictions to concept definitions.

Similarly for inverses LOOM's interpretation is assertion based: using the *:inverse* key word in a LOOM relation definition is an assertion that for any tuple (I_1,I_2) which satisfies the relation, (I_2,I_1) will satisfy the specified inverse relation. In GRAIL the definition of an inverse is used by symmetrical operations such as sanctioning and ***necessary*** statements. The translation module notes inverse relations and uses this information to perform symmetrical operations when required.

4.3.2 Primitive Concepts

Both GRAIL and LOOM provide for the definition of primitive concepts with explicitly asserted subsumption relations. However GRAIL's syntax, which requires the incremental definition of concepts with multiple parents, and a problem with LOOM, which restricts the retrieval of concept definitions[24], makes the translation process surprisingly difficult.

The implemented translation of GRAIL's ***addSuper*** and ***addSub*** statements retrieves the currently asserted subsumers and adds the new subsumer to form a new definition. This will only function correctly if the definition of a concept's subsumers is completed before any sanctioning or ***necessary*** statements are applied. This is adequate for translating the CORE model which conforms to approved modelling practice and always asserts any explicit subsumers immediately after a concept's definition.

[24] The version of LOOM used did not allow the range restriction concept C to be retrieved from existentially quantified role restriction concepts of the form ($\exists$R:C). This problem has been fixed in more recent versions of LOOM.

4.3.3 Non-primitive concepts and GRAIL knowledge names

GRAIL ***which*** statements perform a dual query/create function: if a concept matching the definition is located in the model it is simply returned; if not a new concept is created. The which statement does not name concepts but a 'knowledge name' can be added with a subsequent ***name*** statement.

Although the LOOM ***define-concept*** function can be used in a similar manner, another problem with LOOM[25] prevents the direct translation of ***which*** and ***name*** statements into ***define-concept*** and ***rename-concept*** functions. As a result the translation process is much more complex and requires two hash tables to map between GRAIL and LOOM names. The first hash table maps GRAIL 'knowledge names' to LOOM concept names; the second hash maps LOOM names back to GRAIL definitions and 'knowledge names'.

4.3.4 *Necessary* statements

The semantics of GRAIL ***necessary*** statements cannot be emulated by the LOOM T-box as LOOM's classifier ignores all non-definitional characteristics. However the translation module can use the LOOM matcher to emulate ***necessary*** statements in the A-box by instantiating concepts.

When a concept is instantiated all its inherited characteristics, both definitional and assertional, are used by the LOOM matcher to infer the 'types'[26] of the resulting individual. The recognition of new types can cause extra assertional characteristics to be inherited and initiate further iterations of the matching process; iterations continue until no new types are discovered [MB92]. This closely parallels the upwards phase of classification in GRAIL – the types and characteristics of a LOOM concept's instantiation will be the same as the subsumers and characteristics of an equivalent GRAIL concept. The translation module can use instances to answer queries about subsumers and sanctioning with identical results to those obtained from the GRAIL model. Instances can also be used to find the GRAIL-equivalent subsumees of a concept but this is more complex, requiring multiple LOOM queries and additional processing in the translation module.

4.3.5 Sanctioning

The translation module implements sanctioning using LOOM's ability to have arbitrary LISP objects as role fillers. Like sanctions, role fillers do not affect classification – the concept hierarchy is used only as an inheritance mechanism. Two special relations are defined for GRAIL's 'grammatical' and 'sensible' sanctions. When a sanction is applied the sanctioned criterion, in the form of a two element list, is added to the fillers of the corresponding role.

[25] LOOM crashes if a concept with a system-generated name is renamed after assertions have been added to its definition. The existence of this bug is acknowledged by LOOM's developers but they state that fixing the bug is low priority as "our design philosophy is that users shouldn't reference anything but user-named concepts" [Rus95].

[26] The types of an individual are those concepts which it is recognised as being an instance of.

The translation module tests the sanctioning of a new non-primitive concept by checking that all its defining criteria are subsumed by a filler of the appropriate sanctioning role: the grammatical role in the case of ***whichG*** statements and the sensible role in the case of ***which*** statements. As the process of deleting a classified LOOM concept is rather drastic – all concepts which reference it are recursively deleted – the translation module does not delete un-sanctioned concepts. Instead it prints a message informing the user of the sanctioning violation. The LOOM ***find-subsumers&subsumees*** function could be used to check sanctioning before installing new concepts in the hierarchy but this would result in all new non-primitive concepts being classified twice.

4.3.6 Refinement and transitivity

Refinement and transitivity follow a similar pattern to ***necessary*** statements: GRAIL implements both in the classifier whereas LOOM provides the facility for making suitable assertions which will only affect A-box inferences. In [Bri93] an example definition of R^+, the transitive closure of relation R, is given which uses the ***satisfies*** relation forming operator; a similar result could be achieved using the ***predicate*** or ***function*** operators. Using these operators is equivalent to an extensional definition of the relation in first order logic:

$$R(x,z) \lor \exists y(R(x,y) \land R^+(y,z)) \rightarrow R^+(x,z)$$

An A-box tuple (x,z) satisfies the transitive closure relation R^+ if (x,z) satisfies R or if there exists some individual y such that (x,y) satisfies R and (y,z) satisfies R^+. Refined relations can be similarly defined. For example the relation R^{+R1}, which is equivalent to the GRAIL statement *R* ***specialisedBy*** *R_1*, can be defined as:

$$R(x,z) \lor \exists y(R(x,y) \land R_1(y,z)) \rightarrow R^{+R1}(x,z)$$

Using ***satisfies***, ***predicate*** or ***function*** operators to implement transitive and refined relations in the A-box necessitates instantiating roles as well as concepts. The translation module achieves this using ***implies*** and ***default*** statements to assert role fillers when a concept's definition includes ***which***, ***whichG*** or ***necessary*** statements.

4.3.7 A concept only model

The implementation of ***necessary*** statements, transitivity and refinement described above – using complex and opaque LISP functions to answer classification based queries – is very clumsy and is clearly using LOOM in a way contrary to its design philosophy. To give a more balanced performance comparison two extra LOOM models are used which progressively eliminate these features from the translation module. LOOM model 'A' is the full implementation as described above; LOOM model 'B' does not include transitivity and refinement; LOOM model 'C' is a concept only model and does not support transitivity and refinement or ***necessary*** statements. The reduced functionality of the 'B' and 'C' models results in LOOM missing some subsumption relations which are found by GRAIL.

4.4 Quantitative experiments using the models.

Simple measurements of the times taken to compile the various models are of limited significance in comparing the performances of the two systems. The efficiency of the underlying host systems – Smalltalk in the case of GRAIL and LISP/CLOS in the case of LOOM – can vary widely. Factors such as compilation strategy and garbage collection can also have a significant effect on performance and cannot be controlled consistently across both systems. The experiments are therefore designed to test the scaleability of performance by measuring the rate of change of classification speed with increasing model size.

4.4.1 Compiling the models

The performance scaleability tests use pre-compiled versions of the CORE model. The compilation times for all four models, the GRAIL model plus the three LOOM versions of the model, were measured, taking an average over three compilations. Core images of the successfully compiled models were then saved for use in future tests.

4.4.2 Creating the test files

The performance scaleability experiments require the size of the CORE model to be increased by adding sanctioned specialisations. A LISP function in the translation module was used to automatically generate this data. It does this by examining every concept in the LOOM model and searching the list of sensibly sanctioned criteria for one which could be used to create a new concept. Before searching, the list is randomised using the LISP *random* function; this avoids any bias towards the use of criteria which are applied at more general levels within the model and which might be expected to occur near the beginning of sanction lists. If an appropriate criterion can be found, a GRAIL statement which defines the new concept is written to an output file. Using this function with the LOOM 'B' model yielded a test file containing 1,859 new concept definitions.

To counter any effects due to the ordering of the generated concepts, and to check the consistency of the timings, the experiments use 5 randomised copies of the test file; the LISP *random* function was again used in the randomisation process. Each of the test files is divided into 10 groups for timing purposes – 9 groups of 186 concepts and one group of 185 concepts.

4.4.3 Classifying and installing new concepts

The time taken to classify and install each of the 10 groups of concepts is measured and the measurements are repeated for each of the 5 randomised test files. Installing the 1,859 new concepts increases the size of the CORE model from 2,128 to 3,987 concepts although it does not increase the amount of knowledge represented – the existence of these concepts has been inferred from existing sanctioning knowledge.

Of the initial 2,128 concepts in the CORE model, only 483 (23%) are non-primitive. Installing the new concepts, all of which are non-primitive, changes this proportion to 2,342 non-primitive concepts out of a total of 3,987 (59%).

4.4.4 Querying the models

As well as classifying and installing new concepts a TeS application would be expected to spend a considerable amount of its time answering queries by classifying concept descriptions and discovering identical concepts already installed in the model. In fact it is to be hoped that as the model grows, an increasing proportion of classifications would result in the discovery of an installed concept – if this is not the case the size of the model could be expected to grow out of control.

Re-classifying installed concepts ought to be faster than adding new ones as it is only necessary to perform the first phase of classification – locating the concept in the hierarchy by finding its subsumers. If this process discovers an identical concept already installed in the hierarchy the second phase of classification – locating the concepts subsumees – can be dispensed with.

Due to time constraints the re-classification of installed concepts is only measured for one of the five test files; the test file which produced the most nearly average results for the classification tests, file number 3, is used in this experiment. The experiment measures the time taken to classify and install and then to re-classify each of the 10 groups of concepts in the test file.

A minor complication caused by this method is that the average size of the model during the query phase is larger, because the whole group has already been classified. In order to avoid this it would be necessary to classify and re-classify each concept individually. While feasible for LOOM this would not be possible for GRAIL due to Smalltalk's lack of support for precise timing measurements.

4.4.5 Measuring performance

Performance is measured in terms of CPU time used. As Smalltalk does not provide any built-in means of measuring CPU time the GRAIL timings were performed by running the UNIX 'top' utility and noting the CPU time used before and after each section of the test. This method gives a maximum precision of 1 second – adequate for the purposes of this experiment. Timings for the LOOM models were performed using the LISP *time* function to run each section of the test. This returns the CPU time used in units of 10 milli-seconds.

LOOM uses lazy evaluation for A-box inferences and truth maintenance – they are not performed until required by an A-box query or until forced by a special form of the assertion macro provided for this purpose. LOOM refers to this evaluation process as 'sealing the network'. All timed groups of LOOM code conclude with a call to the macro which causes the network to be sealed. For versions of the LOOM model which instantiate concepts separate timings are provided for the building and sealing phases.

The translation process is assumed to be insignificant in the overall timings of the LOOM models. This assumption was tested by running the translation module with a set of dummy LOOM functions; timings of the translation-only process showed that the overhead represents less than 0.5% of the measured times.

4.4.6 System specification

All tests were run on a Sun SPARCstation 20/61 equipped with a 60MHz superSPARC processor, a 1Mbyte off-chip cache and 128Mbytes of RAM. The operating system was SunOS Release 4.1.3_U1.

The LOOM tests used LOOM version 2.1 patch level 144 compiled using Harlequin LispWorks version 3.2.0. LOOM was compiled after setting optimisations as follows:

(proclaim '(optimize (speed 3) (safety 1) (compilation-speed 0)))

The GRAIL tests used TeS version 1.4b running under VisualWorks, Release 1.0.

CORE model version 1.4.5j was used in all the tests.

CHAPTER 5 EXPERIMENTAL RESULTS

This chapter presents and analyses the results of the performance comparison experiments.

5.1 Compiling the CORE model

Although the primary purpose of the experiments is to compare performance degradation when the size of the CORE model is increased, the compilation times for the different versions of the model are of interest and are presented in table 5.1.

Table 5.1 – CORE model compilation times

model	compilation time (CPU seconds)			
version	load	seal	total	per concept
GRAIL	5,143	–	5,143	2.42
LOOM ‘A’	>216,000	–	–	–
LOOM ‘B’	5,296	43,208	48,504	22.79
LOOM ‘C’	1,535	48	1,583	0.74

It proved impossible to compile version ‘A’ of the LOOM model. After sixty hours of CPU time only a fraction of the model had been compiled and the rate of compilation – monitored by observing LOOM’s trace output – was continually degrading. Comparison of the compilation times for versions ‘A’ and ‘B’ of the LOOM model make it clear that it is the inclusion of A-box transitivity and refinement which is causing the unacceptably poor performance.

This problem was brought to the attention of the LOOM development team at ISI. After investigation they stated that the poor performance is caused by inefficiencies in the LOOM code which handles relation definitions using ***satisfies***, ***function*** or ***predicate*** operators. Work is underway to fix this problem but as the changes required are considerable they will only be incorporated into a future version of LOOM 3.0 [Mac95]. In the meantime further experiments using the LOOM ‘A’ model were abandoned.

5.2 Increasing the size of the CORE model

The three pre-compiled versions of the CORE model (GRAIL, LOOM ‘B’ and LOOM ‘C’) were used to conduct the performance scaleability experiments. Measurements were taken when adding the 5 different random orderings of the new concepts created from the LOOM ‘B’ model.

5.2.1 The GRAIL model

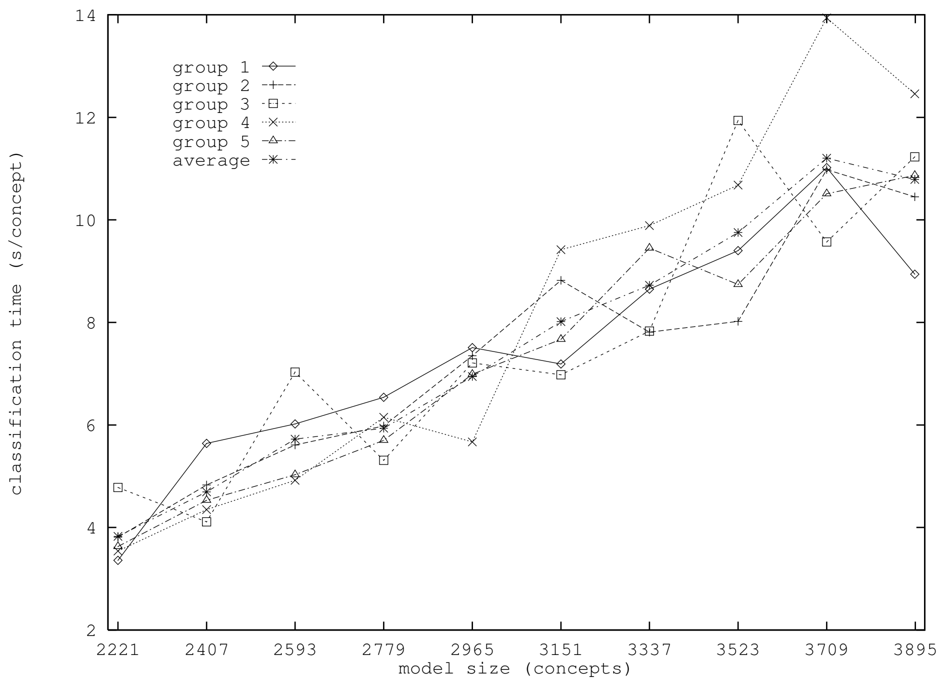
Table 5.2 lists classification times for the GRAIL model. All the concepts created from the LOOM 'B' model were successfully classified by GRAIL.

Table 5.2 – GRAIL classification times

avg. model	data set timings (CPU seconds)					
size	1	2	3	4	5	average
2,221	625	709	889	659	676	711.6
2,407	1,049	899	765	810	842	873.0
2,593	1,120	1,044	1,308	915	936	1,064.6
2,779	1,217	1,113	988	1,143	1,061	1,104.4
2,965	1,396	1,368	1,341	1,054	1,300	1,291.8
3,151	1,337	1,640	1,298	1,753	1,427	1,491.0
3,337	1,609	1,453	1,456	1,839	1,757	1,622.8
3,523	1,748	1,491	2,221	1,986	1,625	1,814.2
3,709	2,050	2,043	1,780	2,593	1,954	2,084.0
3,894	1,654	1,933	2,077	2,305	2,011	1,996.0
total	13,805	13,693	14,123	15,057	13,589	14,053.4

The graph in figure 5.1 plots the average time taken to classify each concept against the average model size.

Figure 5.1 – GRAIL classification time –v– model size



The increase in classification time seems to be fairly linear and, taking the average of the five orderings, is approximately 1.6 times greater than the increase in model size.

5.2.2 LOOM model 'B'

LOOM model 'B' is the version of the LOOM model which uses A-box inferences to imitate the behaviour of GRAIL ***necessary*** statements. It proved impossible to run the timing experiments using this model as LOOM consistently crashed while classifying the new concepts. The debug output from LOOM stated that an internal integrity check had failed.

This problem has been reported to the LOOM development team but they have yet to find a solution. Partial timings obtained for 2 of the data sets are listed in table 5.3.

Table 5.3 – LOOM model 'B' classification times

average model size	data set timings (CPU seconds)					
	1			2		
	load	seal	total	load	seal	total
2,221	13,971	9,678	23,649	14,012	11,052	25,064
2,407	16,302	11,772	28,074	17,698	–*	–
2,593	19,050	–*	–	–	–	–

Entries marked * indicate that LOOM crashed during this phase of the test.

5.2.3 LOOM model 'C'

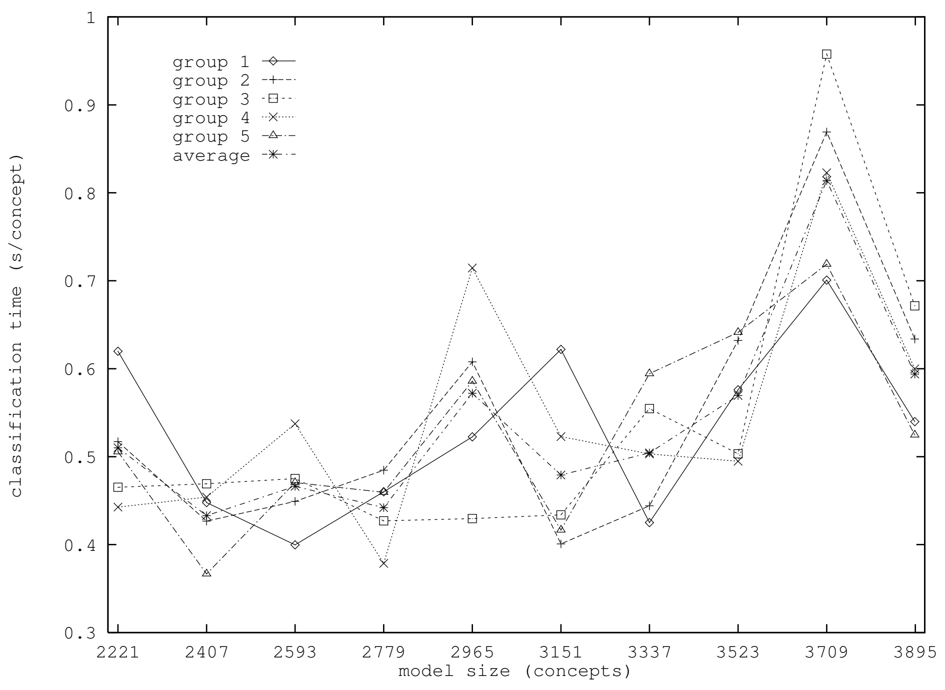
LOOM model 'C' is the concept only model – no use is made of the LOOM A-box. Classification times are listed in table 5.4; the times given are those to load the concepts into LOOM – the time taken to seal the network was 0 in all cases as no instances were created. Ninety-two of the new concepts proved to be inadequately sanctioned when classified by the LOOM 'C' model. This is due to the fact that the LOOM 'C' model misses subsumption relations which are found by the 'B' model using its emulation of GRAIL's ***necessary*** statements. Sanctions which would be inherited across these subsumption relations are therefore not present in the 'C' model. However the test timings should be unaffected as the translation module classifies inadequately sanctioned concepts in the normal manner, only causing a notification to be printed on the terminal.

Table 5.4 – LOOM model 'C' classification times

avg. model	data set timings (CPU seconds)					
size	1	2	3	4	5	avg.
2,221	115.31	96.18	86.53	82.34	94.02	93.42
2,407	83.30	79.37	87.25	84.44	68.21	79.84
2,593	74.35	83.54	88.36	99.93	87.52	85.85
2,779	85.56	90.14	79.44	70.43	85.46	81.16
2,965	97.18	113.06	79.91	132.89	108.98	105.34
3,151	115.70	74.55	80.68	97.24	77.48	87.99
3,337	79.07	82.61	103.18	93.58	110.54	92.84
3,523	107.15	117.57	93.59	92.03	119.30	104.42
3,709	130.34	161.67	178.14	153.05	133.69	149.73
3,894	99.86	117.26	124.25	110.97	97.05	108.14
total	987.82	1,015.95	1,001.33	1,016.90	982.25	988.73

The graph in figure 5.2 plots the average time taken to classify each concept against the average model size. The data does not show a linear increase in classification times but displays marked peaks when the model size reaches approximately 3,000 and 3,700 concepts. There may also be a peak at a model size of approximately 2,200 concepts but this is not so clear as it coincides with the first data point.

Figure 5.2 – LOOM model 'C' classification time –v– model size



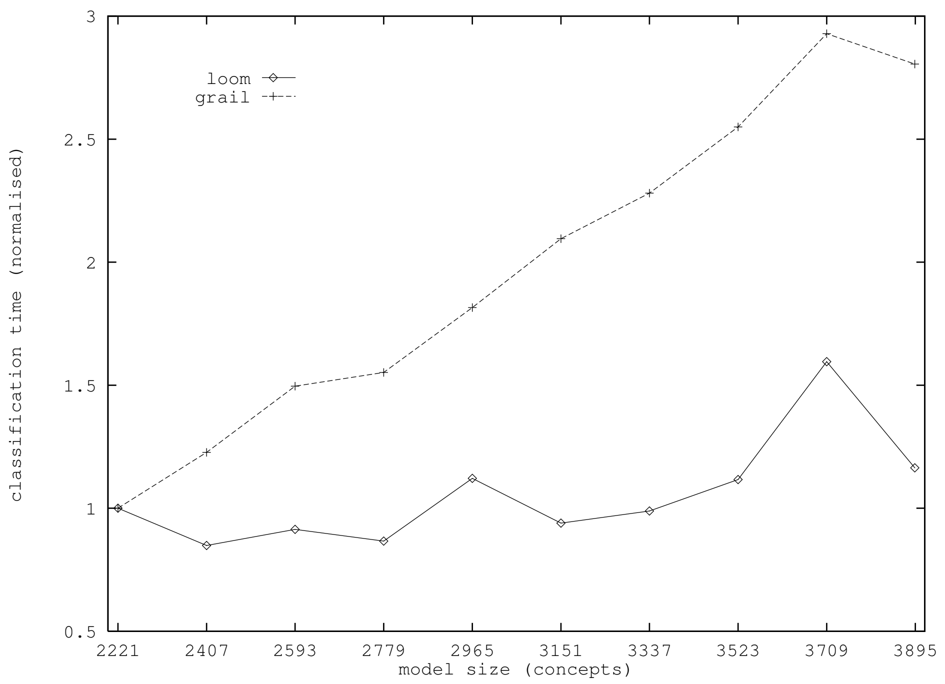
It seems likely that the peaking effect is caused by some "housekeeping" operation being performed by LOOM or LISP rather than being characteristic of the performance of LOOM's classification algorithm. The obvious candidate was LISP garbage collection but initiating a full garbage collection sweep between the classification of each set of concepts did not significantly change the results. It has been suggested by LOOM's developers that the peaking could be caused by LOOM changing data structures from association lists to hash tables although this would normally be expected only when using the A-box.

The peaking effect makes it difficult to give a precise rate at which classification time is increasing relative to model size. However comparing the average time taken to classify each concept in the first 5 groups (0.48s) with that of the second 5 groups (0.58s) gives an increase in classification time approximately 0.9 times the increase in average model size.

5.2.4 Comparing GRAIL with LOOM

To compare GRAIL with LOOM the graph in figure 5.3 plots the average classification times for the GRAIL and LOOM 'C' models scaled so that the time taken to classify the first group of new concepts is 1. This clearly shows the different rates of increasing classification time for the two models.

Figure 5.3 – Normalised classification time –v– model size



5.3 Querying the models

After the failure of the earlier experiments with the LOOM 'A' and 'B' models only the GRAIL and LOOM 'C' models were used in this test. The CORE model was again expanded by adding the 3rd randomly ordered test file, but this time each of the 10 groups was classified twice. A pair of timings for classification and querying/re-classification is given for each group.

5.3.1 The GRAIL model

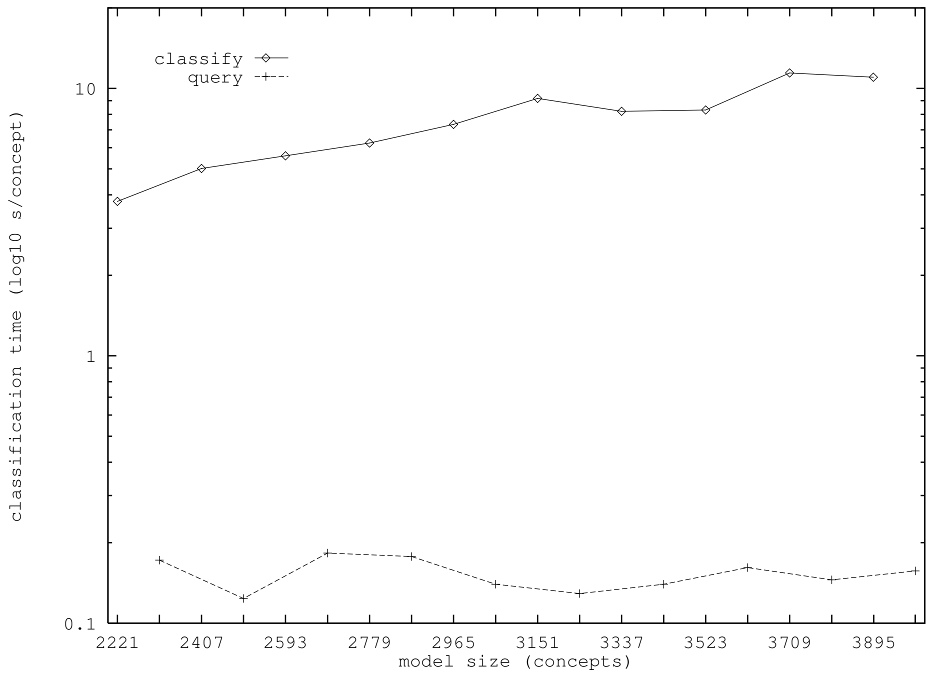
Classification and query times for the GRAIL model are listed in table 5.5.

Table 5.5 – GRAIL model classification and query times

classify (CPU seconds)			query (CPU seconds)		
model size	group	per concept	model size	group	per concept
2,221	704	3.78	2,314	32	0.17
2,407	934	5.02	2,500	23	0.12
2,593	1042	5.60	2,686	34	0.18
2,779	1162	6.25	2,872	33	0.18
2,965	1366	7.34	3,058	26	0.14
3,151	1708	9.18	3,244	24	0.13
3,337	1527	8.21	3,430	26	0.14
3,523	1545	8.31	3,616	30	0.16
3,709	2124	11.42	3,802	27	0.15
3,894	2036	11.01	3,987	29	0.16
total	14148	7.61	total	284	0.15

The graph in figure 5.4 plots the average time taken to classify and query each concept against the average model size.

Figure 5.4 – GRAIL classification and query times –v– model size



The results show a remarkable contrast between classification and querying with the total times varying by a factor of 50:1. The difference is so great that it was necessary to use a logarithmic scale for time in the graph in figure 5.4. The other notable point about this data is that the query times show very little variation with increasing model size.

The algorithm used by the GRAIL classifier gives a possible explanation for the large difference between the classification and query times [Bec95]. GRAIL concepts have at most one primitive base and the classifier optimises its search for subsumers by starting at the primitive base instead of at bottom. After finding all subsumers in this reduced search the classifier would normally have to search the rest of the model for any additional subsumers. However if the classifier checks for equality at this point and finds a concept which matches the definition being classified it can avoid searching the rest of the model as well as avoiding the subsumee phase of classification.

5.3.2 The LOOM ‘C’ model

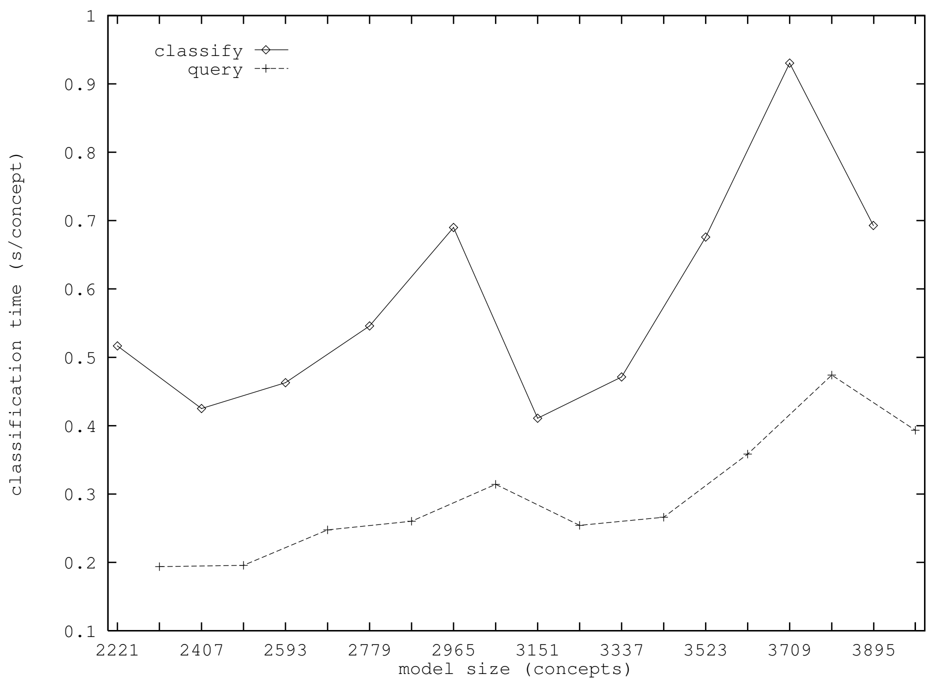
Classification and query times for the LOOM ‘C’ model are listed in table 5.6.

Table 5.6 – LOOM 'C' model classification and query times

classify (CPU seconds)			query (CPU seconds)		
model size	group	per concept	model size	group	per concept
2,221	96.10	0.5167	2,314	36.03	0.1937
2,407	79.07	0.4251	2,500	36.40	0.1957
2,593	86.09	0.4628	2,686	46.03	0.2475
2,779	101.54	0.5459	2,872	48.40	0.2602
2,965	128.34	0.6900	3,058	58.45	0.3142
3,151	76.43	0.4109	3,244	47.26	0.2541
3,337	87.67	0.4713	3,430	49.49	0.2661
3,523	125.71	0.6759	3,616	66.67	0.3584
3,709	173.05	0.9304	3,802	88.15	0.4739
3,894	128.19	0.6929	3,987	72.78	0.3934
total	1,082.19	0.5821	total	549.66	0.2958

The graph in figure 5.5 plots the average time taken to classify and re-classify each concept against the average model size.

Figure 5.5 – LOOM classification and query times –v– model size



While classification in the LOOM model is slower than querying, the ratio of approximately 2:1 is far less than that observed with the GRAIL model. A full explanation of this effect is impossible without more information about LOOM's classification algorithm; one possible explanation is that LOOM takes about the same time for subsumer and subsumee classification and, as querying saves having to perform the subsumee phase, classification times are halved.

5.4 Summary

CPU time used is a very crude measure of performance; when studying optimisations of the KRIS classifier subsumption tests were assumed to be by far the most expensive operation and the number of subsumption tests made was used as a measure of performance [BHNP92]. However this assumption may not be completely valid for GRAIL, which has been designed to make subsumption testing relatively easy. More detailed profiling, showing where CPU time was being used, might enable the observed differences in performance to be explained – and would indicate where efforts at optimisation could best be concentrated – but without this kind of information it is difficult to draw any firm conclusions.

In spite of these limitations the results do show some interesting features: there is no obvious sign of an exponential increase in classification time with increasing model size; GRAIL showed a dramatically improved performance when querying the knowledge base; and LOOM performed well in spite of its potential intractability. These features are discussed in more detail in section 7.3.

CHAPTER 6 QUALITATIVE ANALYSIS

This chapter compares GRAIL and LOOM based on the experience gained in carrying out the translation and experimental work. The features of the two systems are examined with particular reference to their applicability to the representation of medical terminology. Support for knowledge acquisition, ease of use and robustness are also considered.

6.1 Terminological expressiveness

GRAIL's terminological language has deliberately limited expressive power. The range of concept and relation forming operators provided is extremely restricted – it is far from satisfying the minimum criteria set out in [PSS93] for a core knowledge base. Primitive concepts can be formed from arbitrary conjunctions but the only operation available for the formation of non-primitive concepts is the addition of criteria using ***which*** and ***whichG*** statements. GRAIL's range of relation forming operators is even more restricted: relations can only be formed from primitive conjunctions. This means that the role hierarchy has only 'operational semantics' [WS92]: the position of a role in the hierarchy is explicitly determined and is independent of its definition. The ability to build a role hierarchy with 'criterial semantics' similar to those of the concept hierarchy is central to KL-ONE influenced knowledge representation systems [ibid.].

In contrast LOOM has a highly expressive terminological language with a wide range of concept and role forming operators. Constructs which are available in LOOM but not in GRAIL include:

- Arbitrary non-primitive conjunctions
- Disjunctions
- Negation
- A full range of cardinality restrictions
- Role chains
- Role value maps
- Ordered sets

LOOM's expressiveness facilitates the representation of some concepts which are difficult or impossible to represent in GRAIL. This is exemplified by Essential Hypertension, a problem case from [RNGM93].

6.1.1 Essential Hypertension

Essential Hypertension is described in [RNGM93] as 'hypertension which, after investigation, does not have a known cause'. Secondary Hypertension is hypertension which does have a known cause (for example a renal condition). This is difficult to represent in

GRAIL as the concept “essential hypertension” is defined by the exclusion of causes – equivalent to a zero cardinality restriction on the “cause” relation – and GRAIL can only represent cardinalities of one and many. In this example it is important not to confuse exclusion with lack of specification as this could result in the classification of all “hypertensions” as “essential hypertension” unless a cause is stated. LOOM supports arbitrary role cardinality restrictions, allowing the two forms of hypertension to be defined as:

secondary-hypertension = hypertension $\wedge$ ($\geq$1 cause)

essential-hypertension = hypertension $\wedge$ (0 cause)

A “hypertension” would only be classified as “essential hypertension” if the exclusion of “cause” was explicitly stated or could be inferred; this is not the case for descriptions of “hypertension” which merely fail to specify a “cause”. “Secondary hypertension” would be recognised whenever a “hypertension” was defined as having some “cause”. LOOM’s ability to define concepts with arbitrary role cardinality restrictions solves this representation problem in a clear and principled manner.

It is possible to represent “essential hypertension” in GRAIL – the solution suggested in [RNGM93] is to define it as a primitive concept and to cancel the sanctioning of all criteria which include the “cause” relation. However this has a number of disadvantages: the meaning of “essential hypertension” is not clear from its definition – it is simply introduced into the model as a primitive; “essential hypertension” can never be recognised from a concept description unless it is explicitly stated; allowing cancellation is an exception to the general principle of sanctioning and is “still not completely understood” [RNGM93]. Future versions of GRAIL may be able to improve on this solution by adding support for a limited form of negation.

6.2 GRAIL’s special features

Although GRAIL has restricted concept and relation forming operators it does provide a number of powerful features which enhance classification based reasoning. These features are deeply embedded in the design of the GRAIL classifier and proved difficult or impossible to satisfactorily implement using LOOM.

6.2.1 *Necessary* Statements

Necessary statements are difficult to translate into LOOM as they embody a fundamental difference in the behaviour of the two classifiers: the LOOM classifier will only consider definitional characteristics whereas the GRAIL classifier is also able to consider non-definitional characteristics when evaluating a concept’s subsumers.

Although it is technically feasible to imitate the behaviour of the GRAIL classifier using LOOM’s A-box and additional LISP code in the translation module this is unsatisfactory in terms of design transparency and also resulted in an unacceptable performance penalty. In fact it proved impossible to add all the test concepts to the LOOM model using the translation module in this mode without causing the LOOM classifier to crash.

6.2.2 Sanctioning

Sanctioning's underlying closed world assumption and the ability to relax the implied range restrictions make it impossible to translate sanctions directly into LOOM. Sanctioning was however relatively easy to implement by treating sanctions as extrinsic information, using the concept hierarchy purely as an inheritance mechanism and writing LISP functions to perform the sanction checking.

Apart from the undesirability of implementing additional features in LISP code, using a sanction checking mechanism which is not tightly integrated with the classifier is bound to result in a performance penalty. A concept's sanctioning can only be checked after sanctions have been inherited from all its subsumers. In GRAIL sanctioning can be checked after the evaluation of a concept's subsumers but before evaluating its subsumees and installing it in the hierarchy. If the concept is appropriately sanctioned the classifier can proceed with classification and installation; if not no change is made to the existing hierarchy and the concept is rejected.

It is impossible to stop half way through LOOM's classification process and check on sanctioning. As deleting concepts from a LOOM model is difficult sanctioning must either be checked before classification or un-sanctioned concepts must be left in the model. Checking sanctioning using the ***find-subsumers&subsumees*** function will result in all concepts being fully classified and sanctioned concepts being classified twice; leaving un-sanctioned concepts in the model will still result in their being fully classified and will cause an unnecessary increase in model size.

6.2.3 Refinement and Transitivity

The LOOM classifier does not support refinement or transitivity. Although it is technically feasible to imitate GRAIL's refinement and transitivity using LOOM's A-box, the resulting performance is so poor as to make this solution unusable in large concept models.

6.3 Modelling Tools

GRAIL provides a complete modelling environment which includes an extensive array of tools designed to facilitate the model building task. These include browsers and editors for working directly with the model or with source files [GALEN94]. LOOM incorporates no facilities of this kind and working with anything other than 'toy' sized models proved extremely difficult.

LOOM's sole concession to interactive usability is the provision of a number of macros designed for the interactive retrieval of information from the model. The interface is however limited to typing macros and parameters into the LISP interpreter and the information retrieved by any one macro is limited. For example to retrieve a concept's definition the user can type:

(***pc*** *concept-name*)

This will cause the LOOM definition of the concept to be printed on the terminal. Output from the LOOM ‘C’ model in response to the command ***(pc femur)*** is shown in figure 6.1 with the *sensible* and *grammatical* role fillers truncated for clarity.

Figure 6.1 – LOOM display for (*pc femur*)

```
(defconcept |Femur|
  :is-primitive |LongBone|
  :implies (:and (:some |hasTopology| |Topology*G48712|)
            (:all |hasTopology| |Topology*G48712|)
            (:at-most 1 |hasTopology|)
            (:some |hasShapeAnalagousTo| |AnatomicalShape*G47812|)
            (:all |hasShapeAnalagousTo| |AnatomicalShape*G47812|)
            (:at-most 1 |hasShapeAnalagousTo|)
            (:filled-by Grammatical (|boundsSpace| |BodySpace|)
                (|hasLayer| |BodyStructure*G52289|) (|hasLayer| |BodySubstance|)
                (|isStructuralComponentOf| |Organism|)
                (|isSolidDivisionOf| |BodyStructure|) .... )
            (:at-least 57 Grammatical)
            (:some |hasIntrinsicAbnormalityStatus| |normal|)
            (:all |hasIntrinsicAbnormalityStatus| |normal|)
            (:at-most 1 |hasIntrinsicAbnormalityStatus|)
            (:some |hasSurfaceVisibility| |internal|)
            (:all |hasSurfaceVisibility| |internal|)
            (:at-most 1 |hasSurfaceVisibility|)
            (:some |hasCountability| |discrete|)
            (:all |hasCountability| |discrete|)
            (:at-most 1 |hasCountability|)
            (:filled-by Sensible (|boundsSpace| |BodySpace|)
                (|isActedOnBy| |BodyProcess*G51050|)
                (|isActedOnSpecificallyBy| |Ischaemia|)
                (|isLocationOf| |DegenerativeLesion|) .... )
            (:at-least 29 Sensible)
            (:some |isStructuralComponentOf| |Thigh|)
            (:filled-by Necessary (|hasTopology| |Topology*G48712|)
                (|hasShapeAnalagousTo| |AnatomicalShape*G47812|)
                (|hasCountability| |discrete|) (|hasSurfaceVisibility| |internal|) .... )
            (:at-least 6 Necessary))
  :characteristics (:closed-world :monotonic)
  :system-characteristics (:closed-world))
```

Given the fact that all non-primitive concepts in the translated CORE model have meaningless system generated names, such as |BodyStructure*G52289|, examining the model via this method is extremely inconvenient. The provision of browsing and editing tools would greatly facilitate the use of LOOM in a large modelling task.

6.4 Problems with the GRAIL classifier

The process of formalising GRAIL's semantics, translating the CORE model into LOOM and conducting the timing experiments brought to light a number of problems with the GRAIL classifier.

6.4.1 Inferred non-primitive conjunctions

GRAIL's closed world assumption in respect of sanctioning, as described in section 3.5.3, is also extended to concept disjunction: GRAIL makes the default assumption that all primitive concepts are disjoint. This assumption is used by the GRAIL classifier when checking for incoherence caused by conflicting cardinality restrictions.

Consider for example a concept of the form:

$$C = C_x \wedge (R{:}C_y) \wedge (R{:}C_z)$$

where the criteria $(R{:}C_y)$ and $(R{:}C_z)$ are both sensibly sanctioned with respect to C_x. If R is a single valued relation the classifier can infer that the range restriction is the conjunction of C_y and C_z:

$$C_x \wedge (R{:}C_y) \wedge (R{:}C_z) \wedge (1R) \Rightarrow C_x \wedge (R{:}(C_y \wedge C_z)) \wedge (1R)$$

In the general case all GRAIL concepts are of the form:

$$B \wedge (R_1{:}C_1) \wedge \ldots \wedge (R_n{:}C_n)$$

where B is a primitive base and $(R_i{:}C_i)$ are criteria. If n=0, there are no criteria and the concept is the primitive B. The conjunction $(C_y \wedge C_z)$ thus becomes:

$$B_y \wedge B_z \wedge (R_{y1}{:}C_{y1}) \wedge \ldots \wedge (R_{yn}{:}C_{yn}) \wedge (R_{z1}{:}C_{z1}) \wedge \ldots \wedge (R_{zn}{:}C_{zn})$$

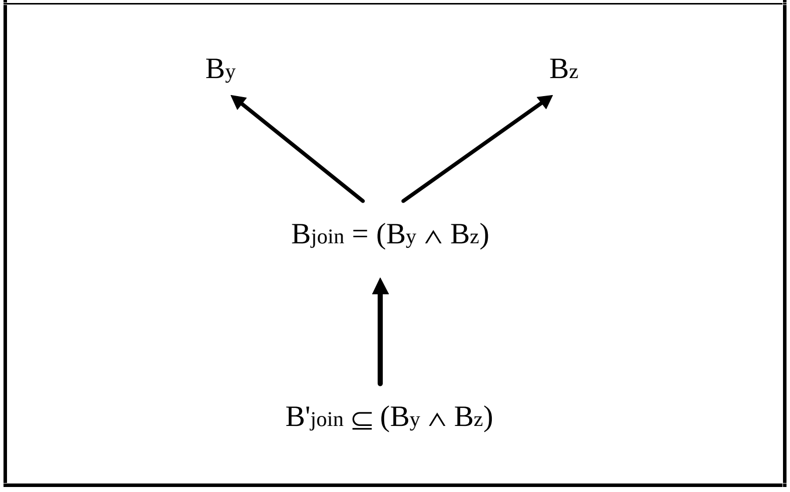
If one of B_y or B_z subsumes the other, or if they are equal, the primitive base reduces to the most specialised of the two. In this case the conjunction is a normal GRAIL concept, which must be sanctioned as all of the criteria were sanctioned on one of B_y or B_z, and the new concept C is accepted by the classifier. If B_y and B_z do not subsume the classifier assumes they are disjoint and rejects C as incoherent.

Unfortunately this assumption is not valid as a primitive conjunction of B_y and B_z may already exist in the hierarchy:

$$(B_y\ \textit{\textbf{newSub}}\ B'_{join})\ \textit{\textbf{addSuper}}\ B_z \Rightarrow B'_{join} \subseteq (B_y \wedge B_z).$$

The conjunction B'_{join} must be primitive as GRAIL's concept forming operators only support the definition of non-primitive concepts in the special case where a single primitive is conjoined with role restriction concepts (criteria). The existence of B'_{join} proves that B_y and B_z are not disjoint. However the range restriction conjunction $(B_y \wedge B_z)$ cannot be inferred to be equal to B'_{join} – a primitive's definition consists of conditions which are necessary but not sufficient for membership; the range restriction is a non-primitive conjunction B_{join} which subsumes B'_{join}. GRAIL provides no mechanism for defining or describing such a concept.

Figure 6.2 – Primitive and non-primitive conjunctions


B_y
B_z
$B_{join} = (B_y \wedge B_z)$
$B'_{join} \subseteq (B_y \wedge B_z)$

In these circumstances the classifier can either classify the concept C as incoherent (as it does now) or allow the inferred existence of a range restriction concept B_{join} which cannot be described using the GRAIL language. Clearly neither of these choices is completely satisfactory and an extension to the language, which would allow sanctioned non-primitive conjunctions to be defined, is currently under consideration.

6.4.2 Cardinality restrictions and the relation hierarchy

GRAIL fails to detect incoherence resulting from an interaction between cardinality restrictions and the relation hierarchy. Single valued relations with multiple subsumees can result in an implied cardinality inconsistency which is not detected by the classifier. For example given 3 single-valued relations R, R_1 and R_2:

if $\quad R_1 \subseteq R;\ R_2 \subseteq R;\ C_1 \parallel C_2$

then $\quad C \wedge (R_1{:}C_1) \wedge (R_2{:}C_2) \Rightarrow \geq 2R \Rightarrow$ *incoherent*.

6.4.3 Inconsistencies resulting from concept and relation redefinitions

GRAIL's simple syntax means that many concepts and relations must be incrementally redefined by adding explicit subsumers, explicit subsumees, ***necessary*** statements, refinement and transitivity. Changing the definition of concepts and relations in this way is potentially dangerous as it allows inconsistencies to be introduced into the concept hierarchy. For example, assuming the existence of the appropriate sanctioning, that R_1 is single valued and that C_x and C_y are disjoint:

*C_1 **topicNecessarily** R_1 C_x.*

*(C_1 **newSub** C_3) **addSuper** C_2.*

C_2 ***topicNecessarily*** R_1 C_y.

will result in the incoherence of C_3 due to its inheriting conflicting role restrictions from the two ***necessary*** statements. However this incoherence goes undetected because it did not exist when C_3 was defined: it results from the subsequent redefinition of C_2.

In order to be certain of maintaining the consistency of the model it is necessary not only to reclassify redefined concepts[27] but also to recursively reclassify all concepts whose definition references a redefined concept or relation. This is not done by the GRAIL classifier on the grounds that it is a potentially expensive operation[28]: instead GRAIL users are advised that concepts and relations should be fully defined before being used in other definitions.

Not initiating an integrity check when concepts and relations are redefined has a number of disadvantages:

- The occasions when an integrity check would be very costly are precisely those when one is most necessary: the redefinition of concepts or relations with large numbers of dependants. If users follow the modelling guidelines and fully define concepts and relations before using them in other definitions the cost of integrity checking should be minimal.
- Delaying integrity checking throws away the advantage of being able to restrict the range of the check to those concepts which might be affected.
- Delaying integrity checking might make it difficult to identify the operation which caused the inconsistency.

LOOM's syntax provides for the complete definition of concepts and relations using a single statement. This means that, unlike GRAIL, redefinitions can be the exception rather than the rule. In the case where an existing concept or relation is redefined LOOM does perform all the necessary reclassification and checking to ensure that the consistency of the model is maintained.

6.4.4 Inheritance of refinement and transitivity

In the current version of the GRAIL classifier refinement and transitivity characteristics are inherited down the relation hierarchy [BS94]. This is not semantically justified. For example the relations "parent", "grandparent" and "great-grandparent" would all be subsumed by the transitive relation "ancestor", but it would be wrong for any of them to inherit the transitivity characteristic.

[27] In most terminological logics it would also be necessary to reclassify a redefined relation but this is not the case in GRAIL as the relation hierarchy does not have criterial semantics - the position of a relation in the hierarchy does not depend on its definition.

[28] A model checking tool is provided, using which it is possible to initiate a consistency check of the entire model.

6.5 Problems with the LOOM classifier

The LOOM classifier has far greater inferential power than the GRAIL classifier: it finds all the inferences missed by GRAIL and can make many more inferences as a result of its much richer syntax. However this increased power does seem to result in an increased fragility.

6.5.1 Fragility of the LOOM classifier

While experimenting with the translation process a number of bugs were discovered which resulted in a total failure of the LOOM system. For example:

```
(defconcept a)
(defconcept b)
(defrelation r)
(implies a (:all r b))
(get-role 'a 'r)
(implies a (:at-least 2 r))
(get-role 'a 'r)
(implies a (:at-most 5 r))
(get-role 'a 'r)
```

caused the system to crash and enter the debugger indicating a CLOS error.

All of the bugs encountered were reported to LOOM's development team and several have now been fixed, including the above example. Judging from this experience and the frequency of bug reports observed in the LOOM users mailing list it is clear that the current version of LOOM is very much a beta-test release.

6.5.2 Performance of the LOOM A-box

The performance of LOOM's A-box was so poor as to render it unusable with a knowledge base as large as the CORE model. Simply creating one instance of each concept caused compilation to slow by a factor of 30 – classification was 5 times slower with the remaining difference being accounted for by the A-box. Attempts to enlarge the CORE model caused LOOM to fail completely.

When instance definitions were expanded to include role fillers classification performance became so poor that it proved impossible to compile the CORE model.

6.6 Maturity, stability and development

It became clear while using the two systems that neither is fully stable and mature. Ongoing research and development in respect of both systems result in regular bug fixes and new releases.

CHAPTER 7 DISCUSSION AND CONCLUSIONS

This chapter summarises what has been learned and the extent to which the aims of the project have been satisfied. It concludes with a discussion of possible future work.

7.1 Recapitulation of aims

The aims of this thesis, as stated in the introduction, were to compare GRAIL and LOOM in order to: achieve a better understanding of GRAIL; test the performance and scaleability of both systems; assess how well the systems satisfied the requirements of the GALEN and PEN&PAD projects; and to enhance GRAIL's future development.

7.2 A better understanding of GRAIL

The formalising of GRAIL's semantics and the comparison with LOOM has resulted in a much clearer understanding of GRAIL's features and how they relate to other KL-ONE languages. The improved understanding of GRAIL's semantics has also highlighted some problems with the GRAIL classifier.

One basic difference which has emerged is that GRAIL is fundamentally oriented towards constraint whereas LOOM is fundamentally oriented towards inference. When presented with a new piece of information, in the form of a concept definition, GRAIL's approach can be characterised as: "given what I know so far, does this new piece of information make sense?". In the same circumstances LOOM's approach can be characterised as: "given what I know so far, how can I make sense of this new piece of information?".

GRAIL's constraint orientated approach is reflected in its closed world assumptions with respect to sanctioning and disjunction. The default assumption of concept disjunction has been shown to be problematical in some circumstances and this will probably result in a redefinition of part of the language.

The relationship between sanctioning and role restrictions has also been clarified. The semantics of sanctioning have been shown to be poorly understood and to be difficult to reconcile with the semantics of a KL-ONE language. Sanctions correspond more closely to conceptual relations which Sowa describes as "constraints on the use of relations in conceptual graphs" [Sow84].

7.3 Performance and scaleability

Given the well known tractability problems associated with TKRSs the results of the performance tests were encouraging. When increasing the size of the knowledge base in a very selective manner, by installing sanctioned specialisations, both GRAIL and LOOM showed a relatively slow increase in classification times, in the order of 1.6n for GRAIL and

0.9n for LOOM. GRAIL also showed a remarkably improved performance when querying the knowledge base – averaging over 50 times faster than classification – with no measurable degradation of performance as the size of the knowledge base increased.

These results are much better than might be theoretically expected, and much better than those obtained by testing classifiers with large randomly generated knowledge bases when it was observed that runtime grew “at least quadratically with the size of the knowledge base” [HKNP94]. LOOM’s better classification performance, in spite of its very expressive terminological language, and the very large discrepancy between classification and query times in GRAIL, indicate that there may be considerable scope for optimisation in the GRAIL classifier.

LOOM’s performance also shows that an expressive and potentially intractable terminological language can give acceptable performance when used in a limited manner – the user is able to “pay only as s/he goes” [HKNP94]. This indicates that one of GRAIL’s design precepts – that tractability requires severely restricted expressiveness – may not be completely valid.

7.4 Representing medical terminology

LOOM’s expressive terminological language could be usefully employed to solve some of the representational problems encountered in medical terminology. However GRAIL’s special features – features which were designed specifically for the representation of medical terminology – proved almost impossible to translate into LOOM: only sanctioning could be represented without using LOOM’s A-box.

Using LOOM’s A-box to implement features not supported in the T-box, such as refinement and ***necessary*** statements, proved impractical due to the its very poor performance. Even if performance was not an issue, the use of opaque external code is antithetical to a TKRS which should have “a well understood declarative semantics” [BHH+91].

7.5 Future work

It is clear from this study that while GRAIL does come closer to satisfying the requirements for a system designed to represent medical terminology, work remains to be done before GRAIL could be considered a complete solution:

- Experience of using GRAIL in the GALEN project has inevitably resulted in the identification of additional modelling requirements, in particular the use of sanctioning knowledge to enhance canonical reduction [Rec95].
- Although the scaleability experiments were reasonably encouraging, GRAIL’s absolute performance is still poor. Work is already underway to evaluate the possible benefits of parallel processing in classification [GGJ94].
- GRAIL’s limited support for role cardinality proved to be a serious restriction when attempting to use the system in a multimedia database modelling application [Hau95].

As well as application specific enhancements to GRAIL there are a number of areas which would be of general interest in TKRS research:

- The representation of part-whole relations in TKRSs is an area of active research [Hor94], [PL94], [SPS94]. GRAIL's refinement mechanism gives it a considerable advantage but work is needed to study part-whole inferences and their implementation.
- GRAIL's ***necessary*** statements provide a powerful mechanism for representing assertional information within the classifier. However the current implementation does not support multiple semi-disjoint definitions which are required in order to represent concepts such as "triangle" which can be defined in more than one way: as a 3-sided polygon which necessarily has 3 angles or as a 3-angled polygon which necessarily has 3 sides [Woo91].

7.5.1 The GRAIL classifier

It has been demonstrated in sections 6.4.1 (inferred non-primitive conjunctions) and 6.4.2 (cardinality restrictions and the relation hierarchy) that there are some serious problems with GRAIL's classification algorithm. In certain circumstances the classifier identifies concepts as incoherent when this is not justified by the semantics of the language; in other circumstances it fails to identify provably incoherent concepts. Given these two behaviours the current classifier is certainly incomplete and is arguably unsound. Many useful TKRSs have been shown to have incomplete classifiers but soundness is a fundamental requirement – a system which is both incomplete and unsound allows nothing to be inferred with certainty. It should however be pointed out that the areas of incompleteness and unsoundness in GRAIL are small and (now) well understood.

Complete, sound and provably correct classification algorithms now exist, based on lattice theory [AKBLN89], and have been successfully applied in the CLASSIC system [BPS94]. These algorithms need to be studied to see if they can be adapted to deal with GRAIL's ***necessary*** statements and refinement.

7.6 GRAIL versus LOOM?

It was never the intention of this study to pick an overall 'best' and, given the diversity of the two systems, it would be impossible to do so. GRAIL and LOOM both have powerful features but there is surprisingly little common ground: most of LOOM's expressive terminological language could not be represented in GRAIL and GRAIL's enhanced classification based reasoning proved impossible to implement in LOOM. The diversity of the two systems might prompt a reconsideration of the range of design choices available in TKRSs.

BIBLIOGRAPHY

[AKBLN89] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989.

[Bec95] S. K. Bechhofer. The GRAIL classifier. Unpublished technical report, May 1995. Available from: University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK.

[BFL83] R. J. Brachman, R. E. Fikes, and H. J. Levesque. KRYPTON: A functional approach to knowledge representation. *IEEE Computer*, 16(10):67–73, October 1983.

[BH91] F. Baader and B. Hollunder. KRIS: Knowledge Representation and Inference System. *SIGART Bulletin*, 2(3):8–15, 1991.

[BHH+91] F. Baader, H.-J. Bürckert, J. Heinsohn, B. Hollunder, J. Müller, B. Nebel, W. Nutt, and H.-J. Profitlich. Terminological knowledge representation: A proposal for a terminological logic. Technical Memo TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), 1991.

[BHNP92] F. Baader, B. Hollunder, B. Nebel, and H-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems. In B. Nebel, C. Rich, and W. Swartout, editors, *Principals of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, Cambridge, Massachusetts, pages 270–281. Morgan Kaufmann, San Mateo, California, 1992. Also available as DFKI RR-93-03.

[BMPSR91] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. F. Sowa, editor, *Principals of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. Morgan Kaufmann, San Mateo, California, 1991.

[Bor92] A. Borgida. Description logics are not just for the flightless-birds: A new look at the utility and foundations of description logics. Unpublished paper, June 1992. Available from:
ftp://athos.rutgers.edu/pub/technical-reports/dcs-tr-295.ps.Z.

[BPS94] A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994. Available from:
http://www.cs.washington.edu/research/jair/home.html.

[Bri93] D. Brill. LOOM reference manual version 2.0. University of Southern California, Information Sciences Institute, December 1993. Available from:
http://www.isi.edu/isd/LOOM/LOOM-HOME.html.

[BS94] S. K. Bechhofer and W. D. Solomon. GALEN documentation, volume C: A tutorial introduction to the GRAIL kernel, August 1994. Available from: Project Co-ordinator, University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK.

[BS85] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April 1985.

[DP91] J. Doyle and R. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297, 1991.

[GALEN94] The GALEN implementation and modelling teams. GALEN documentation, volume H: A guide to GALEN 's concept services, August 1994. Available from: Project Co-ordinator, University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK.

[GBS+94] C. A. Goble, S. K. Bechhofer, W. D. Solomon, A. L. Rector, W. A. Nowlan, and A. J. Glowinski. Conceptual, semantic and information models for medicine. In *Proceedings of the 4th European-Japanese Seminar on Information Modelling and Knowledge Bases*, Stockholm, 31st May – 3rd June 1994. To be published in book form by ISO publishing, 1995.

[GGJ94] C. A. Goble, J. R. Gurd, and K. G. Jeffery. PAEPR: Parallel Architecture for Electronic Patient Records. EPSRC AIKMS grant proposal, 1994. Available from: University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK.

[Hau95] C. Haul. Evaluation of Grail on multimedia database modelling. Studienarbeit, Technische Universität Braunschweig, Wintersemester 1994/95. Available from: University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK.

[HKNP94] J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68:367--397, 1994.

[HKQ+93] T. Hoppe, C. Kindermann, J. Quantz, A. Schmiedel, and M. Fischer. BACK v5 – tutorial & manual. KIT Report 100, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, 1993.

[Hor94] P. Hors. Description logics to specify the part-whole relation. In *Proceedings of the ECAI'94 Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 103–109, 1994.

[Kob91] A. Kobsa. First experiences with the SB-ONE knowledge representation workbench in natural-language applications. *SIGART Bulletin*, 2(3):70–76, 1991.

[LB85] H. J. Levesque and R. J. Brachman. A fundamental trade-off in knowledge representation and reasoning. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufmann, Los Altos, California, 1985.

[LG89] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems*. Addison-Wesley, 1989.

[LG91] D. B. Lenat and R. V. Guha. The evolution of CycL, the Cyc representation language. *SIGART Bulletin*, 2(3):84–87, 1991.

[LOOM93] University of Southern California, Information Sciences Institute. LOOM Tutorial, 1993. Available from:
http://www.isi.edu/isd/LOOM/LOOM-HOME.html.

[LOOM94] University of Southern California, Information Sciences Institute. LOOM 2.1 release notes, 1994. Available from:
http://www.isi.edu/isd/LOOM/LOOM-HOME.html.

[Mac91a] R. M. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. F. Sowa, editor, *Principals of Semantic Networks: Explorations in the representation of knowledge*, pages 385–400. Morgan Kaufmann, San Mateo, California, 1991.

[Mac91b] R. M. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[Mac94] R. M. MacGregor. A description classifier for the predicate calculus. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, pages 213–220. 1994.

[Mac95] R. M. MacGregor, USC/Information Sciences Institute. Personal communication, March 1995.

[MB92] R. M. MacGregor and D. Brill. Recognition algorithms for the LOOM classifier. Technical report, University of Southern California, Information Sciences Institute, 1992.

[MDW91] E. Mays, R. Dionne, and R. Weida. K-rep system overview. *SIGART Bulletin*, 2(3):93–97, 1991.

[Neb88] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, April 1988.

[Neb90] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.

[Now93] W. A. Nowlan. Structured methods of information management for medical records. PhD thesis, University of Manchester, 1993.

[NR91] W. A. Nowlan and A. L. Rector. Medical knowledge representation and predictive data entry. In M. Stefanelle, A. Hasman, M. Fieschi, and J. Talmon, editors, *Lecture notes in Medical Informatics 44, Proceedings of AIME 91*, Berlin, Germany, pages 105–116. Springer-Verlag, 1991.

[NRK+90] W. A. Nowlan, A. L. Rector, S. Kay, C. A. Goble, B. Horan, T. J. Howkins, and A. Wilson. PEN&PAD : A doctors' workstation with intelligent data entry and summaries. In R. A. Miller, editor, *Proceedings of the 14th Annual Symposium on Computer Applications in Medical Care (SCAMC 90),* Washington DC, pages 941–942. IEEE Computer Society Press, Los Alamitos, California, 1990.

[NSA+94] S. Navathe, A. Savasere, T. Anwar, H. Beck, and S. Gala. Object modelling using classification in CANDIDE and its applications. In A. Dogac, T. Ozsu, A. Briliris, and T. Sellis, editors, *Advances in Object-Oriented Database Systems*, pages 435–476. NATO ASI Series Vol 130, 1994.

[Pel91] C. Peltason. The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119, 1991.

[PFPS+92] R. S. Patil and R. E. Fikes and P. F. Patel-Schneider and D. McKay and T. Finin and T. R. Gruber and R. Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebel, C. Rich, and W. Swartout, editors, *Principals of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, Cambridge, Massachusetts, pages 485–496. Morgan Kaufmann, San Mateo, California, 1992.

[PL94] L. Padgham and P. Lambrix. A framework for part-of hierarchies in terminological logics. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principals of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR '94)*, Bonn, Germany, pages 485–496. Morgan Kaufmann, San Francisco, California, 1994.

[PS84] P. F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principals of Knowledge Based Systems*, pages 559–565, December 1984. Extended version available as AI Technical Report No. 37, Schlumberger Palo Alto Research, (October 1984).

[PS89] P. F. Patel-Schneider. Undecidability of subsumption in nikl. *Artificial Intelligence*, 39(2):263–272, 1989.

[PS91] P. F. Patel-Schneider. The CLASSIC knowledge representation system: Guiding principals and implementation rationale. *SIGART Bulletin*, 2(3):108–113, 1991.

[PSS93] P. F. Patel-Schneider and B. Swartout. Working version (draft): Description logic specification from the KRSS effort, June 1993. Available from:
http://WWW–KSL.Stanford.EDU:80/knowledge–sharing/papers/dl-spec.ps

[Rec95] A. L. Rector. Why GRAIL & why GRAIL II. Paper in preparation, Medical Informatics Group, University of Manchester.

[RGG+94] A. L. Rector, A. Gangemi, E. Galeazzi, A. J. Glowinski, and A Rossi-Mori. The GALEN CORE model schemata for anatomy: towards a re-usable application-independent model of medical concepts. In P. Barahona, M. Veloso, and J. Bryant, editors, *Proceedings of Medical Informatics in Europe (MIE 94),* pages 229–233, 1994.

[RNG93] A. L. Rector, W A Nowlan, and A Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC 93),* pages 414–418, Washington DC, USA, 1993.

[RNGM93] A. L. Rector, W. A. Nowlan, A. J. Glowinski, and G. Matthews. The GRAIL kernel: GALEN Representation And Integration Language, version 1. Available from: Project Co-ordinator, University of Manchester, Medical Informatics Group, Department of Computer Science, Manchester M13 9PL, UK, March 1993.

[RNK92] A. L. Rector, W. A. Nowlan, and S. Kay. Conceptual knowledge: the core of medical information systems. In K. C. Lun, P. Degoulet, and T. E. Rienhoff, editors, *Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92)*, pages 1420–1426. North-Holland Publishers, 1992.

[RSNR94] A. L. Rector, W. D. Solomon, W. A. Nowlan, and T. W. Rush. A terminology server for medical language and medical information systems. *In Proceedings of IMIA WG6*, Geneva, May 1994.

[Rus95] T. A. Russ, USC/Information Sciences Institute. Personal communication, February 1995.

[Sow84] J. F. Sowa. *Conceptual structures: Information processing in mind and machine.* Addison-Wesley, 1984.

[SPS94] P-H. Speel and P. F. Patel-Schneider. A whole-part extension for description logics. In *Proceedings of the ECAI'94 Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 111–121, 1994.

[SS89] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference (KR '89)*, pages 421–431. Morgan Kaufmann, 1989.

[Woo91] W. A. Woods. Understanding subsumption and taxonomy: a framework for progress. In J. F. Sowa, editor, *Principals of semantic networks: Explorations in the representation of knowledge*, pages 45–94. Morgan Kaufmann, San Mateo, California, 1991.

[WS92] W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers and Mathematics with Applications – Special Issue on Artificial Intelligence*, 23(2-5):133–177, 1992.

APPENDIX 1 GRAIL SYNTAX

Producing a parser revealed a number of errors and omissions in the existing formal syntax specification. The following specification is adapted from [RNGM93] with the main differences being:

- All operations are terminated with a period (including directives);
- The use of a semicolon for repeat operations on an entity;
- Separate definitions for criteria/non-expandable lists and entities/expandable lists.

The specification is not claimed to be exhaustive, particularly with respect to compiler directives, but is sufficient to parse the CORE model.

```
%%% Token Classes %%%

<whitespace>    :[\s\t\r]+
<comment>       :\"~[\"]*\"
<identifier>    :[a-zA-Z_][a-zA-Z0-9_]*
<variable>      :$[a-zA-Z_][a-zA-Z0-9_]*
<integer>       :[0-9]+
<string>        :'~[']*'


%%% Production Rules %%%

Operations      : [Operation '.']*
                ;

Operation       : Directive
                | Assignment
                | Statement
                ;
```

```
Directive       : '#include'
                | '#include' String
                | '#path' String
                | '#addPathSegment' String
                | '#removePathSegment'
                | '#saveImageAs' String
                | '#begin' Identifier
                | '#end'
                | '#model' String
                | '#modelTitle' String
                | '#require' String
                | '#requireTMVersion' String
                | '#author' Identifier
                | '#resetAuthor'
                | '#printTime'
                | '#printMessage' String
                ;

Assignment      : <variable> '=' Entity

Statement       : Entity
                | Entity Constructs
                ;

Constructs      : Construct
                | Construct ';' Constructs
                ;

Construct       : MonadicOp
                | DyadicOp Entity
                | CriterionOp Criteria
                | TripleOp Criteria Qualifier
                | CommentOp String
                | 'newAttribute' Identifier Identifier Inheritance Cardinality
                | 'descriptions' Qualifier
                | 'precedence' Number
                ;

MonadicOp       : 'dependants'
                | 'retract'
                | 'subs'
                | 'supers'
                | 'transitiveDown'
                | 'isAbstract'
                ;
```

```
DyadicOp        : 'addSub'
                | 'addSuper'
                | 'name'
                | 'newSub'
                | 'refinedAlong'
                | 'specialisedBy'
                ;

CriterionOp     : 'which'
                | 'whichG'
                | 'grammatically'
                | 'sensibly'
                | 'necessarily'
                | 'topicNecessarily'
                | 'valueNecessarily'
                | 'grammaticallyAndSensibly'
                | 'grammaticallySensiblyAndNecessarily'
                | 'grammaticallySensiblyAndTopicNecessarily'
                | 'grammaticallySensiblyAndValueNecessarily'
                | 'sensiblyAndNecessarily'
                | 'sensiblyAndTopicNecessarily'
                | 'sensiblyAndValueNecessarily'
                | 'mayNot'
                | 'byDefault'
                ;

TripleOp        : 'triple'
                | 'whichQ'
                ;

Entity          : Entity1
                | '[' Entity+ ']'
                ;

Entity1         : Identifier
                | Number
                | '(' Statement ')'
                ;

Criteria :      Entity Entity
                | '<' Criteria1+ '>'
                ;

Criteria1       : Entity1 Entity1
                ;
```

```
Qualifier       : Qualifier1
                | '[' Qualifier+ ']'
                ;

Qualifier1      : 'conceivable'
                | 'grammatical'
                | 'possible'
                | 'necessary'
                ;

Cardinality     : Cardinality1
                | '[' Cardinality+ ']'
                ;

Cardinality1    : 'oneOne'
                | 'oneMany'
                | 'manyOne'
                | 'manyMany'
                ;

Inheritance     : Inheritance1
                | '[' Inheritance+ ']'
                ;

Inheritance1    : 'nilNil'
                | 'nilAll'
                | 'allNil'
                | 'allAll'
                ;

Identifier      : <identifier>
Number          : <integer>
String          : <string>
```

APPENDIX 2 EXAMPLE TRANSLATION

The following is a short example showing the translation of a GRAIL knowledge base into LOOM. The knowledge base used is a 'toy' example taken from the GRAIL Tutorial [BS94]; it demonstrates the main features of GRAIL including ***necessary*** statements, sanctioning and refinement. The translation module is used in its fully featured mode – as in the 'A' version of the CORE model – which includes the A-box implementation of ***necessary*** statements and refinement; this is made possible by the small size of the model.

A2.1 GRAIL source code

This section shows the GRAIL source code which defines the concepts and relations which make up the knowledge base.

```
"** Create a primitive concept hierarchy."
TopCategory newSub TransportCategory.
"** TopCategory is a built-in concept used as Top in the concept hierarchy."
TransportCategory newSub Vehicle.
Vehicle newSub [RoadVehicle RailVehicle
          ElectricVehicle PetrolVehicle DieselVehicle].
"** GRAIL syntax - a list enclosed in square brackets is equivalent to the
   expanded form Vehicle newSub RoadVehicle. Vehicle newSub RailVehicle. etc."
RoadVehicle newSub Bus.
"** Bus is both a RoadVehicle and a DieselVehicle."
DieselVehicle addSub Bus.
RoadVehicle newSub Car.
"** Car is both a RoadVehicle and a PetrolVehicle."
PetrolVehicle addSub Car.
RailVehicle newSub Train.
RailVehicle newSub Tram.
"** Tram is both a RailVehicle and an ElectricVehicle."
Tram addSuper ElectricVehicle.
TopCategory newSub Person.
(SymbolicValueType newSub SexValueType) newSub [male female].
(SymbolicValueType newSub AgeValueType) newSub [young old].

"** Create some relations. In this example the relation hierarchy is completely
   'flat' - all relations are direct subs of the Top relation."
Attribute newAttribute DomainAttribute inverseDomainAttribute allAll manyMany.
"** Attribute is a built-in relation used as Top in the relation hierarchy."
DomainAttribute newAttribute hasDriver isDriverOf allAll manyOne.
DomainAttribute newAttribute hasSex isSexOf allAll manyOne.
DomainAttribute newAttribute hasAge isAgeOf allAll manyOne.
DomainAttribute newAttribute hasInsuranceRating isRatingOf allAll manyOne.

"** Sanction some conjunctions."
Vehicle grammatically hasDriver Person.
"** It is reasonable to talk about Vehicles having drivers of type Person."
RoadVehicle sensibly hasDriver Person.
```

```
"** In our model only RoadVehicles can actually have drivers."
Person grammaticallyAndSensibly hasSex SexValueType.
Person grammaticallyAndSensibly hasAge AgeValueType.

"** An example of a necessary statement - an assertion that the concept formed
   from the conjunction of Person and (∃isDriverOf:RoadVehicle) is also subsumed
   by the concept (hasAge:old)."
(Person which isDriverOf RoadVehicle) necessarily hasAge old.

"** Form some non-primitive conjunctions and name them."
((Person which hasSex male) which hasAge old) name Man.
((Person which hasSex male) which hasAge young) name Boy.
((Person which hasSex female) which hasAge old) name Woman.
((Person which hasSex female) which hasAge young) name Girl.
(Person which isDriverOf Car) name CarDriver.

"** Create a non-primitive conjunction which is only grammatically sanctioned
   and use it to add sanctioning knowledge at a more general level than would
   otherwise be possible."
(Person whichG isDriverOf Vehicle) grammaticallyAndSensibly
         hasInsuranceRating IntegerValueType.
"** IntegerValueType is a built in concept representing the set of integers."

"** An example of refinement."
(SymbolicValueType newSub CompanyValueType) newSub
         [britRail busCo megaCorp gmBuses].
DomainAttribute newAttribute isOwnedBy owns allAll manyMany.
DomainAttribute newAttribute isPartOf contains allAll manyMany.
isOwnedBy specialisedBy isPartOf.
Vehicle grammaticallyAndSensibly isOwnedBy CompanyValueType.
CompanyValueType grammaticallyAndSensibly isPartOf CompanyValueType.
busCo necessarily isPartOf megaCorp.
"** Any concept which is subsumed by (∃isOwnedBy:busCo) will now also be subsumed
   by (∃isOwnedBy:megaCorp).
   For example:-"
Bus which isOwnedBy busCo.

"** An example of transitivity:-"
isPartOf transitiveDown.
"** NOTE: this is bad practice - the relation isPartOf should have been fully
   defined, including the transitivity characteristic, before being used in
   concept forming statements."
CompanyValueType newSub enormousMultiNational.
megaCorp necessarily isPartOf enormousMultiNational.
"** Any concept which is subsumed by (∃isPartOf:megaCorp) – for example busCo –
   will now also be subsumed by (∃isPartOf:enormousMultiNational)."

"** Form some un-named non-primitive conjunctions (specialisations). Note that
   these add no knowledge to the model - their potential existence is implied
   by the sanction 'RoadVehicle sensibly hasDriver Person'."
Car which hasDriver Person.
Car which hasDriver Man.
```

A2.2 LOOM knowledge base

This section shows a LOOM dump of the translated knowledge base; it consists of the concept, relation and instance definitions which make up the LOOM version of the knowledge base. Due to problems with the LOOM ***rename-concept*** function (as described in section 4.3.3) GRAIL knowledge names are not reflected in the names of non-primitive LOOM concepts; to aid clarity, comments have been added which include GRAIL concept definitions and knowledge names where applicable.

```
;** Relation definitions.
(defrelation |Attribute|)
(defrelation Criteria-List)
(defrelation |DomainAttribute|
  :is-primitive |Attribute|
  :characteristics :monotonic)
(defrelation Grammatical)
(defrelation Necessary)
(defrelation Sensible)
(defrelation |contains|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)
(defrelation |hasAge|
  :is-primitive |DomainAttribute|
  :characteristics :monotonic)
(defrelation |hasDriver|
  :is-primitive |DomainAttribute|
  :characteristics :monotonic)
(defrelation |hasInsuranceRating|
  :is-primitive |DomainAttribute|
  :characteristics :monotonic)
(defrelation |hasSex|
  :is-primitive |DomainAttribute|
  :characteristics :monotonic)
(defrelation |inverseDomainAttribute|
  :is-primitive |Attribute|
  :characteristics :monotonic)
(defrelation |isAgeOf|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)
(defrelation |isDriverOf|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)

;** An example of refinement – isOwnedBy specialisedBy isPartOf.
(defrelation |isOwnedBy|
  :is-primitive |DomainAttribute|
  :characteristics (:open-world :monotonic)
  :antecedents (:satisfies (?X ?Z) (:for-some ?Y (:and (|isOwnedBy| ?X ?Y) (|isPartOf| ?Y ?Z)))))

;** An example of transitivity – isPartOf transitiveDown.
(defrelation |isPartOf|
  :is-primitive |DomainAttribute|
  :characteristics (:open-world :monotonic)
  :antecedents (:satisfies (?X ?Z) (:for-some ?Y (:and (|isPartOf| ?X ?Y) (|isPartOf| ?Y ?Z)))))

(defrelation |isRatingOf|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)
(defrelation |isSexOf|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)
(defrelation |owns|
  :is-primitive |inverseDomainAttribute|
  :characteristics :monotonic)

;** Concept definitions.
(defconcept |AgeValueType|
  :is-primitive |SymbolicValueType|
  :implies (:and (:filled-by Grammatical '(|isAgeOf| |Person|))
               (:filled-by Sensible '(|isAgeOf| |Person|)))
  :characteristics (:open-world :monotonic))

;** |Bus| is both a |RoadVehicle| and a |DieselVehicle|.
(defconcept |Bus|
  :is-primitive (:and |DieselVehicle| |RoadVehicle|)
  :characteristics :monotonic)
;** Bus which isOwnedBy busCo.
;    Instances of this concept – such as |I*Bus*G24158| – would also be recognised by the
;    A-box as instances of the concept (∃isOwnedBy:megaCorp).
(defconcept |Bus*G24158|
  :is (:and |Bus|
          (:some |isOwnedBy| |busCo|))
  :implies (:and (:filled-by Grammatical (|hasDriver| |Person|)
                         (|isOwnedBy| |CompanyValueType|))
               (:at-least 2 Grammatical)
               (:filled-by Sensible (|isOwnedBy| |CompanyValueType|) (|hasDriver| |Person|))
               (:at-least 2 Sensible)
               (:some |isOwnedBy| |busCo|)
               (:filled-by Criteria-List (|isOwnedBy| |busCo|))
               (:at-least 1 Criteria-List))
  :defaults (:and (:filled-by |isOwnedBy| |I*busCo|))
  :characteristics (:monotonic :open-world))

;** |Car| is both a |RoadVehicle| and a |PetrolVehicle|.
(defconcept |Car|
  :is-primitive (:and |PetrolVehicle| |RoadVehicle|)
  :characteristics :monotonic)

;** Car which hasDriver Person.
(defconcept |Car*G24525|
  :is (:and |Car|
          (:some |hasDriver| |Person|)
          (:all |hasDriver| |Person|)
          (:at-most 1 |hasDriver|))
  :implies (:and (:filled-by Grammatical (|hasDriver| |Person|)
                         (|isOwnedBy| |CompanyValueType|))
```

```
                (:at-least 2 Grammatical)
                (:filled-by Sensible (|isOwnedBy| |CompanyValueType|) (|hasDriver| |Person|))
                (:at-least 2 Sensible)
                (:some |hasDriver| |Person|)
                (:all |hasDriver| |Person|)
                (:at-most 1 |hasDriver|)
                (:filled-by Criteria-List (|hasDriver| |Person|))
                (:at-least 1 Criteria-List))
  :defaults (:and (:filled-by |hasDriver| |I*Person|))
  :characteristics (:monotonic :open-world))
;** Car which hasDriver Man.
(defconcept |Car*G24671|
  :is (:and |Car|
           (:some |hasDriver| |Person*G23868|)
           (:all |hasDriver| |Person*G23868|)
           (:at-most 1 |hasDriver|))
  :implies (:and (:filled-by Sensible (|isOwnedBy| |CompanyValueType|)
                        (|hasDriver| |Person|))
                (:at-least 2 Sensible)
                (:filled-by Grammatical (|hasDriver| |Person|)
                        (|isOwnedBy| |CompanyValueType|))
                (:at-least 2 Grammatical)
                (:some |hasDriver| |Person*G23868|)
                (:all |hasDriver| |Person*G23868|)
                (:at-most 1 |hasDriver|)
                (:filled-by Criteria-List (|hasDriver| |Person|) (|hasDriver| |Person*G23868|))
                (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |hasDriver| |I*Person|)
                 (:filled-by |hasDriver| |I*Person*G23868|))
  :characteristics (:monotonic :open-world))

(defconcept |CompanyValueType|
  :is-primitive |SymbolicValueType|
  :implies (:and (:filled-by Grammatical '(|owns| |Vehicle|))
                (:filled-by Sensible '(|owns| |Vehicle|))
                (:filled-by Grammatical '(|isPartOf| |CompanyValueType|))
                (:filled-by Grammatical '(|contains| |CompanyValueType|))
                (:filled-by Sensible '(|isPartOf| |CompanyValueType|))
                (:filled-by Sensible '(|contains| |CompanyValueType|)))
  :characteristics (:open-world :monotonic))
(defconcept |DieselVehicle|
  :is-primitive |Vehicle|
  :characteristics :monotonic)
(defconcept |DomainCategory|)
(defconcept |ElectricVehicle|
  :is-primitive |Vehicle|
  :characteristics :monotonic)
(defconcept |IntegerValueType|
  :is-primitive Loom::Built-In-Theory^Thing
  :implies (:and (:filled-by Grammatical '(|isRatingOf| |Person*G24032|))
                (:filled-by Sensible '(|isRatingOf| |Person*G24032|)))
  :characteristics :open-world)
(defconcept |Person|
  :is-primitive |TopCategory|
  :implies (:and (:filled-by Grammatical '(|isDriverOf| |Vehicle|))
               (:filled-by Sensible '(|isDriverOf| |RoadVehicle|))
               (:filled-by Grammatical '(|hasSex| |SexValueType|))
               (:filled-by Sensible '(|hasSex| |SexValueType|))
               (:filled-by Grammatical '(|hasAge| |AgeValueType|))
               (:filled-by Sensible '(|hasAge| |AgeValueType|)))
  :characteristics (:open-world :monotonic))
;** Person which isDriverOf RoadVehicle.
;   The necessary statement:
;   (Person which isDriverOf RoadVehicle) necessarily hasAge old.
;   has become the assertion:
;   :implies (:and (:some |hasAge| |old|) (:all |hasAge| |old|) (:at-most 1 |hasAge|)).
;   All instances of this concept will be recognised by the A-box as instances of
;   the concept (hasAge:old)."
(defconcept |Person*G23810|
  :is (:and |Person|
          (:some |isDriverOf| |RoadVehicle|))
  :implies (:and (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                      (|hasAge| |AgeValueType|))
               (:at-least 3 Sensible)
               (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                      (|hasAge| |AgeValueType|))
               (:at-least 3 Grammatical)
               (:some |isDriverOf| |RoadVehicle|)
               (:filled-by Criteria-List (|isDriverOf| |RoadVehicle|))
               (:at-least 1 Criteria-List)
               (:some |hasAge| |old|)
               (:all |hasAge| |old|)
               (:at-most 1 |hasAge|)
               (:filled-by Necessary '(|hasAge| |old|)))
  :defaults (:and (:filled-by |isDriverOf| |I*RoadVehicle|)
               (:filled-by |hasAge| |I*old|))
  :characteristics (:recursive :monotonic :open-world))

;** Person which hasSex male.
(defconcept |Person*G23853|
  :is (:and |Person|
          (:some |hasSex| |male|)
          (:all |hasSex| |male|)
          (:at-most 1 |hasSex|))
  :implies (:and (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                      (|hasAge| |AgeValueType|))
               (:at-least 3 Sensible)
               (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                      (|hasAge| |AgeValueType|))
               (:at-least 3 Grammatical)
               (:some |hasSex| |male|)
               (:all |hasSex| |male|)
               (:at-most 1 |hasSex|)
               (:filled-by Criteria-List (|hasSex| |male|))
               (:at-least 1 Criteria-List)
               (:filled-by Criteria-List '(|hasSex| |male|)))
  :defaults (:and (:filled-by |hasSex| |I*male|))
  :characteristics (:recursive :monotonic :open-world))
;** ((Person which hasSex male) which hasAge old) name Man.
(defconcept |Person*G23868|
  :is (:and |Person*G23853|
          (:some |hasSex| |male|)
          (:all |hasSex| |male|)
          (:at-most 1 |hasSex|)
          (:some |hasAge| |old|)
          (:all |hasAge| |old|)
          (:at-most 1 |hasAge|))
  :implies (:and (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                     (|hasAge| |AgeValueType|))
              (:at-least 3 Grammatical)
              (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                     (|hasAge| |AgeValueType|))
              (:at-least 3 Sensible)
              (:some |hasAge| |old|)
              (:all |hasAge| |old|)
              (:at-most 1 |hasAge|)
              (:some |hasSex| |male|)
              (:all |hasSex| |male|)
              (:at-most 1 |hasSex|)
              (:filled-by Criteria-List (|hasSex| |male|) (|hasAge| |old|))
              (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |hasSex| |I*male|)
               (:filled-by |hasAge| |I*old|))
  :characteristics (:monotonic :open-world))

;** ((Person which hasSex male) which hasAge young) name Boy.
(defconcept |Person*G23900|
  :is (:and |Person*G23853|
          (:some |hasSex| |male|)
          (:all |hasSex| |male|)
          (:at-most 1 |hasSex|)
          (:some |hasAge| |young|)
          (:all |hasAge| |young|)
          (:at-most 1 |hasAge|))
  :implies (:and (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                     (|hasAge| |AgeValueType|))
              (:at-least 3 Grammatical)
              (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                     (|hasAge| |AgeValueType|))
              (:at-least 3 Sensible)
              (:some |hasAge| |young|)
              (:all |hasAge| |young|)
              (:at-most 1 |hasAge|)
              (:some |hasSex| |male|)
              (:all |hasSex| |male|)
              (:at-most 1 |hasSex|)
              (:filled-by Criteria-List (|hasSex| |male|) (|hasAge| |young|))
              (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |hasSex| |I*male|)
                (:filled-by |hasAge| |I*young|))
  :characteristics (:monotonic :open-world))
;** Person which hasSex female.
(defconcept |Person*G23927|
  :is (:and |Person|
           (:some |hasSex| |female|)
           (:all |hasSex| |female|)
           (:at-most 1 |hasSex|))
  :implies (:and (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
                (:at-least 3 Sensible)
                (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
                (:at-least 3 Grammatical)
                (:some |hasSex| |female|)
                (:all |hasSex| |female|)
                (:at-most 1 |hasSex|)
                (:filled-by Criteria-List (|hasSex| |female|))
                (:at-least 1 Criteria-List)
                (:filled-by Criteria-List '(|hasSex| |female|)))
  :defaults (:and (:filled-by |hasSex| |I*female|))
  :characteristics (:recursive :monotonic :open-world))

;** ((Person which hasSex female) which hasAge old) name Woman.
(defconcept |Person*G23942|
  :is (:and |Person*G23927|
           (:some |hasSex| |female|)
           (:all |hasSex| |female|)
           (:at-most 1 |hasSex|)
           (:some |hasAge| |old|)
           (:all |hasAge| |old|)
           (:at-most 1 |hasAge|))
  :implies (:and (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
                (:at-least 3 Grammatical)
                (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
                (:at-least 3 Sensible)
                (:some |hasAge| |old|)
                (:all |hasAge| |old|)
                (:at-most 1 |hasAge|)
                (:some |hasSex| |female|)
                (:all |hasSex| |female|)
                (:at-most 1 |hasSex|)
                (:filled-by Criteria-List (|hasSex| |female|) (|hasAge| |old|))
                (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |hasSex| |I*female|)
                (:filled-by |hasAge| |I*old|))
  :characteristics (:monotonic :open-world))
```

```
;** ((Person which hasSex female) which hasAge young) name Girl.
(defconcept |Person*G23974|
  :is (:and |Person*G23927|
          (:some |hasSex| |female|)
          (:all |hasSex| |female|)
          (:at-most 1 |hasSex|)
          (:some |hasAge| |young|)
          (:all |hasAge| |young|)
          (:at-most 1 |hasAge|))
  :implies (:and (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                    (|hasAge| |AgeValueType|))
              (:at-least 3 Grammatical)
              (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                    (|hasAge| |AgeValueType|))
              (:at-least 3 Sensible)
              (:some |hasAge| |young|)
              (:all |hasAge| |young|)
              (:at-most 1 |hasAge|)
              (:some |hasSex| |female|)
              (:all |hasSex| |female|)
              (:at-most 1 |hasSex|)
              (:filled-by Criteria-List (|hasSex| |female|) (|hasAge| |young|))
              (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |hasSex| |I*female|)
              (:filled-by |hasAge| |I*young|))
  :characteristics (:monotonic :open-world))

;** Person which isDriverOf Car.
(defconcept |Person*G24001|
  :is (:and |Person|
          (:some |isDriverOf| |Car|))
  :implies (:and (:filled-by Necessary (|hasAge| |old|))
              (:at-least 1 Necessary)
              (:some |hasAge| |old|)
              (:all |hasAge| |old|)
              (:at-most 1 |hasAge|)
              (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                    (|hasAge| |AgeValueType|))
              (:at-least 3 Grammatical)
              (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                    (|hasAge| |AgeValueType|))
              (:at-least 3 Sensible)
              (:some |isDriverOf| |Car|)
              (:filled-by Criteria-List (|isDriverOf| |RoadVehicle|) (|isDriverOf| |Car|))
              (:at-least 2 Criteria-List))
  :defaults (:and (:filled-by |isDriverOf| |I*RoadVehicle|)
              (:filled-by |isDriverOf| |I*Car|) (:filled-by |hasAge| |I*old|))
  :characteristics (:monotonic :open-world))
```

```
;** Person whichG isDriverOf Vehicle.
(defconcept |Person*G24032|
  :is (:and |Person|
          (:some |isDriverOf| |Vehicle|))
  :implies (:and (:filled-by Sensible (|isDriverOf| |RoadVehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
               (:at-least 3 Sensible)
               (:filled-by Grammatical (|isDriverOf| |Vehicle|) (|hasSex| |SexValueType|)
                        (|hasAge| |AgeValueType|))
               (:at-least 3 Grammatical)
               (:some |isDriverOf| |Vehicle|)
               (:filled-by Criteria-List (|isDriverOf| |Vehicle|))
               (:at-least 1 Criteria-List)
               (:filled-by Grammatical '(|hasInsuranceRating| |IntegerValueType|))
               (:filled-by Sensible '(|hasInsuranceRating| |IntegerValueType|)))
  :defaults (:and (:filled-by |isDriverOf| |I*Vehicle|))
  :characteristics (:recursive :monotonic :open-world))

(defconcept |PetrolVehicle|
  :is-primitive |Vehicle|
  :characteristics :monotonic)
(defconcept |RailVehicle|
  :is-primitive |Vehicle|
  :characteristics :monotonic)
(defconcept |RoadVehicle|
  :is-primitive |Vehicle|
  :implies (:and (:filled-by Sensible '(|hasDriver| |Person|)))
  :characteristics (:open-world :monotonic))
(defconcept |SexValueType|
  :is-primitive |SymbolicValueType|
  :implies (:and (:filled-by Grammatical '(|isSexOf| |Person|))
               (:filled-by Sensible '(|isSexOf| |Person|)))
  :characteristics (:open-world :monotonic))
(defconcept |SymbolicValueType|)
(defconcept |TopCategory|)
(defconcept |Train|
  :is-primitive |RailVehicle|
  :characteristics :monotonic)

;** |Tram| is both a |RailVehicle| and an |ElectricVehicle|.
(defconcept |Tram|
  :is-primitive (:and |ElectricVehicle| |RailVehicle|)
  :characteristics :monotonic)

(defconcept |TransportCategory|
  :is-primitive |TopCategory|
  :characteristics :monotonic)
```

```
(defconcept |Vehicle|
  :is-primitive |TransportCategory|
  :implies (:and (:filled-by Grammatical '(|hasDriver| |Person|))
               (:filled-by Grammatical '(|isOwnedBy| |CompanyValueType|))
               (:filled-by Sensible '(|isOwnedBy| |CompanyValueType|)))
  :characteristics (:open-world :monotonic))
(defconcept |britRail|
  :is-primitive |CompanyValueType|
  :characteristics :monotonic)

;** Instances of this concept – such as |I*busCo| – would also be recognised by the
;    A-box as instances of the concept (∃isPartOf:enormousMultiNational).
(defconcept |busCo|
  :is-primitive |CompanyValueType|
  :implies (:and (:some |isPartOf| |megaCorp|)
               (:filled-by Necessary '(|isPartOf| |megaCorp|)))
  :defaults (:and (:filled-by |isPartOf| |I*megaCorp|))
  :characteristics (:open-world :monotonic))

(defconcept |enormousMultiNational|
  :is-primitive |CompanyValueType|
  :implies (:and (:some |contains| |megaCorp|)
               (:filled-by Necessary '(|contains| |megaCorp|)))
  :defaults (:and (:filled-by |contains| |I*megaCorp|))
  :characteristics (:open-world :monotonic))
(defconcept |female|
  :is-primitive |SexValueType|
  :characteristics :monotonic)
(defconcept |gmBuses|
  :is-primitive |CompanyValueType|
  :characteristics :monotonic)
(defconcept |male|
  :is-primitive |SexValueType|
  :characteristics :monotonic)
(defconcept |megaCorp|
  :is-primitive |CompanyValueType|
  :implies (:and (:some |contains| |busCo|)
               (:filled-by Necessary '(|contains| |busCo|))
               (:some |isPartOf| |enormousMultiNational|)
               (:filled-by Necessary '(|isPartOf| |enormousMultiNational|)))
  :defaults (:and (:filled-by |contains| |I*busCo|)
               (:filled-by |isPartOf| |I*enormousMultiNational|))
  :characteristics (:open-world :monotonic))
(defconcept |old|
  :is-primitive |AgeValueType|
  :implies (:and (:some |isAgeOf| |Person*G23810|)
               (:filled-by Necessary '(|isAgeOf| |Person*G23810|)))
  :defaults (:and (:filled-by |isAgeOf| |I*Person*G23810|))
  :characteristics (:open-world :monotonic))
(defconcept |young|
  :is-primitive |AgeValueType|
  :characteristics :monotonic)
```

```
;** Instantiate all concepts
(CREATE '|I*Bus| '|Bus|)
(CREATE '|I*Car*G24671| '|Car*G24671|)
(CREATE '|I*Person*G23810| '|Person*G23810|)
(CREATE '|I*Person*G23868| '|Person*G23868|)
(CREATE '|I*Person*G23974| '|Person*G23974|)
(CREATE '|I*Person| '|Person|)

;** Inherited default implications ⇒ |I*Bus*G24158| |isOwnedBy| |I*busCo|
;    and |I*busCo| |isPartOf| |I*megaCorp|. From the definition of the |isOwnedBy|
;    relation the A-box is therefore able to infer |I*Bus*G24158| |isOwnedBy| |I*megaCorp|.
(CREATE '|I*Bus*G24158| '|Bus*G24158|)

(CREATE '|I*Car| '|Car|)
(CREATE '|I*britRail| '|britRail|)
(CREATE '|I*female| '|female|)
(CREATE '|I*RailVehicle| '|RailVehicle|)
(CREATE '|I*AgeValueType| '|AgeValueType|)
(CREATE '|I*TransportCategory| '|TransportCategory|)
(CREATE '|I*enormousMultiNational| '|enormousMultiNational|)
(CREATE '|I*Person*G24001| '|Person*G24001|)
(CREATE '|I*CompanyValueType| '|CompanyValueType|)
(CREATE '|I*young| '|young|)
(CREATE '|I*Car*G24525| '|Car*G24525|)
(CREATE '|I*RoadVehicle| '|RoadVehicle|)
(CREATE '|I*old| '|old|)
(CREATE '|I*DieselVehicle| '|DieselVehicle|)
(CREATE '|I*Tram| '|Tram|)
(CREATE '|I*Person*G23853| '|Person*G23853|)
(CREATE '|I*male| '|male|)
(CREATE '|I*Person*G23900| '|Person*G23900|)

;** Inherited default implications ⇒ |I*busCo| |isPartOf| |I*megaCorp| and
;    |I*megaCorp| |isPartOf| |I*enormousMultiNational|. From the definition of the
;    |isPartOf| relation the A-box is therefore able to infer
;    |I*busCo| |isPartOf| |I*enormousMultiNational|.
(CREATE '|I*busCo| '|busCo|)

(CREATE '|I*gmBuses| '|gmBuses|)
(CREATE '|I*Vehicle| '|Vehicle|)
(CREATE '|I*PetrolVehicle| '|PetrolVehicle|)
(CREATE '|I*SexValueType| '|SexValueType|)
(CREATE '|I*Train| '|Train|)
(CREATE '|I*Person*G23927| '|Person*G23927|)
(CREATE '|I*Person*G24032| '|Person*G24032|)
(CREATE '|I*ElectricVehicle| '|ElectricVehicle|)
(CREATE '|I*Person*G23942| '|Person*G23942|)
(CREATE '|I*megaCorp| '|megaCorp|)
```