Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Reasoning Processes, Methods and Systems for Use in Knowledge-Based Situation Analysis Support Systems

Jean Roy
Alain Auger
DRDC Valcartier

Defence R&D Canada – Valcartier
Technical Memorandum
DRDC Valcartier TM 2006-756
October 2008

Canada

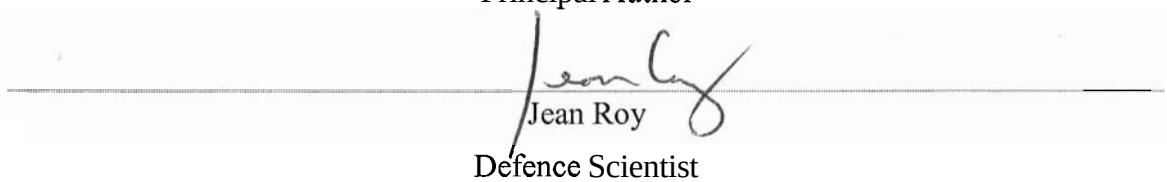
Reasoning Processes, Methods and Systems for Use in Knowledge-Based Situation Analysis Support Systems

Jean Roy
Alain Auger
DRDC Valcartier

Defence R&D Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2006-756
October 2008

Principal Author



Jean Roy

Defence Scientist

Approved by

Stéphane Paradis

Section Head / Intelligence & Information Section, DRDC Valcartier

Approved for release by

Christian Carrier

Chief Scientist, DRDC Valcartier

This document results from S&T work conducted under DRDC projects 11bh (Halifax Class Situation Analysis Support Systems), 11hg (Collaborative Knowledge Exploitation for Maritime Domain Awareness), and 120f (Situation Analysis for the Tactical Army Commander)

Abstract

Adopting a knowledge-centric view of situation analysis and information fusion ultimately requires that one cares about knowledge representation and reasoning, i.e., the area of artificial intelligence concerned with how knowledge can be represented symbolically and manipulated in an automated way by programs simulating reasoning. This memorandum reviews reasoning and inference processes, methods and systems. Aspects being discussed include the notion of an inference engine and other related concepts such as logical arguments, inference chains and theorems. Forward chaining and backward chaining are briefly described. Three basic categories of reasoning, i.e., deduction, induction, and abduction are presented, along with other reasoning methods including analogical, generate-and-test, model-based, qualitative, default, and autoepistemic reasoning. Many of the reasoning/inference systems developed and used over the years to achieve automated reasoning in computer systems are discussed. Emphasis is given to logic, rule-based, frame-based, and case-based reasoning systems, as these paradigms and systems are the most relevant to the development of knowledge-based situation analysis support systems. Finally, non-monotonic systems that allow the retraction of facts and truth maintenance systems are presented. The memorandum provides a comprehensive description of reasoning, thereby contributing to the development of a foundational R&D framework for two projects at Defence R&D Canada (11hg, Collaborative Knowledge Exploitation for Maritime Domain Awareness, and 12of, Situation Analysis for the Tactical Army Commander).

Résumé

Adopter une vue de l'analyse de la situation et de la fusion d'information centrée sur la connaissance requiert ultimement que l'on se soucie de la représentation de la connaissance et du raisonnement, i.e., l'aire de l'intelligence artificielle concernée par comment la connaissance peut être représentée symboliquement et manipulée de façon automatique par des programmes qui simulent le raisonnement. Ce mémoire passe en revue les processus, méthodes et systèmes de raisonnement et d'inférence. Les aspects discutés incluent la notion de moteur d'inférence et d'autres concepts reliés, tel que les arguments logiques, les chaînes d'inférence et les théorèmes. Les enchaînements vers l'avant et vers l'arrière sont brièvement décrits. Les trois catégories de base de raisonnement, i.e., la déduction, l'induction, et l'abduction sont présentées, en plus d'autres méthodes de raisonnement incluant le raisonnement par analogie, de type "génère-et-teste", basé sur un modèle, qualitatif, par défaut, et autoépistémique. Plusieurs des systèmes de raisonnement/inférence développés et utilisés au fil des ans pour accomplir le raisonnement automatisé dans les systèmes d'ordinateurs sont discutés. L'emphase est mise sur les systèmes logiques, à base de règles, à base de cadre, et à base de cas, puisque ces paradigmes et systèmes sont les plus pertinents pour le développement des systèmes de soutien à l'analyse de la situation basés sur la connaissance. Finalement, les systèmes non-monotones qui permettent la rétraction de faits et les systèmes de maintien de la vérité sont présentés. Le mémoire fournit une description d'ensemble du raisonnement, contribuant ainsi à la mise en place d'un cadre de référence de R&D pour deux projets à R&D pour la défense Canada (11hg, Exploitation collaborative de la connaissance pour l'éveil situationnel du domaine maritime, et 12of, Analyse de la situation pour le commandant d'armée tactique).

This page intentionally left blank.

Executive summary

Reasoning Processes, Methods and Systems for Use in Knowledge-Based Situation Analysis Support Systems

Roy, J., Auger, A.; DRDC Valcartier TM 2006-756; Defence R&D Canada – Valcartier; October 2008.

Over the years, situation awareness has emerged as an important concept supporting dynamic human decision-making in both military and public security environments; situation analysis is defined as the process that provides and maintains a state of situation awareness for the decision makers. Information fusion is a key enabler to meeting the demanding requirements of situation analysis in future command and control (C2) and intelligence support systems.

Adopting a knowledge-centric view of situation analysis and information fusion (SAIF) ultimately requires that one cares about knowledge representation (KR) and reasoning, i.e., the area of artificial intelligence (AI) concerned with how knowledge can be represented symbolically (i.e., using formal symbols to represent a collection of propositions believed by some putative agent) and manipulated in an automated way by reasoning programs. To reason is to use the faculty of reason so as to arrive at conclusions or to discover, formulate, or conclude by the use of reason. Similarly, to infer is "to derive as a conclusion from facts or premises". The terms *reasoning* and *inference* are generally used to cover any process by which conclusions are reached.

This memorandum reviews reasoning processes, methods and systems. The memorandum doesn't present "solutions" or findings and results of completed activities. The intent is more to provide a comprehensive description of reasoning, in order to contribute to the development of a foundational R&D framework for two projects that are just starting under the Applied Research Program (ARP) at Defence R&D Canada: SATAC (Situation Analysis for the Tactical Army Commander), and CKE-4-MDA (Collaborative Knowledge Exploitation for Maritime Domain Awareness). Aspects being discussed include the notion of an inference engine and other related concepts such as logical arguments, inference chains and theorems. Forward chaining and backward chaining are briefly described. Three basic categories of reasoning, i.e., deduction, induction, and abduction are presented, along with other reasoning methods including analogical, generate-and-test, model-based, qualitative, default, and autoepistemic reasoning. Many of the reasoning/inference systems developed and used over the years to achieve automated reasoning in computer systems are discussed. Emphasis is given to logic, rule-based, frame-based, and case-based reasoning systems, as these paradigms and systems are the most relevant to the development of knowledge-based situation analysis support systems. Finally, non-monotonic systems that allow the retraction of facts and truth maintenance systems are presented.

Situation analysis and information fusion play a critical role in the C2 and intelligence processes, and have already received significant attention for military applications. Numerous ongoing projects of the Department of National Defence (DND) and also at Defence R&D Canada possess an important SAIF component. However, despite the importance given to SAIF in many DND strategic documents and projects, the current systems and the associated technology often fail to meet the demanding requirements of the operational decision makers; major science and

technology advances are required to really exploit the full potential of the related enabling technologies and to best serve the operational communities. The effort reported here constitutes one step in this direction; it contributes to the establishment of a solid basis on which a long-term R&D program should be built.

This memorandum only provides an initial overview of the many issues regarding the technologies for automated reasoning; one needs to dig deeper into these issues to really exploit these technologies and develop appropriate support systems meeting the demanding requirements of the operational communities. Hence, R&D activities have been and are still currently being conducted to further investigate knowledge representation concepts, paradigms and techniques, and reasoning processes, methods and systems for use in knowledge-based SAIF support systems. Adopting a knowledge-centric view of SAIF and a corresponding well-structured development process within a holistic approach and framework requires that knowledge (i.e., expertise) is eventually acquired from the subject matter experts (SMEs) of the different military and public security application domains. In this regard, knowledge and ontological engineering techniques have been and are still being investigated at the moment. Knowledge elicitation and validation sessions with SMEs are about to be conducted under the two abovementioned R&D projects (SATAC and CKE-4-MDA).

Sommaire

Reasoning Processes, Methods and Systems for Use in Knowledge-Based Situation Analysis Support Systems

Roy, J., Auger, A.; DRDC Valcartier TM 2006-756; R & D pour la défense Canada – Valcartier; octobre 2008.

Au fil des ans, l'éveil situationnel a émergé en tant que concept important en soutien à la prise de décision dynamique par des humains dans les milieux militaires et de sécurité publique; l'analyse de la situation est définie comme étant le processus qui fournit et maintient un état d'éveil situationnel pour les preneurs de décisions. La fusion d'information constitue un habilitant clé pour répondre aux besoins exigeants de l'analyse de la situation dans les systèmes futurs de soutien au commandement et contrôle (C2) et au renseignement.

Adopter une vue de l'analyse de la situation et de la fusion d'information (ASFI) centrée sur la connaissance requiert ultimement que l'on se soucit de la représentation de la connaissance (RC) et du raisonnement, i.e., l'aire de l'intelligence artificielle (IA) concernée par comment la connaissance peut être représentée symboliquement (i.e., en utilisant des symboles formels pour représenter une collection de propositions crues par un agent présumé) et manipulée de façon automatique par des programmes qui raisonnent. Raisonner, c'est “utiliser la faculté de la raison pour tirer des conclusions” ou “découvrir, formuler, ou conclure en utilisant la raison”. De façon similaire, inférer consiste à “dériver comme conclusion à partir de faits ou de prémisses”. Les termes raisonnement et inférence sont généralement utilisés pour couvrir tout processus par lequel des conclusions sont tirées.

Ce mémoire passe en revue les processus, méthodes et systèmes de raisonnement. Le mémoire ne présente pas de “solutions” ou de découvertes et résultats d'activités complétées. L'intention est plutôt de fournir une description d’ensemble du raisonnement, dans le but de contribuer à la mise en place d'un cadre de base de R&D pour deux projets qui démarrent en ce moment sous le programme de recherches appliquées (PRA) à R&D pour la défense Canada: ASCAT (Analyse de la situation pour le commandant d'armée tactique), et ECCESDM (Exploitation collaborative de la connaissance pour l'éveil situationnel du domaine maritime). Les aspects discutés incluent la notion d'un moteur d'inférence et d'autres concepts reliés, tel que les arguments logiques, les chaînes d'inférence et les théorèmes. Les enchaînements vers l'avant et vers l'arrière sont brièvement décrits. Les trois catégories de base de raisonnement, i.e., déduction, induction, et abduction sont présentées, en plus d'autres méthodes de raisonnement incluant le raisonnement par analogie, de type “génère-et-teste”, basé sur un modèle, qualitatif, par défaut, et autoépistémique. Plusieurs des systèmes de raisonnement/inférence développés et utilisés au fil des ans pour accomplir le raisonnement automatisé dans les systèmes d'ordinateurs sont discutés. L'emphase est mise sur les systèmes logiques, à base de règles, à base de cadre, et à base de cas, puisque ces paradigmes et systèmes sont les plus pertinents pour le développement des systèmes de soutien à l'analyse de la situation basés sur la connaissance. Finalement, les systèmes non-monotones qui permettent la rétraction de faits et les systèmes de maintien de la vérité sont présentés.

L'analyse de la situation et la fusion d'information ont un rôle critique à jouer dans les processus de C2 et de renseignement et elles ont déjà reçu une grande attention pour les applications militaires. De nombreux projets courants du Ministère de la Défense Nationale (MDN) et aussi à R&D pour la défense Canada comportent une importante composante d'ASFI. Cependant, en dépit de l'importance donnée à l'ASFI dans plusieurs documents et projets stratégiques du MDN, les systèmes actuels et la technologie associée n'arrivent souvent pas à rencontrer les besoins exigeants des preneurs de décisions opérationnels; des percées majeures en science et technologie sont requises pour vraiment exploiter le plein potentiel des technologies habilitantes associées et pour servir au mieux les communautés opérationnelles. L'effort rapporté ici est une étape en ce sens; il contribue à l'établissement d'une base solide sur laquelle un programme de R&D à long-terme devrait être construit.

Ce mémoire ne donne qu'une vue d'ensemble initiale de la multitude d'aspects concernant les technologies pour le raisonnement automatisé; il s’avère nécessaire d'approfondir ces aspects pour exploiter vraiment ces technologies et développer des systèmes de soutien appropriés et rencontrant les besoins exigeants des communautés opérationnelles. Pour cette raison, des activités de R&D ont été et sont encore menées afin d’étudier plus à fond les concepts, paradigmes et techniques de représentation de la connaissance, et les processus, méthodes et systèmes de raisonnement pour leur utilisation dans des systèmes de soutien à l'ASFI basés sur la connaissance. Adopter une vue de l'ASFI centrée sur la connaissance ainsi qu'un processus correspondant de développement bien structuré avec une approche et un cadre holistiques nécessite que la connaissance (i.e., l'expertise) soit acquise des experts du domaine (ED) des différents domaines d'application militaires et de sécurité publique. À ce propos, les techniques d'ingénierie de la connaissance et ontologique ont été et sont encore étudiées en ce moment. Des sessions d’élicitation et de validation de connaissances avec des ED sont sur le point d'être effectuées dans le cadre des deux projets de R&D mentionnés précédemment (ASCAT et ECCESDM).

Table of contents

Abstract i
Résumé i
Executive summary iii
Sommaire v
Table of contents vii
List of figures xi
List of tables xii
Important Notice xiii
1. Introduction 1
2. The Reasoning Processes 6
    2.1 Reasoning in Artificial Intelligence 6
    2.2 The Inference Engine 6
    2.3 Pattern Matching 6
    2.4 Theorems and Inference Chains 7
        2.4.1 Forward Chaining (Data-Driven / Data-Directed Inference) 7
        2.4.2 Backward Chaining (Goal-Driven / Goal-Directed Inference) 8
        2.4.3 Comparison Between Backward and Forward Chaining 9
3. Reasoning Methods 10
    3.1 Deductive Reasoning 12
    3.2 Inductive Reasoning 13
        3.2.1 Basic Forms 13
        3.2.2 Other Aspects of Induction (Analogical Reasoning, Learning) 13
        3.2.3 Human Inductive Discovery 14
    3.3 Abductive Reasoning 15
        3.3.1 Creating and Testing Hypotheses 17
        3.3.2 Hypothesis Selection 17
        3.3.3 Abductive Method of Scientific Investigation 18
    3.4 Integrating Deduction, Induction and Abduction with Retroduction 18
    3.5 Analogical Reasoning 20
    3.6 Generate-and-Test Reasoning 20
    3.7 Model-Based Reasoning 21
        3.7.1 Deducing Hidden Properties of the World 21
    3.8 Qualitative Reasoning 22
    3.9 Default Reasoning 22
    3.10 Autoepistemic Reasoning 22
4. Reasoning / Inference Systems 24

4.1 Logic Systems 24
4.1.1 Logical Entailment 24
4.1.1.1 Models and Entailment 26
4.1.1.2 Semantic Interpretation and Formal Inference in Computers 26
4.1.1.3 Knowledge Base (KB) and Entailment 27
4.1.1.4 The Key Point About Logical Entailment/Consequence 27
4.1.2 Logical Inference Mechanism / Procedure 28
4.1.2.1 The Truth-Table Method of Inference 28
4.1.2.2 Soundness / Truth-Preservation 29
4.1.2.3 Completeness 33
4.1.2.4 Decidability 34
4.1.3 Computational Difficulties with Logical Reasoning Procedures 35
4.1.3.1 Introducing Language Restrictions 35
4.1.3.2 Theorem Proving as a Search Problem 38
4.1.4 Logic Programming Systems 39
4.1.5 Theorem Provers 39
4.1.5.1 Theorem Provers Versus Logic Programming Language 39
4.1.5.2 Theorem Provers as Assistants 40
4.1.6 Description Logic Systems 40
4.1.6.1 Subsumption 40
4.1.6.2 Satisfiability 41
4.1.6.3 Subsumption Algorithms 42
4.1.6.4 Tractability of Inference / Subsumption 42
4.1.6.5 Very Expressive Description Logics 44
4.1.6.6 Examples of Description Logic Systems 44
4.2 Rule-Based Systems 44
4.2.1 Forward and Backward Chaining in Rule-Based Systems 45
4.2.2 The Rule Interpreter 46
4.2.3 Inference Cycle 46
4.2.4 Incorrect Rules? 47
4.2.5 Priority Schemes 47
4.2.5.1 Markov Algorithms 47
4.2.6 The Rete Algorithm 47
4.2.7 Conflict Resolution 48
4.2.8 Refraction 48
4.2.9 Top-Level (Command Interpreter & System Development) 48
4.2.10 The Inference Tree of a Rule-Based System 48
4.2.11 Cognitive Architectures 49
4.2.12 A Final Remark on Rule-Based Reasoning 49
4.3 Frame-Based Systems 49
4.4 Case-Based Reasoning Systems 51

4.4.1 The Process of Case-Based Reasoning....51
4.4.2 Comparison of Case-Based and Rule-Based Reasoning....53
4.4.3 Advantages of Case-Based Reasoning....54
4.4.4 Case-Based Reasoning Uses and Issues....54
4.4.5 Success Factors for a Case-Based Reasoning System....55
5. Non-Monotonic Reasoning....56
5.1 Retractions and Defeasibility....56
5.2 Truth-Maintenance Systems (TMS)....57
5.2.1 Providing Explanations of Propositions in a TMS....58
5.2.2 Dealing with Inconsistencies in a TMSs....58
5.2.3 Justification-Based Truth Maintenance Systems (JTMS)....58
5.2.4 Assumption-Based Truth Maintenance Systems (ATMS)....58
5.2.5 TMS Computational Complexity....59
6. Conclusion....60
References....61
List of symbols/abbreviations/acronyms/initialisms....63

This page intentionally left blank.

List of figures

Figure 1: Knowledge representation and reasoning in KBSs (first-order logic perspective) 3
Figure 2: Graphical representation of discovery [Waltz, 2003].....14
Figure 3: Integrating the basic reasoning flows [Waltz, 2003].....19
Figure 4: The semantics provides the connection between sentences and facts [Russell, Norvig, 1995] 25
Figure 5: Case-based reasoning flow chart [Turban, Aronson, 1998].....52

List of tables

Table 1: Some methods of reasoning/inference [Giarratano, Riley, 1998] [Turban, Aronson, 1998] 10

Table 2: Summary of the purpose of forward chaining, backward chaining, and abduction [Giarratano, Riley, 1998] 16

Table 3: Hypothesis evaluation criteria [Waltz, 2003] 17

Table 4: Tasks of a typical inference cycle [Giarratano, Riley, 1998] 46

Table 5: Comparison of case-based and rule-based reasoning [Turban, Aronson, 1998] 53

Important Notice

The work resulting in the present technical memorandum began during the summer of 2005, when the authors were about to undertake new research projects involving many aspects of artificial intelligence (AI), knowledge acquisition and representation, automated reasoning, knowledge and ontological engineering, and other related notions. The level of expertise of the people to be involved in the projects regarding these aspects was somewhat uneven, and the need was felt to produce some sort of "primer" document to help the project participants to get up to speed, and to establish some common background (or foundation) for the projects. Along this line of thoughts, available books were quickly identified and gathered, all from well established authors in these fields (Brachman, Levesque, Russell, Norvig, Stefik, Giarratano, Riley, Ginsberg, Turban, Aronson, Gómez-Pérez, Fernández-López, Corcho). Of course, there are many other authors and references that could have been used, but the set was limited to the ones listed here, given that it was felt sufficient to fulfill our objective. Intense reading and animated discussion sessions with colleagues at DRDC Valcartier then occurred, while very detailed notes were taken, and synthesis diagrams created; some of these notes and diagrams compose the essence of this document.

It is very important to note that many text passages in this memorandum have been taken up integrally from the original documents of the authors mentioned above. This was done, on purpose, to assure integrity of the words and ideas as the author(s) meant them. In part this is because of our own limitations in these subjects, and in part because their words are both adequate and terse. Also, the many literal quotations from the original text have been inserted in this memorandum without using quotation marks ("") to highlight the direct quotations, as is normally required. This was done to avoid making the text too busy and cluttered. However, all of the material used is referenced to the appropriate original author(s) in the body of the text, mostly from paragraph to paragraph, but sometime from sentence to sentence. It is thus easy for the reader (and actually recommended) to go back to the original book for more details, or to put the material back in its original context.

All of the authors mentioned above cover a lot of ground in their respective books. The synthesis process for this memorandum was thus an attempt to extract from each one only the aspects judged relevant to our R&D projects, and then to merge the individual ideas into a unified smooth and fluid text. Unfortunately, the attempt was more or less successful, resulting in some mix of styles, which could be annoying to some readers.

Some may fairly note that most of the reference documents are not "recent" documents. Certainly, the current strong interest in ontologies and other related semantic web technologies for example has produced many recent documents that could be used for our purpose. Things have evolved since the summer of 2005, and they still progress at a very fast pace. This being said, although some of the semantic web technologies may look fresh and new, many of the proposed concepts are actually brought back and resurrected from the huge body of existing work in AI. Some of this recent material can in fact be considered as a "variation on a (somewhat old) theme". It may thus be beneficial to go back to the basics of AI first, in order to better appreciate later the more recent advancements that constitute the state-of-the-art at the moment. Moreover, although some aspects, such as expert systems, may rightfully be considered very mature, they

have not yet been fully exploited in the context of higher-level information fusion in knowledge-based situation analysis support systems (KB SASS).

This technical memorandum was not intended as a contribution to the advancement of the AI domain. Furthermore, no explicit and/or firm recommendation is made on the best approaches to build a KB SASS. Such recommendations are yet to come as a result of the current projects being undertaken in specific applied problem areas of interest to our sponsors (e.g., maritime domain awareness, geo-intelligence, situation analysis for the tactical Army commanders). The discussion is more at the general level, directly based on the established references that were selected.

Writing this document has certainly helped the authors to get up to speed regarding the aspects discussed; it is believed (hoped) that it could similarly help other newcomers.

1. Introduction

Situation awareness (SAW) has emerged as an important concept supporting dynamic human decision-making in military and public security environments. Actually, SAW is a general concept that has been shown to be of interest in a very large number of settings. Given this wide interest for SAW, the author has previously proposed another concept, situation analysis (SA), defined as a process that provides and maintains a state of situation awareness for the decision maker(s) [Roy, 2001].

A key enabler to meeting the demanding requirements of situation analysis in future command and control (C2) and intelligence systems (i.e., in achieving high-quality situation awareness for optimal decision making) is data/information fusion. An initial lexicon defined data fusion as a process dealing with the association, correlation, and combination of data and information from single and multiple sources to achieve refined position and identity estimates, and complete and timely assessments of situations and threats as well as their significance [White, 1987]. This definition has evolved over the years, and multiple variants have been proposed. Recently, [Lambert, 2001] defined information fusion as the process of utilizing one or more information sources over time to assemble a representation of aspects of interest in an environment. Among the many reasons for interest in this technology, there is that data/information fusion:

- provides extended spatial and temporal coverage, increased confidence, reduced ambiguity, improved entity detection, etc.,
- allows for the management of large volumes of information and the correlation of seemingly unrelated, overlooked, or deceptive information to present a coherent representation of an evolving situation to a decision maker, and,
- enables the commander to cope with the complexity and tempo of operations in modern dynamic operational theatres.

Clearly, situation analysis and information fusion (SAIF) have a critical role to play, and have already received significant attention for military applications. Numerous ongoing projects of the Department of National Defence (DND) have an important SAIF component. Examples, just to name a few, are:

- Project No. 00000276: Land Force Intelligence, Surveillance, Target Acquisition and Reconnaissance (LF ISTAR)
- Project No. 00000806: Marine Security Operations Centres (MSOC)
- Project No. 00000624: Joint Information and Intelligence Fusion Capability (JIIFC)

However, despite the importance surrounding SAIF in many DND strategic documents and projects, the current supporting systems and the associated technology often fail to meet the demanding requirements of the operational decision makers, and major Science and Technology (S&T) advancements are required to really achieve the full potential of the related enabling technologies and to best serve the operational communities. In this line of thought, many R&D projects ongoing under the Technology Demonstration Program (TDP) at Defence R&D Canada (DRDC) have an important SAIF component as well:

- Joint Command Decision Support for the 21st Century (JCDS 21)
- Innovative Naval Combat Management Decision Support (INCOMMANDS)

SAIF is certainly expected to play a crucial role in the next generation of support systems for aiding decision makers in military and public security operations.

Taking into account the context described above, [Roy, 2007-A] proposed that developing and adopting a knowledge-centric view of situation analysis should provide a more holistic perspective of this process, leading to the development of better, more adequate SAIF support systems for the operational communities. This was mostly based on the fact that *awareness* ultimately has to do with *having knowledge of something.* As discussed in [Roy, 2007-A], expert systems constitute a branch of artificial intelligence (AI) that makes extensive use of specialized knowledge to solve problems at the level of a human expert. They are the paradigmatic application of AI techniques to hard problems. Although traditional expert systems have shown limitations, expert system technologies are expected to significantly contribute to the development of the future, state-of-the-art knowledge-based situation analysis support systems (SASSs).

Adopting the knowledge-centric view of situation analysis eventually requires that one cares about knowledge representation (KR) and reasoning, i.e., the area of artificial intelligence (AI) concerned with how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs [Brachman, Levesque, 2004]. According to the dictionary [Merriam-Webster, 2003], to *reason* is "to use the faculty of reason so as to arrive at conclusions" or "to discover, formulate, or conclude by the use of reason". Similarly, to *infer* is "to derive as a conclusion from facts or premises". The terms *reasoning* and *inference* are generally used to cover any process by which conclusions are reached. The term *inference* is generally used for mechanical systems such as expert systems, while *reasoning* is generally used in human thinking.

In general, reasoning in knowledge-based systems is the formal manipulation of symbols representing a collection of believed propositions to produce representations of new ones [Brachman, Levesque, 2004]. The symbols must be concrete enough that one can manipulate them (move them around, take them apart, copy them, string them together) in such a way as to construct representations of new propositions. Reasoning is a form of calculation, not unlike arithmetic, but over symbols standing for propositions rather than numbers.

To see the motivation behind reasoning in a knowledge-based system, it suffices to observe that one would like action to depend on what such a system believes about the world, as opposed to just what the system has explicitly represented [Brachman, Levesque, 2004]. In general, much of what one expects to put in a knowledge base (KB) (cf. Fig. 1, subsection 4.1.1.3, and [Roy, Auger, 2008-B]) will involve quite general facts, which will then need to be applied to particular situations. One doesn't want to condition behaviour only on the represented facts that one is able to retrieve, like in a database system. The beliefs of the system must go beyond these.

Figure 1, a very busy diagram, illustrates many aspects of knowledge representation and reasoning in knowledge-based systems, from the perspective of first-order logic (FOL).

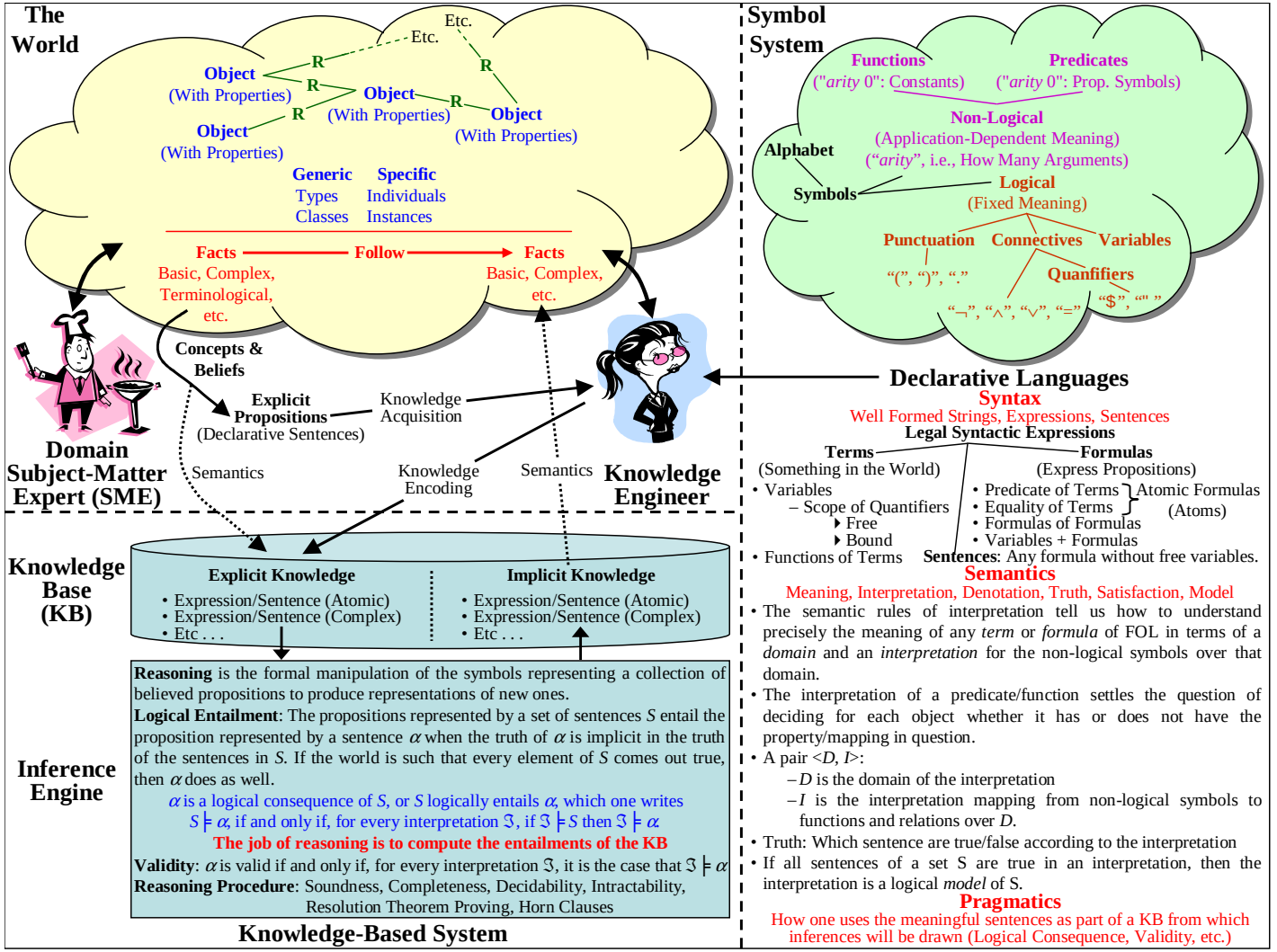


Figure 1: Knowledge representation and reasoning in KBSs (first-order logic perspective)

Knowledge representation should support the tasks of acquiring and retrieving knowledge, as well as subsequent reasoning [Turban, Aronson, 1998]. Actually, as illustrated in Fig. 1, there is an interplay between representation and reasoning; it is not enough to write down what needs to be known in some formal representation language, nor is it enough to develop reasoning procedures that are effective for various tasks [Brachman, Levesque, 2004]. Knowledge representation and reasoning is best understood as the study of how knowledge can at the same time be represented as comprehensively as possible and be reasoned with as effectively as possible. There is a trade off between these two concerns. Needing to reason with knowledge structures has an impact on the form and scope of the languages used to represent a system's knowledge.

Although there is this strong link between KR and reasoning, the discussion of these two aspects has been split in two documents. This memorandum reviews reasoning processes, methods and systems for use in knowledge-based SASSs, while [Roy, Auger, 2008-A] discusses knowledge representation concepts, paradigms and techniques for use in such systems. The memorandum doesn't present “solutions”, or findings and results of completed activities. The intent is more to provide a fair description of reasoning, in order to contribute to the set up of a foundational R&D framework for two DRDC projects that are just starting under the Applied Research Program (ARP):

- SATAC (Situation Analysis for the Tactical Army Commander), and,
- CKE-4-MDA (Collaborative Knowledge Exploitation for Maritime Domain Awareness), formerly known as AKAMIA (Advanced Knowledge Acquisition for Maritime Information Awareness).

The memorandum is organized as follows. Section 2 begins the discussion on reasoning with some definitions, and then presents the notion of an inference engine. Other related concepts, including logical arguments, inference chains and theorems are also introduced. The two main general methods of inference that are commonly used as problem-solving strategies, i.e., forward and backward chaining, are briefly described.

There are several ways people reason and solve problems. The most basic taxonomy of inferential reasoning processes distinguishes three basic categories of reasoning, i.e., deduction, induction, and abduction. These are discussed in Section 3, along with other reasoning methods including analogical, generate-and-test, model-based, qualitative, default, and autoepistemic reasoning. The integration of deduction, induction and abduction with retroduction is an important aspect also discussed in Section 3.

Many reasoning/inference systems have been developed and used over the years to achieve automated reasoning in computer systems. Many of these are discussed in Section 4. Emphasis is given to logic, rule-based, frame-based, and case-based reasoning systems, as these paradigms and systems are the most relevant to the development of knowledge-based SASSs.

A knowledge base (cf. [Roy, Auger, 2008-B]) is not of much use to an agent unless it can be expanded. Normally, the addition of new axioms to a logic system means that more theorems can be proved because there are more axioms from which theorems can be derived. This property of increasing theorems with increasing axioms is known as monotonicity. This being said, most logical reasoning systems, regardless of their implementation, also have to deal with retractions. Unfortunately, monotonic systems cannot deal with changes in the truth of axioms and theorems;

a non-monotonic system allows the retraction of facts. These issues are discussed in Section 5. In particular, the truth maintenance problem, i.e., maintaining the correctness or truth of a system when one retracts a proposition, and also the process of keeping track of which additional propositions need to be retracted, is discussed.

Finally, concluding remarks are provided in Section 6.

2. The Reasoning Processes

As previously introduced and according to [Merriam-Webster, 2003], to reason is:

- To use the faculty of reason so as to arrive at conclusions.
- To discover, formulate, or conclude by the use of reason.

Similarly, to infer is to derive as a conclusion from facts or premises. The terms *reasoning* and *inference* are generally used to cover any process by which conclusions are reached [Russell, Norvig, 1995]. The term *inference* is generally used for mechanical systems such as expert systems [Giarratano, Riley, 1998]. Reasoning is generally used in human thinking.

A belief is a conviction of the truth of some statement, especially when based on examination of evidence [Merriam-Webster, 2003]. Reasoning/inference processes analyze evidence and synthesize explanations [Waltz, 2003]. They create, manipulate, evaluate, modify, and assert beliefs.

2.1 Reasoning in Artificial Intelligence

Artificial intelligence (AI) involves an attempt by machines to exhibit reasoning capabilities [Turban, Aronson, 1998]. The reasoning consists of making inferences from facts and rules using heuristics or other search approaches.

2.2 The Inference Engine

Once the knowledge representation in the knowledge base (cf. Fig. 1, subsection 4.1.1.3, and [Roy, Auger, 2008-B]) is completed, or is at least at a sufficiently high level of accuracy, it is ready to be used [Turban, Aronson, 1998]. One needs a computer program to access the knowledge for making inferences. This program is an algorithm that controls the reasoning process. It is usually called the inference engine or the control program. In rule-based systems, it is also called the rule interpreter.

The inference engine directs the search through the knowledge base [Turban, Aronson, 1998]. The process may involve the application of inference rules in what is called pattern matching. The control program decides which rule to investigate, which alternative to eliminate, and which attribute to match.

2.3 Pattern Matching

AI is unique in that it makes inferences by using a pattern-matching approach [Turban, Aronson, 1998]. There are two main variations on pattern matching, notably one-way pattern matching and two-way pattern matching (unification) [Stefik, 1995].

In one-way pattern matching, a pattern with variables is matched against an expression containing constants [Stefik, 1995]. At the end of the operation, the variables in the pattern may take on the

values corresponding to parts of the constant structure. In unification, patterns with variables are matched against other patterns with variables. Variables in both patterns may take on values at the end of the matching operation.

A key element in the syntax of pattern matching is the specification of pattern variables [Stefik, 1995].

2.4 Theorems and Inference Chains

A group of multiple inferences that connects a problem with its solution is called a chain [Giarratano, Riley, 1998]. A logical argument is a group of statements in which the last is claimed to be justified on the basis of the previous ones in the chain of reasoning. A theorem is the conclusion of a valid argument. The conclusion of an inference chain is a theorem because it is proved by the chain of inference. Systems that use an inference chain to establish a conclusion are really using theorems.

Typically, inference rules can be used in two ways [Russell, Norvig, 1995]. The question of forward versus backward search (in AI) has an obvious analog in an inferential setting [Ginsberg, 1993]. As with search, there are meta-level decisions to be made in theorem proving; the question of whether to work forward or backward is one of them [Ginsberg, 1993].

Two general methods of inference are thus commonly used as problem-solving strategies: forward chaining and backward chaining [Giarratano, Riley, 1998]. It is helpful to visualize forward and backward chaining in terms of a path through a problem space in which the intermediate states correspond to intermediate hypotheses under backward chaining, or intermediate conclusions under forward chaining.

2.4.1 Forward Chaining (Data-Driven / Data-Directed Inference)

An inference chain that is searched or traversed from a problem to its solution is called a forward chain [Giarratano, Riley, 1998]. Forward chaining is reasoning from facts to the conclusions resulting from those facts. On can start with the sentences in the knowledge base and generate new conclusions that in turn can allow more inferences to be made [Russell, Norvig, 1995].

Actually, one starts from available information as it becomes available, or from a basic idea, and then tries to draw conclusions [Turban, Aronson, 1998]. Forward chaining is usually used when a new fact is added to the knowledge base (i.e., it is triggered by the addition of a new fact to the knowledge base) and one wants to generate its consequences [Russell, Norvig, 1995]. The idea is to find all implications that have the new fact as a premise; then if the other premises are already known to hold, one can add the consequent of the implication to the knowledge base, triggering further inference.

The forward-chaining procedure makes use of the idea of a renaming, and the idea of a composition of substitutions [Russell, Norvig, 1995].

By custom, higher-level constructs that are composed of lower-level ones are put at the top [Giarratano, Riley, 1998]. Forward chaining is called bottom-up reasoning because it reasons

from the low-level evidence, facts, to the top-level conclusions that are based on the facts. It builds up a picture of the situation gradually as new data comes in [Russell, Norvig, 1995]. Its inference processes are not directed toward solving any particular problem; for this reason it is called a data-driven or data-directed procedure.

There are no queries in a forward-chaining approach [Russell, Norvig, 1995]. Instead, inference rules are applied to the knowledge base, yielding new assertions. This process repeats forever, or until some stopping criterion is met.

2.4.2 Backward Chaining (Goal-Driven / Goal-Directed Inference)

It may happen that forward chaining will generate many conclusions; for some situation-dependent reasons, a lot of these can potentially be irrelevant to the main problem of current interest. In such cases, it is often better to use backward chaining, which directs all its effort toward the question at hand [Russell, Norvig, 1995]. That is, alternatively, one can start with something one wants to prove, find implication sentences that would allow him/her to conclude it, and then attempt to establish their premises in turn. This is called backward chaining, because it uses the inference rules backwards. Backward chaining thus involves “reasoning in reverse”.

As stated above, backward chaining is normally used when there is a goal to be proved [Russell, Norvig, 1995]. That is, another way of describing a backward chain is in terms of a goal which can be accomplished by satisfying sub-goals [Giarratano, Riley, 1998]. Typically, backward chaining proceeds by defining smaller sub-goals that must be satisfied if the initial goal is to be satisfied. These sub-goals are then further broken down into smaller sub-goals, and so forth. Hence, one of the advantages of backward chaining systems is that execution can proceed in parallel, i.e., if multiple processors were available, they could work on satisfying sub-goals simultaneously.

In view of the preceding discussion, backward chaining is called a goal-driven or goal-directed procedure, in which one starts from an expectation (hypothesis), then seek evidence that supports (or contradicts) the expectation [Turban, Aronson, 1998]. A hypothesis can be viewed as a fact whose truth is in doubt and needs to be established, a potential conclusion to be proved [Giarratano, Riley, 1998]. An inference chain that is traversed from a hypothesis back to the evidence that support the hypothesis is a backward chain. Often, this entails formulating and testing intermediate hypotheses (or sub-hypotheses) [Turban, Aronson, 1998]. Reasoning from the higher-level constructs such as hypotheses down to the lower-level facts which may support the hypotheses is called top-down reasoning [Giarratano, Riley, 1998].

Backward chaining is designed to find all answers to a question posed to the knowledge base [Russell, Norvig, 1995]. Given a query, it searches for a constructive proof that establishes some substitution that satisfies the query. The backward-chaining algorithm works by first checking to see if answers can be provided directly from sentences in the knowledge base. It then finds all implications whose conclusion unifies with the query, and tries to establish the premises of those implications, also by backward chaining. If the premise is a conjunction, then the procedure processes the conjunction conjunct by conjunct, building up the unifier for the whole premise as it goes.

2.4.3 Comparison Between Backward and Forward Chaining

Which one is better? The answer depends on the purpose of the reasoning and the shape of the search space [Turban, Aronson, 1998]. For example, if the goal is to discover all that can be deduced from a given set of facts, the system should run forward. In some cases, the two strategies can be mixed (bidirectional).

3. Reasoning Methods

Several methods can direct search and reasoning [Turban, Aronson, 1998]. It is interesting to examine how people reason, which is what AI attempts to mimic. There are several ways people reason and solve problems. An interesting view of the problem-solving process is one in which people draw on "sources of power" [Turban, Aronson, 1998]:

- Formal reasoning methods (such as logical deduction).
- Heuristic reasoning, or IF-THEN rules.
- Focus, or common sense applied to specific goals.
- Divide and conquer, or dividing complex problems into sub-problems (sometimes called chunking).
- Parallelism, e.g., neural processors (perhaps a million) operating in parallel.
- Representation, or ways of organizing pieces of information.
- Analogy, or the ability to associate and relate concepts.
- Synergy, in which the whole is greater than the sum of its parts.
- Serendipity, or fortuitous accidents.

These sources of power range from the purely deductive reasoning best handled by computer systems to inductive reasoning that is more difficult to computerize [Turban, Aronson, 1998]. Some believe that the future of AI lies in finding ways to tap sources that have only begun to be exploited. These sources of power can be translated into specific reasoning or inference methods.

The most basic taxonomy of inferential reasoning processes distinguishes three basic categories of reasoning [Waltz, 2003]:

- Deduction
- Induction
- Abduction

It is worth noting that while induction and deduction are the classical formal reasoning forms found in most philosophy and logic texts, abduction is a more recent pragmatic form of reasoning that is the less formal but more common approach of inference to achieve the best explanation with uncertain evidence [Waltz, 2003]. In addition to providing brief descriptions of the three basic categories of reasoning, Table 1 lists some other specific reasoning or inference methods.

Table 1: Some methods of reasoning/inference [Giarratano, Riley, 1998] [Turban, Aronson, 1998]

Method	Description
Deductive reasoning	Logical reasoning in which conclusions must follow from their premises. Move from a general principle to a specific inference.

Method	Description
	General principle is composed of premises.
Inductive reasoning	Inference from the specific case to the general. Move from some established facts to draw general conclusions.
Abductive reasoning	Reasoning back from a true conclusion to the premises that may have caused the conclusion.
Analogical reasoning	Inferring a conclusion based on the similarities to another situation. Derive answer to a question by known analogy. It is a verbalization of an internalized learning process. Use of similar, past experiences.
Generate-and-test reasoning	Trial and error. Often used with planning for efficiency (it is then called the plan-generate-test approach).
Model-based reasoning	Reasoning based on knowledge of the structure and behaviour of devices.
Qualitative reasoning	Making inferences using general, physical knowledge about the world. Making inference from problem statements having much less information than is usually known in traditional mathematical formulations.
Default reasoning	In the absence of specific knowledge, assume general or common knowledge by default.
Autoepistemic reasoning	Reasoning about one's own knowledge, as distinct from knowledge in general.
Formal reasoning	Syntactic manipulation of data structure to deduce new facts following prescribed rules of inference (such as predicate calculus).
Meta-level reasoning	Knowledge about what one knows (for example, about the importance and relevance of certain facts and rules).
Heuristics reasoning	Rules of thumb based on experience.
Intuition reasoning	No proven theory. The answer just appears, possibly by unconsciously recognizing an underlying pattern. In general, expert systems do not implement this type of inference. Artificial neural systems (ANSs) may hold promise for this type of inference since they can extrapolate from their training rather than just provide a conditioned response or interpolation. That is, a neural net will always give its best guess for a solution.
Non-monotonic reasoning	Previous knowledge may be incorrect when new evidence is obtained.

Other methods used for more specific needs may include [Giarratano, Riley, 1998]:

- means-ends analysis,
- problem reduction,

- backtracking,
- hierarchical planning and the least commitment principle, and,
- constraint handling.

Although not explicitly listed in Table 1, commonsense knowledge may be a combination of any of these types [Giarratano, Riley, 1998]. Commonsense reasoning is what people use in ordinary situations, and is very difficult for computers to master.

One can characterize the most fundamental inference processes by their process and products [Waltz, 2003]:

- Process. The direction of the inference process refers to the way in which beliefs are asserted. The process may move from specific (or particular) beliefs toward more general beliefs, or from general beliefs to assert more specific beliefs.
- Products. The certainty associated with an inference distinguishes two categories of results of inference. The asserted beliefs that result from inference may be infallible (e.g., an analytic conclusion is derived from infallible beliefs and infallible logic is certain) or fallible judgments (e.g., a synthesized judgment is asserted with a measure of uncertainty; "probably true", "true with 0.95 probability", or "more likely true than false").

Neither analogy, nor generate and test, nor abduction is deductive, and guaranteed to work all the time [Giarratano, Riley, 1998]. From true premises, these methods cannot prove true conclusions. However, these techniques are useful in reducing the search space by generating reasonable hypotheses that can then be used with deduction.

3.1 Deductive Reasoning

One of the most frequently used method of drawing inferences is deductive logic, which has been used since ancient times to determine the validity of an argument [Giarratano, Riley, 1998]. Deduction is reasoning about premises to derive conclusions [Waltz, 2003]. That is, given some facts, a conclusion that follows is inferred [Giarratano, Riley, 1998]. Deduction is the method of inference by which a conclusion is inferred by applying the rules of a logical system to manipulate statements of belief to form new logically consistent statements of belief [Waltz, 2003]. In an argument, the premises are used as evidence to support the conclusions [Giarratano, Riley, 1998]. The premises are also called the antecedent and the conclusion is called the consequent.

The essential characteristic of deductive logic is that the true conclusion must follow from true premises [Giarratano, Riley, 1998]. This form of inference is infallible, in that the conclusion (belief) must be as certain as the premise (belief). It is belief preserving in that conclusions reveal no more than that expressed in the original premises [Waltz, 2003].

Texts on formal logic present the variety of logical systems that may be defined to provide foundations for deductive inference [Waltz, 2003]. One type of logical argument is the syllogism [Giarratano, Riley, 1998]; this is a deductive scheme of a formal argument consisting of a major and a minor premise and a conclusion (as in "every virtue is laudable; kindness is a virtue;

therefore kindness is laudable") [Merriam-Webster, 2003]. In general, a syllogism is any valid deductive argument having premises and a conclusion [Giarratano, Riley, 1998].

The main advantage of studying syllogisms is that it is a simple, well-understood branch of logic that can be completely proven. Deduction can be expressed in a variety of syllogisms, including the more common forms of propositional logic [Waltz, 2003]. Actually, the classical propositional logic system (or calculus) is the basic deductive tool of formal logic; predicate calculus is the system of mathematics that extends deductive principles to the quantitative realm. As an example, modus ponens, the form of deduction most commonly applied, states that:

If A is true, then B is also true; A is true, therefore B is true.

Finally, as shown above, syllogisms are often useful because they can be expressed in terms of IF . . . THEN rules.

3.2 Inductive Reasoning

Induction is the method of inference by which a more general or more abstract belief is developed by observing a limited set of observations or instances [Waltz, 2003]. Induction moves from specific beliefs about instances to general beliefs about larger and future populations of instances.

3.2.1 Basic Forms

The form of induction most commonly applied to extend belief from a sample of instances to a larger population is inductive generalization:

All observed As are Bs; therefore, all As are Bs.

By this method, analysts extend the observations about a limited number of targets (e.g., observations of the money laundering tactics of several narcotics rings within a drug cartel) to a larger target population (e.g., the entire drug cartel) [Waltz, 2003]. Inductive prediction extends belief from a population to a specific future sample:

All observed As are Bs; therefore, the next observed A will be a B.

By this method, an analyst may use several observations of behaviour (e.g., the repeated surveillance behaviour of a foreign intelligence unit) to create a general detection template to be used to detect future surveillance activities by that or other such units [Waltz, 2003]. The induction presumes future behaviour will follow past patterns.

3.2.2 Other Aspects of Induction (Analogical Reasoning, Learning)

In addition to these forms, induction can provide a means of analogical reasoning (induction on the basis of analogy or similarity) and inference to relate cause and effects [Waltz, 2003]. The basic scientific method applies the principles of induction to develop hypotheses and theories that can subsequently be tested by experimentation over a larger population or over future periods of

time. The subject of induction is central to the challenge of developing automated systems that generalize and learn by inducing patterns and processes (rules).

3.2.3 Human Inductive Discovery

The essence of induction is the recognition of a more abstract or general pattern of relationships or behaviours that explains a set of data or observations. The essence of human inductive discovery can be observed in three common forms [Waltz, 2003]:

1. *Aha!* This is the exclamation at the point of scientific study in which the scientist rapidly discerns a new insight or principle, i.e., a discovery.
2. *Ha ha!* This is the response of laughter to the sudden recognition of irony as the punch line of a joke is told; the hearer realizes the alternative and parallel (but hidden) explanation for the story leading up to the line.
3. *Ahhh...* This expression is the appreciation of the higher, aesthetic beauty of a work of art by realizing the more abstract pattern of meaning not found merely in the details of the artwork, but in the holistic meaning conveyed by the external visual image.

In each of these three cases, the underlying inductive act is the sudden discovery of a new or novel pattern, previously hidden in the details, to the discoverer.

The process can be graphically illustrated in a geometric analogy [Waltz, 2003]. Consider the *ha ha* discovery process by representing the story line of a humorous story (Fig. 2).

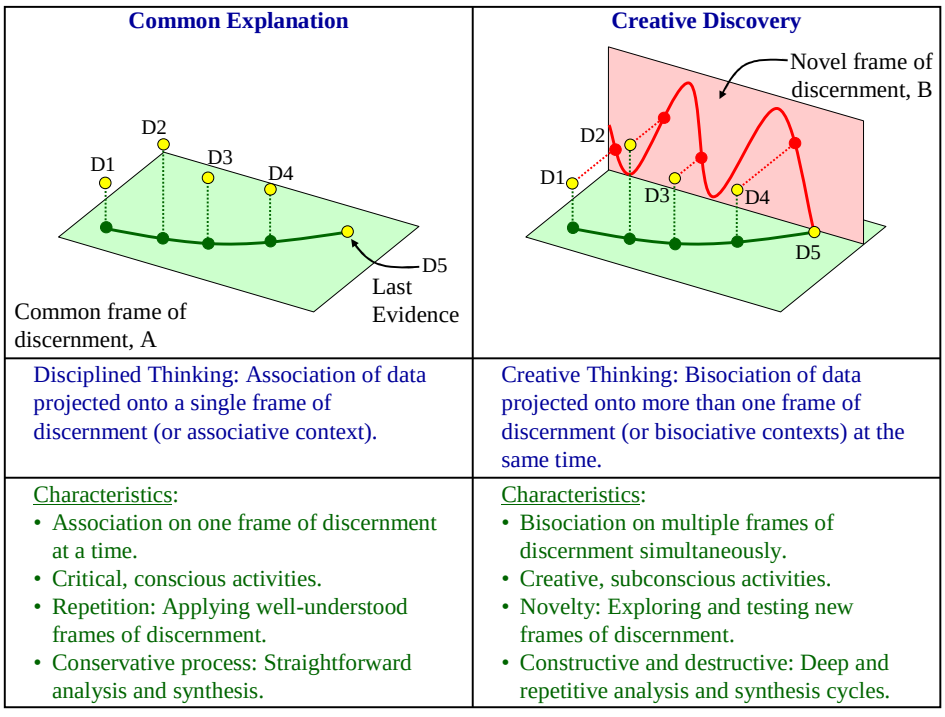


Figure 2: Graphical representation of discovery [Waltz, 2003]

The sequence of evidence is the series of facts (D1, D2, . . .) sequentially presented in the story [Waltz, 2003]. These facts are projected onto an immediate or common explanation, i.e., a frame of discernment to interpret the facts, represented as the plane, A. But, as the punch line (the last piece of evidence, D5) is introduced, it suddenly reveals an entirely different plane in which all of the evidence perfectly fit, revealing a hidden but parallel and ironic explanation for the story. In the geometric analogy, a sinusoid is revealed to fit the data. The cognitive-emotive reaction to the sudden realization is to exclaim "ha ha!".

The term *bisociation* is used to describe the process of viewing multiple explanations (or multiple associations) of the same data simultaneously [Waltz, 2003]. In the example of Fig. 2, the data can be projected onto a common plane of discernment in which the data represents a simple curved line; projected onto an orthogonal plane, the data can explain a sinusoid. Though undersampled, as much intelligence data is, the sinusoid represents a new and novel explanation that may remain hidden if the analyst does not explore more than the common, immediate, or simple interpretation.

In a similar sense, the inductive discovery by an intelligence analyst (aha!) may take on many different forms, following the simple geometric metaphor. For example [Waltz, 2003]:

- A subtle and unique correlation between the timing of communications (by traffic analyst) and money transfers of a trading firm may lead to the discovery of an organized crime operation.
- A single anomalous measurement may reveal a pattern of denial and deception to cover the true activities at a manufacturing facility in which many points of evidence are, in fact, deceptive data "fed" by the deceiver. Only a single piece of anomalous evidence (D5 in Fig. 2) is the clue that reveals the existence of the true operations (a new plane in Fig. 2). The discovery of this new plane will cause the analyst to search for additional supporting evidence to support the deception hypothesis.

Each frame of discernment is a framework for creating a single or a family of multiple hypotheses to explain the evidence [Waltz, 2003]. The creative analyst is able to entertain multiple frames of discernment, alternatively analyzing possible "fits" and constructing new explanations, exploring the many alternative explanations. This is the constructive-destructive process of discovery.

3.3 Abductive Reasoning

Inference by abduction is another method that is commonly used in diagnostic problem solving [Giarratano, Riley, 1998]. Abduction is the informal or pragmatic mode of reasoning to describe how one "reasons to the best explanation" in everyday life [Waltz, 2003]. It is sometimes referred to as reasoning from observed facts to the best explanation. It is the practical description of the interactive use of analysis and synthesis to arrive at a solution or explanation, creating and evaluating multiple hypotheses. Abduction incorporates both inductive (hypothesis-creating) and deductive (hypothesis-testing) operations.

The reasoning process is expressed as a pragmatic syllogism in the following form [Waltz, 2003]:

D is a collection of data. Hypotheses H_1, H_2, . . ., H_n all can explain D. H_k explains D best. Therefore accept hypothesis H_k as the best explanation.

Of course, the critical stage of abduction is the judgment that H_k is the best explanation [Waltz, 2003]. The process requires a criteria for ranking hypotheses, a method for judging which is best, and a method to assure that the set of candidate hypotheses cover all possible (or feasible) explanations.

The schema of abduction resembles modus ponens (cf. subsection 4.1.2.2.1), but is actually very different [Giarratano, Riley, 1998]:

Abduction	Modus Ponens
$p \Rightarrow q$	$p \Rightarrow q$
q	p
p	q

Abduction is another name for a fallacious argument known as the "Fallacy of the Converse".

A backward chain of abduction is not the same as the customary meaning of backward chaining [Giarratano, Riley, 1998]. The term backward chaining means that one is trying to prove a hypothesis by looking for evidence to support it. Table 2 summarizes the purpose of forward chaining, backward chaining, and abduction.

Table 2: Summary of the purpose of forward chaining, backward chaining, and abduction [Giarratano, Riley, 1998]

Inference	Start	Purpose
Forward chaining	Facts	Conclusions that must follow
Backward chaining	Uncertain conclusion	Facts to support the conclusion
Abduction	True conclusion	Facts that may follow

Unlike infallible deduction, abduction is fallible because it is subject to errors (there may be other hypotheses not considered or another hypothesis, however unlikely, may be correct) [Waltz, 2003]. But unlike deduction, it has the ability to extend belief beyond the original premises. This is the logic of discovery and is a formal model of the process that scientists apply all the time.

Although abduction is not a valid deductive argument, it is a useful method of inference and has been used in expert systems [Giarratano, Riley, 1998]. Abduction may be useful as a heuristic rule of inference. That is, when one has no deductive method of inference, abduction may prove useful, but is not guaranteed to work.

3.3.1 Creating and Testing Hypotheses

Abduction introduces the competition among multiple hypotheses, each being an attempt to explain the evidence available. These alternative hypotheses can be compared, or competed on the basis of how well they explain (or fit) the evidence. Furthermore, the created alternative hypotheses provide a means of identifying three categories of evidence important to explanation [Waltz, 2003]:

- Positive evidence. This is evidence revealing the presence of an object or occurrence of an event in a hypothesis.
- Missing evidence. Some hypotheses may fit the available evidence, but the hypothesis "predicts" additional evidence that should exist if the hypothesis were true; such additional evidence is "missing". Subsequent searches and testing for this evidence may confirm or disconfirm the hypothesis.
- Negative evidence. Evidence of a non-occurrence of an event (or non-existence of an object) may confirm a hypothesis.

This process inherently demands a search for alternative hypotheses that extend beyond the hard evidence available.

3.3.2 Hypothesis Selection

Abduction also poses the issue of defining which hypothesis provides the best explanation of the evidence. The criteria for comparing hypotheses, at the most fundamental level, can be based on two principle approaches established by philosophers for evaluating truth propositions about objective reality [Waltz, 2003]:

- The correspondence theory of the truth of a proposition "p is true" is to maintain that "p corresponds to the facts". For the intelligence analyst this would equate to "hypothesis *h* corresponds to the evidence", i.e., it explains all of the pieces of evidence, with no expected evidence missing, all without having to leave out any contradictory evidence.
- The coherence theory of truth says that a proposition's truth consists of its fitting into a coherent system of propositions that create the hypothesis.

Both concepts contribute to practical criteria for evaluating competing hypotheses (Table 3).

Table 3: Hypothesis evaluation criteria [Waltz, 2003]

Basis of Truth	Hypothesis Testing Criteria	Application to Intelligence Analysis-Synthesis Criteria
Correspondence	The hypothesis corresponds to all of the data.	1. Completeness: All expected data is present (e.g., there is no missing evidence). 2. Exclusivity: All available data matches the hypothesis; no data contradicts the hypothesis. 3. Non-conflicting: There are no mutually

		exclusive hypotheses that also correspond to the data.
Coherence	The hypothesis coheres to (is consistent with) all propositions that make up the hypothesis.	1. Consistency of logic: The hypothesis-creating system that leads from evidence, relationships (e.g., casual, organizational, or behavioural), and processes (e.g., laws of physics or rules of behaviour) to predicted outcomes is logical and consistent. 2. Consistency of hypotheses: All hypotheses follow the same consistent hypothesis-creating system.

3.3.3 Abductive Method of Scientific Investigation

The stages of the abductive method of scientific investigation of a process (as in the empirical scientific method) include abduction, induction, and deduction in the following sequence [Waltz, 2003]:

1. Observe the process that is not explained; collect data.

2. Apply abduction to create feasible hypotheses that are able to explain the process.

3. Apply induction to test the hypotheses in experiments.

4. Apply deduction to confirm that the selected hypothesis is able to properly predict the process and new observed data.

3.4 Integrating Deduction, Induction and Abduction with Retroduction

[Waltz, 2003] discusses an analysis-synthesis process, which combines each of the fundamental modes of reasoning to accumulate, explore, decompose to fundamental elements, and then fit together evidence. The process also creates hypothesized explanations of the evidence and uses these hypotheses to search for more confirming or refuting elements of evidence to affirm or prune the hypotheses, respectively. One can see the paths of reasoning in a simple flow process, illustrated in Fig. 3, which proceeds from a pool of evidence and a question posed about the evidence (a query to explain the evidence).

This process of proceeding from an evidentiary pool to detections, explanations, or discovery has been called evidence marshalling because the process seeks to marshal (assemble and organize) into a representation (a model) that [Waltz, 2003]:

- Detects the presence of evidence that match previously known premises (or patterns of data).
- Explains underlying processes that gave rise to the evidence.

- Discovers new patterns in the evidence, i.e., patterns of circumstances or behaviours not known before (learning).

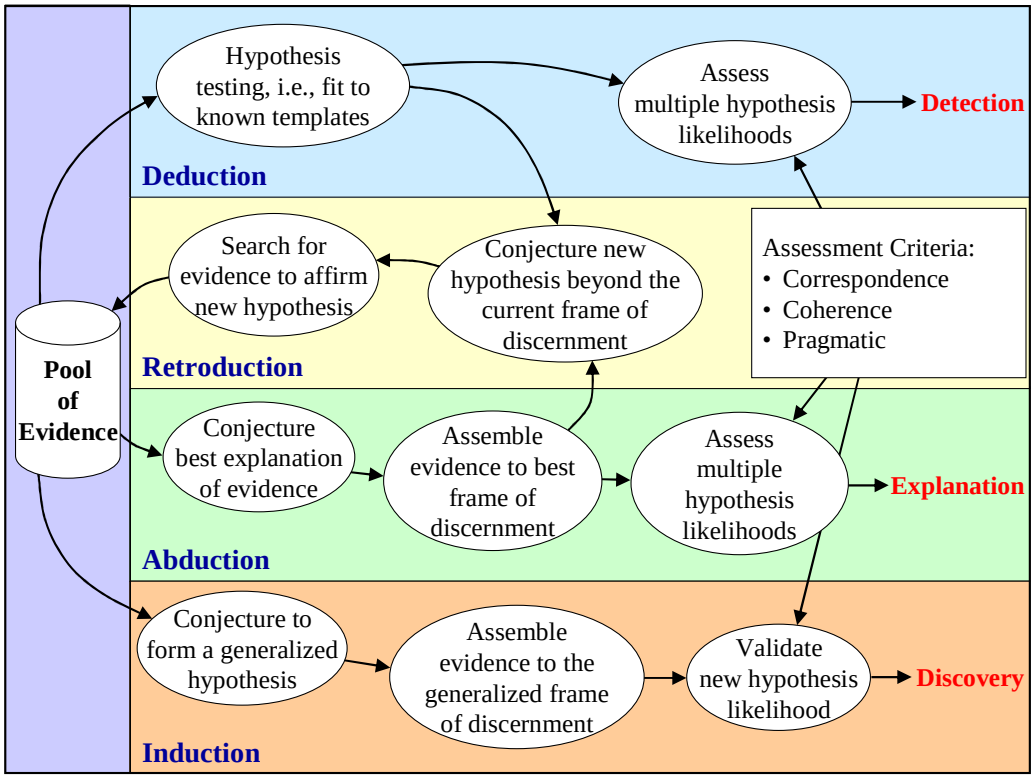


Figure 3: Integrating the basic reasoning flows [Waltz, 2003]

Figure 3 illustrates four basic paths that can proceed from the pool of evidence, i.e., the three fundamental inference modes, and a fourth feedback path [Waltz, 2003]:

1. Deduction. The path of deduction tests the evidence in the pool against previously known patterns (or templates) that represent hypotheses of activities that one seeks to detect. When the evidence fits the hypothesis template, one declares a match. When the evidence fits multiple hypotheses simultaneously, the likelihood of each hypothesis (determined by the strength of evidence for each) is assessed and reported. For example, this likelihood may be computed probabilistically using Bayesian methods, where evidence uncertainty is quantified as a probability and prior probabilities of the hypotheses are known.

2. Retroduction. This feedback path, yet another process of reasoning, occurs when one conjectures (synthesizes) a new conceptual hypothesis (beyond the current frame of discernment) that causes a return to the evidence to seek evidence to match (or test) this new hypothesis. In the testing of hypotheses, one is often inspired to realize new, different hypotheses that might also be tested. In the early implementation of reasoning systems, the forward path of deduction was often referred to as forward chaining by attempting to automatically fit data to previously stored hypothesis templates; the path of retroduction was

referred to as backward chaining, where the system searched for data to match hypotheses queried by an inspired human operator.

3. Abduction. The abduction process, like induction, creates explanatory hypotheses inspired by the pool of evidence and then, like deduction, attempts to fit items of evidence to each hypothesis to seek the best explanation. In this process, the candidate hypotheses are refined and new hypotheses are conjectured. The process leads to comparison and ranking of the hypotheses, and ultimately the best is chosen as the explanation. As a part of the abductive process, one returns to the pool of evidence to seek support for these candidate explanations; this return path is called retroduction.

4. Induction. The path of induction considers the entire pool of evidence to seek general statements (hypotheses) about the evidence. Not seeking point matches to the small sets of evidence, the inductive path conjectures new and generalized explanation of clusters of similar evidence; these generalizations may be tested across the evidence to determine the breadth of applicability before being declared as a new discovery.

3.5 Analogical Reasoning

Another powerful inference method is analogy. Analogical reasoning relates past experiences to a current case [Turban, Aronson, 1998]. The basic idea of reasoning by analogy is to try to consider old situations as guides to new ones [Giarratano, Riley, 1998]. Rather than treat every new situation as unique, it's often helpful to try to relate the new situation to those with which one is familiar. Most living creatures are very good at applying analogical reasoning in their lives, which is essential because of the tremendous number of new situations that are encountered in the real world.

Analogical reasoning is related to induction [Giarratano, Riley, 1998]. Whereas induction makes inferences from the specific to the general of the same situation, analogy tries to make inferences from situations that are not the same. Analogy cannot make formal proofs as deduction can. Instead, analogy is a heuristic reasoning approach that may sometimes work. Reasoning by analogy is an important part of common sense reasoning, which is very difficult for computers. Finally, case-based reasoning (cf. subsection 4.4) can obviously be related to analogical reasoning.

3.6 Generate-and-Test Reasoning

Another method of inference is the basic AI strategy of generate and test, sometimes called generation and test, which involves the generation of a likely solution and then test it to see if the proposed solution meets all the requirements [Giarratano, Riley, 1998]. If the solution is satisfactory, then quit, else generate a new solution, test again, and so forth.

In order to reduce the enormous number of potential solutions, generate-and-test is normally used with a planning program to limit the likely potential solutions for generation [Giarratano, Riley, 1998]. This variation is called plan-generate-test and is used for efficiency in many systems. A plan is essentially finding chains of rules or inferences that connect a problem with a solution, or

goal, with the evidence to support it. Planning is most efficiently done by simultaneously searching forward from the facts and backward from the goal.

Generate-and-test can also be considered the basic inference paradigm of rules [Giarratano, Riley, 1998]. If the conditional elements of a rule are satisfied, it generates some actions such as new facts. The inference engine tests these facts against the conditional elements of rules in the knowledge base. Those rules that are satisfied are put on the agenda and the top priority rule generates its actions, which are then tested, and so on. Thus, generate-and-test produces an inference chain that may lead to the correct solution.

3.7 Model-Based Reasoning

Model-based reasoning is based on knowledge of the structure and behaviour of the devices the system is designed to understand [Turban, Aronson, 1998]. Model-based systems are especially useful in diagnosing difficult equipment problems. Such systems include a (deep-knowledge) model of the devices to be diagnosed that is then used to identify the causes of the equipment's failure. Because they draw conclusions directly from knowledge of a device's structure and behaviour, model-based systems are said to reason from “first principles” (common sense).

In many cases, model-based reasoning is combined with other representation and inference methods [Turban, Aronson, 1998]. Special model-based tools are available (they usually involve frames). Model-based systems can overcome some of the difficulties of rule-based systems.

A necessary condition for model-based reasoning is the creation of a complete and accurate model of the system under study [Turban, Aronson, 1998]. Models can be either mathematical models or component models. A mathematical model can simulate the functions of the system. A component model can contain a functional description of all components and their interactions.

3.7.1 Deducing Hidden Properties of the World

Synchronic (“same time”) rules are axioms relating properties of a world state to other properties of the same world state. There are two main kinds of synchronic rules [Russell, Norvig, 1995]:

1. Causal rules reflect the assumed direction of causality in the world: some hidden property of the world causes certain percepts to be generated. Systems that reason with causal rules are called model-based reasoning systems.

2. Diagnostic rules infer the presence of hidden properties directly from the percept-derived information.

The distinction between model-based and diagnostic reasoning is important in many areas of AI [Russell, Norvig, 1995]. One has the choice of writing diagnostic rules that reason from percepts to propositions about the world, or causal rules that describe how conditions in the world cause percepts to come about. Causal rules are often more flexible and entail a wider range of consequences, but can be more expensive to use in inference.

3.8 Qualitative Reasoning

Qualitative reasoning (QR) is a means of representing and making inferences using general, physical knowledge about the world [Turban, Aronson, 1998]. The main goal of qualitative reasoning is to represent common sense knowledge about the physical world, and the underlying abstractions used by engineers and scientists when they create quantitative models (one doesn't have to know the gravitational constant to know that objects fall). Given such knowledge and appropriate reasoning methods, an expert system could make predictions and diagnoses and explain the behaviour of physical systems qualitatively, even when exact quantitative descriptions are unavailable or intractable.

Qualitative reasoning is a model-based procedure that consequently incorporates deep knowledge about a problem domain [Turban, Aronson, 1998]. Temporal and spatial qualities in decision making are represented effectively by qualitative reasoning methods.

Qualitative reasoning exploits the fact that programs can accept and derive useful inferences from problem statements having much less information than is usually known in traditional mathematical formulations [Turban, Aronson, 1998]. The important point of qualitative representations is not that it is symbolic and uses discrete quantity spaces, but that relevant behaviour is modeled. The way qualitative reasoning works without explicitly solving mathematical models is by applying common-sense mathematical rules to the values of variables and functions in the model (observed) and the interconnections among these elements, such as constraints. Certain relationships among variables hold, and certain valid intervals may be specified for variables (such as 3 to 15, or simply negative, 0, or positive). There are structure rules and behaviour rules. When variables are out of bounds (for example, an infeasible condition occurs), a decision is made to correct the error through the relationships among the variables.

Despite the name qualitative reasoning, typical applications involve the solution of an algorithm or running a simulation to converge on a solution to the problem [Turban, Aronson, 1998]. Typically, qualitative reasoning involves logic.

3.9 Default Reasoning

In the absence of any other information, one would assume that since Tweety is a bird, Tweety can fly [Giarratano, Riley, 1998]. This is an example of default reasoning. Default reasoning can be considered as a rule that makes inferences about rules, or a meta-rule. The inference that Tweety can fly is called a plausible inference that is based on default reasoning (the term plausible means “not impossible”). Default reasoning is non monotonic.

3.10 Autoepistemic Reasoning

Autoepistemic reasoning literally means reasoning about one's own knowledge, as distinct from knowledge in general [Giarratano, Riley, 1998]. Generally, one can do this very well because one knows the limits of his or her knowledge.

Autoepistemic reasoning relies on the closed world assumption [Giarratano, Riley, 1998]. This assumption means that nothing else exists outside the closed world of the expert system.

Anything that cannot be proved is assumed false in a closed world. Any fact that is not known is assumed to be false. Under the closed world assumption, all the possibilities are known. In autoepistemic reasoning, the closed world is one's self-knowledge.

Autoepistemic reasoning is non-monotonic (cf. Section 5), because the meaning of an autoepistemic statement is context sensitive [Giarratano, Riley, 1998]. The term context sensitive means that the meaning changes with the context.

4. Reasoning / Inference Systems

This section reviews the main reasoning/inference systems that have been developed and used over the years to achieve automated reasoning in computer systems.

4.1 Logic Systems

A logic consists of the following [Russell, Norvig, 1995]:

1. A formal system for describing states of affairs, consisting of

    a. the syntax of the language, which describes how to make sentences, and

    b. the semantics of the language, which states the systematic constraints on how sentences relate to states of affairs.

2. The proof theory, i.e., a set of rules for deducing the entailments of a set of sentences

A formal system requires an alphabet of symbols and a set of finite strings of these symbols, the well-formed formulas (wffs) [Giarratano, Riley, 1998]. Some logical arguments are meaningless in a semantic sense; a valid form is essential if the validity of such argument is to be determined. Only wffs can be used in logical arguments.

Any logic system has several goals [Giarratano, Riley, 1998]:

- Specify the form of arguments (i.e., determine the wffs that are used in arguments).
- Indicate the rules of inference that are valid.
- Extend itself by discovering new rules of inference and thus extend the range of arguments that can be proved. By extending the range of arguments, new wffs, called theorems, can be proved by a logic argument.

Logic systems have been developed such as the sentential or propositional calculus, the predicate calculus (i.e., first-order logic (FOL)), and so forth. When a logic system is well developed, it can be used to determine the validity of arguments in a way that is analogous to calculations in systems such as arithmetic, geometry, calculus, physics, and engineering.

4.1.1 Logical Entailment

A logical reasoning procedure allows the manipulation of logical expressions to create new expressions [Turban, Aronson, 1998]. It is the process of deriving new sentences from old ones [Russell, Norvig, 1995] (i.e., because sentences are physical configurations, reasoning must be a process of constructing new physical configurations from old ones).

Proper reasoning should ensure that the new configurations represent facts that actually follow from the facts that the old configurations represent [Russell, Norvig, 1995]. One wants to

generate new sentences that are necessarily true, given that the old sentences are true. This relation between sentences is called entailment, and mirrors the relation of one fact following from another (Fig. 4).

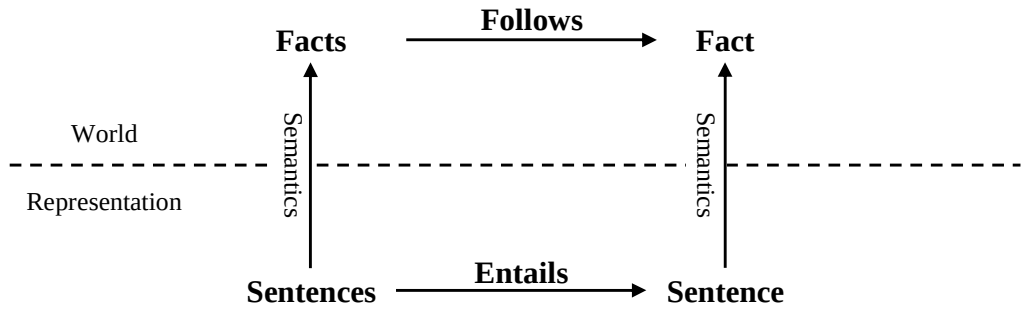


Figure 4: The semantics provides the connection between sentences and facts [Russell, Norvig, 1995]

In mathematical notation, the relation of entailment between a knowledge base *KB* and a sentence α is pronounced "*KB* entails α" and is written as $KB \models \alpha$ [Russell, Norvig, 1995]. More precisely, the concept of logical entailment is as follows [Brachman, Levesque, 2004]:

> One says that the propositions represented by a set of sentences *S* entail the proposition represented by a sentence α when the truth of α is implicit in the truth of the sentences in *S*.

In other words, if the world is such that every element of *S* comes out true, then α does as well.

The reason why logic is relevant to knowledge representation and reasoning is simply that, at least according to one view, logic is the study of entailment relations [Brachman, Levesque, 2004]. Actually, the relevance of first-order logic to AI depends primarily on the entailment relation $\models$ [Ginsberg, 1993]. This is a consequence of the fact that if some agent has an initial state of knowledge consisting of the sentences p_1, p_2, ..., p_n, then it can be argued that the agent will be justified in concluding some new sentence q if (and perhaps only if) $p_1 \wedge p_2 \wedge ... \wedge p_n \models q$.

All that one requires to get some notion of entailment is a language with an account of what it means for a sentence to be true or false [Brachman, Levesque, 2004]. So any knowledge representation language, whatever other features it may have, whatever syntactic form it may take, whatever reasoning procedures one may define over it, ought to have a well-defined notion of entailment.

A knowledge-based system should believe all and only the entailments of what it has explicitly represented [Brachman, Levesque, 2004]. The job of reasoning, then, is to compute the entailments of a KB. However, this is a simplistic account as there are often quite good reasons not to calculate entailments. For one thing, it can be too difficult computationally to decide which sentences are entailed by the kind of KB one will want to use (cf. subsections 4.1.2.4 and 4.1.3). Any procedure that always gives answers in a reasonable amount of time will occasionally either miss some entailments or return some incorrect answers. In the former case, the reasoning process is said to be logically *incomplete* (cf. subsection 4.1.2.3); in the latter case, the reasoning is said to be logically *unsound* (cf. subsection 4.1.2.2). But there are also conceptual reasons why one

might consider unsound or incomplete reasoning. Hence, while it would be a mistake to identify reasoning in a knowledge-based system with logically sound and complete inference, it is the right place to begin.

4.1.1.1 Models and Entailment

An assignment of truth values is an interpretation of a well-formed formula (wff) [Giarratano, Riley, 1998]. A model is an interpretation in which the wff is true. More precisely, a model of a sentence is any world in which that sentence is true under a particular interpretation [Russell, Norvig, 1995].

Models are very important in logic, because, to restate the definition of entailment, a sentence α is entailed by a knowledge base *KB* if the models of *KB* are all models of α [Russell, Norvig, 1995]. If this is the case, then whenever *KB* is true, α must also be true. Stated in a somewhat different way, given two sentences *p* and *q*, one says that *p* entails *q*, writing $p \models q$, if *q* holds in every model in which *p* holds [Ginsberg, 1993]. Yet another perspective is provided by [Brachman, Levesque, 2004]:

> Let *S* be a set of sentences, and α any sentence. One says that α is a logical consequence of *S*, or that *S* logically entails α, which one writes $S \models \alpha$, if and only if, for every interpretation $\mathcal{A}$, if $\mathcal{A} \models S$ then $\mathcal{A} \models \alpha$ [Brachman, Levesque, 2004].

In other words, every model of *S* satisfies α.

4.1.1.2 Semantic Interpretation and Formal Inference in Computers

A well-formed formula (wff) is called *consistent* or *satisfiable* if there is an interpretation (at least one) that makes it true, and *inconsistent* or *unsatisfiable* if the wff is false in all interpretations [Giarratano, Riley, 1998]. A wff is *valid* if it is true in all interpretations; otherwise, it is invalid. Stated differently by [Brachman, Levesque, 2004], one says that a sentence α is logically valid, which one writes $\models \alpha$, when it is a logical consequence of the empty set; α is valid if and only if, for every interpretation $\mathcal{A}$, it is the case that $\mathcal{A} \models \alpha$. A wff is *proved* if it can be shown to be valid [Giarratano, Riley, 1998].

It might seem that valid and unsatisfiable sentences are useless, because they can only express things that are obviously true or false. In fact, validity and unsatisfiability are crucial to the ability of computers to reason [Russell, Norvig, 1995].

The computer suffers from two handicaps [Russell, Norvig, 1995]: 1) it does not necessarily know the interpretation one is using for the sentences in the knowledge base, and 2) it knows nothing at all about the world, except what appears in the knowledge base. What makes formal inference powerful is that there is no limit to the complexity of the sentences it can handle. The great thing about formal inference is that it can be used to derive valid conclusions even when the computer does not know the interpretation one is using. The computer only reports valid conclusions, which must be true regardless of one's interpretation. Because one knows the

interpretation, the conclusions will be meaningful to him/her, and they are guaranteed to follow from the premises.

Hence, one should observe that although the semantic rules of interpretation depend on the interpretation of the non-logical symbols, there are connections among sentences of FOL that do not depend on the meaning of those symbols [Brachman, Levesque, 2004].

4.1.1.3 Knowledge Base (KB) and Entailment

The collection of sentences given as premises to be used as the basis for calculating entailments is what is called a knowledge base (KB). The role of a knowledge representation system is to calculate entailments of this KB [Brachman, Levesque, 2004]. One can think of the KB itself as the beliefs of the system that are explicitly given, and the entailments of that KB as the beliefs that are only implicitly given. This is illustrated in Fig. 1.

One will start with a (large) KB representing what is explicitly known by the knowledge-based system. This KB could be the result of what the system is told, or perhaps what the system found out for itself through perception or learning. Now that one has captured the basic structure of the domain, it is time to turn to the motive one has done this representation in the first place: deriving implicit conclusions from the explicitly represented KB. The goal is to influence the behaviour of the overall knowledge-based system based on what is implicit in this KB, or as close as possible [Brachman, Levesque, 2004]. In general, this will require reasoning.

Just because one is imagining a "rich" collection of sentences in the KB, including the intended connections among the non-logical symbols, one should not be misled into thinking that he/she has done all the work and there is no real reasoning left to do [Brachman, Levesque, 2004]. Actually, it is often non-trivial to move from explicit to implicit beliefs. Calculating what is implicit in a given collection of facts will sometimes involve subtle forms of reasoning. Indeed, it is well known that for FOL the problem of determining whether one sentence is a logical consequence of others is in general unsolvable: No automated procedure can decide validity, and so no automated procedure can tell us in all cases whether or not a sentence is entailed (cf. subsection 4.1.2.4).

4.1.1.4 The Key Point About Logical Entailment/Consequence

Suppose a set of sentences *S* entails a sentence α. Then one does know that whatever the intended interpretation is, if *S* happens to be true in that interpretation, then so must be α [Brachman, Levesque, 2004]. If the user imagines the world satisfying *S* according to his/her understanding of the non-logical symbols, then it satisfies α as well. Other non-entailed sentences may or may not be true, but a knowledge-based system can safely conclude that the entailed ones are. If one tells the system that α is true in the intended interpretation, it can safely conclude any other sentence that is logically entailed, without knowing anything else about that interpretation.

Unfortunately, these conclusions are logically unassailable of course, but not the sort of reasoning one would likely be interested in. In a sense, logical entailment gets us nowhere, since all one is doing is finding sentences that are already implicit in what one was told. This leads us to the fundamental tenet of knowledge representation [Brachman, Levesque, 2004]:

> Reasoning based on logical consequence only allows safe, logically guaranteed conclusions to be drawn. However, by starting with a rich collection of sentences as given premises, including not only facts about particulars of the intended application but also expressing connections among the non-logical symbols involved, the set of entailed conclusions becomes a much richer set, closer to the set of sentences true in the intended interpretation. Calculating these entailments thus becomes more like the form of reasoning one would expect of someone who understood the meaning of the terms involved.

[Brachman, Levesque, 2004] state that, in a sense, this is all there is to knowledge representation and reasoning; the rest is just details.

4.1.2 Logical Inference Mechanism / Procedure

A logical inference mechanism involves thinking of a "logical database" *KB* from which one is trying to derive α [Ginsberg, 1993]. That is, determining what follows from what the knowledge base contains is the job of the inference mechanism [Russell, Norvig, 1995]. From the syntax and semantics, one can derive an inference mechanism that uses the language.

By deductive inference, one means the process of calculating the entailments of a knowledge base [Brachman, Levesque, 2004]. The interesting question is [Ginsberg, 1993]:

"Given a knowledge base *KB* and a sentence α, does $KB \models \alpha$ or not?"

In this regard, an inference procedure can do one of two things [Russell, Norvig, 1995]:

1. Given a knowledge base *KB*, it can generate a new sentence α that claims to be entailed by *KB*.

2. Given a knowledge base *KB* and another sentence α, it can report whether or not α is entailed by *KB*.

An inference procedure *i* can be described by the sentences that it can derive [Russell, Norvig, 1995]. If *i* can derive α from *KB*, then a logician would write $KB \vdash_i \alpha$, which is pronounced "Alpha is derived from *KB* by *i*" or "*i* derives alpha from *KB*". Sometimes the inference procedure is implicit and the *i* is omitted.

4.1.2.1 The Truth-Table Method of Inference

Truth tables can be used not only to define the logical connectives, but also to test for valid sentences [Russell, Norvig, 1995]. Given a sentence, one makes a truth table with one row for each of the possible combinations of truth values for the proposition symbols in the sentence. For each row, one can calculate the truth value of the entire sentence. If the sentence is true in every row, then the sentence is valid.

It is often the case that the sentences input into the knowledge base by the user refer to a world to which the computer has no independent access [Russell, Norvig, 1995]. It is therefore essential

that a reasoning system be able to draw conclusions that follow from the premises, regardless of the world to which the sentences are intended to refer. If a machine has some premises and a possible conclusion, it can determine if the conclusion is true. It can do this by building a truth table for the sentence *Premise* $\Rightarrow$ *Conclusion* and checking all the rows. If every row is true, then the conclusion is entailed by the premises, which means that the fact represented by the conclusion follows from the state of affairs represented by the premises. Even though the machine has no idea what the conclusion means, the user could read the conclusion and use his or her interpretation of the proposition symbols to see what the conclusion means.

Unfortunately, the computation time of the truth-table method of inference is exponential in *n*, and therefore impractical.

4.1.2.2 Soundness / Truth-Preservation

One writes $KB \vdash a$ to indicate that it is possible to derive *a* from *KB*, without really committing on the question of whether or not this derivation is a valid one [Ginsberg, 1993]. One is hoping, of course, that the derivations produced by a procedure are valid; one would like to have $KB \vdash a$ if and only if $KB \models a$.

A reasoning process is considered logically sound if whenever it produces α, then α is guaranteed to be a logical consequence [Brachman, Levesque, 2004]. This rules out the possibility of producing plausible assumptions that may very well be true in the intended interpretation but are not strictly entailed. An inference procedure that generates only entailed sentences is thus called *sound* or *truth-preserving* [Russell, Norvig, 1995]. An inference procedure $\vdash$ will be called sound if whenever $KB \vdash a$, it is also the case that $KB \models a$ [Ginsberg, 1993]. It is sound if it is conservative about the conclusions it draws.

Clearly, sound inference is desirable [Russell, Norvig, 1995], and a desirable property of a logic system is that it be sound [Giarratano, Riley, 1998]. A sound logic system means that every theorem is a logically valid well-formed formula (wff). In other words, a sound logic system will not allow a conclusion to be inferred that is not a logical consequence of its premises. No invalid arguments will be inferred as valid. Sound reasoning is called logical inference or deduction [Russell, Norvig, 1995]. It is a process that implements the entailment relation between sentences.

One thus tries to design sound inference processes that derive true conclusions given true premises [Russell, Norvig, 1995]. How is it achieved? The key to sound inference is to have the inference steps respect the semantics of the sentences they operate upon. That is, given a knowledge base *KB*, the inference steps should only derive new sentences that represent facts that follow from the facts represented by *KB*.

An inference procedure is sound if the conclusion is true in all cases where the premises are true [Russell, Norvig, 1995]. To verify the soundness, one can therefore construct a truth table with one line for each possible model of the proposition symbols in the premise, and shows that in all models where the premise is true, the conclusion is also true.

4.1.2.2.1 Logical Inference Rules

There are certain patterns of inferences that occur over and over again, and their soundness can be shown once and for all [Russell, Norvig, 1995]. Then the pattern can be captured in what is called an inference rule. Once a rule is established, it can be used to make inferences without going through the tedious process of building truth tables. Sanctioned inferences are thus described by rules of inference, which can be used to deduce new facts from old ones [Stefik, 1995].

As previously mentioned, there is the notation $\alpha \vdash \beta$ to say that β can be derived from α by inference [Russell, Norvig, 1995]. However, there is an alternative notation,

$$\frac{\alpha}{\beta}$$

which emphasizes that this is not a sentence, but rather an inference rule. Whenever something in the knowledge base matches the pattern above the line, the inference rule concludes the premise below the line. The letters α, β, etc. are intended to match any sentence.

Rules of inference enable a well-formed formula (wff), A, to be deduced as the conclusion of a finite set, G, of other wffs, where $G = \{A_1, A_2, \ldots, A_n\}$. These wffs must be axioms or other theorems of the logic system. If the argument $A_1, A_2, \ldots, A_n; \therefore A$ is valid, then A is said to be a theorem of the formal logic system and is written with the symbol $\vdash$ [Giarratano, Riley, 1998].

As regards inference rules, there is a fundamental problem with syllogisms because they address only a small portion of the possible logical statements [Giarratano, Riley, 1998]. Propositional logic offers another means of describing arguments. Modus ponens, an inference schema of a particular propositional form, is the best known rule of inference [Stefik, 1995]. The idea of this rule is that if one knows "If p, then q", and if one also knows p, then it is legitimate to conclude q [Ginsberg, 1993]. This can be written as:

$$\frac{\begin{array}{r} p \rightarrow q \\ p \end{array}}{q}$$

Modus ponens can be called by a variety of names [Giarratano, Riley, 1998]:

- direct reasoning
- law of detachment
- assuming the antecedent

It is important because it forms the basis of rule-based expert systems [Giarratano, Riley, 1998]. Modus ponens is sound because the conclusions it draws are always sanctioned by the notion of entailment based on models [Ginsberg, 1993].

The inference rules tell one that if *KB* includes sentences of some specific form, it is legitimate to add sentences of some other form [Ginsberg, 1993]. An example of this is the modus ponens mentioned above. It says that if *KB* contains sentences of the form x and $x \rightarrow y$, it is all right to add y to it.

Many other rules of inference exist for propositional logic. Examples are [Giarratano, Riley, 1998]:

- law of the contrapositive
- modus tollens
- chain rule (law of the syllogism)
- law of disjunctive inference
- law of the double negation
- etc.

However, propositional logic has limitations. Other logics, e.g., first-order logic, are more powerful, and define other rules of inference. Although the rules for propositional logic hold for first-order logic as well, additional inference rules are required to handle first-order logic sentences with quantifiers [Russell, Norvig, 1995]. An important example is the resolution rule [Giarratano, Riley, 1998]. Another important rule of inference is universal instantiation [Stefik, 1995].

4.1.2.2.2 Modus Ponens Extension

One can expand the modus ponens rule to read [Ginsberg, 1993]:

$$\frac{\begin{array}{c} a_1 \wedge \ldots \wedge a_m \rightarrow b \\ a_i \end{array}}{a_1 \wedge \ldots \wedge a_{i-1} \wedge a_{i+1} \wedge \ldots \wedge a_m \rightarrow b}$$

What this rule says is that if b depends on the assumptions $a_1, \ldots, a_m$ and one knows a_i, then one can think of b as depending on only the a's other than a_i. This allows one to remove the assumptions one at a time until one can apply the original version of modus ponens. In fact, one doesn't need to appeal to the "original version" of modus ponens, but can instead apply the above rule again.

If $m = 1$, so that there is only one assumption, and that assumption appears in the database, then the conclusion of the rule says that b is a consequence of a conjunction of no entries [Ginsberg, 1993]. The conjunction of no items is always true, and the conclusion of the rule therefore becomes:

$$T \rightarrow b$$

This is clearly equivalent to *b* itself. Since *d* and $T \rightarrow d$ are equivalent for any *d*, one can rewrite the inference rule as follows. If $d = a_i$,

$$\frac{\begin{array}{r} a_1 \wedge \ldots \wedge a_m \rightarrow b \\ T \rightarrow d \end{array}}{a_1 \wedge \ldots \wedge \hat{a}_i \wedge \ldots \wedge a_m \rightarrow b}$$

where a caret has been included over the term a_i to indicate that it has been dropped from the implication that is the conclusion of the rule.

There is one useful modification that can be made to the above rule [Ginsberg, 1993]. What if *d*, instead of appearing explicitly in the database, itself depends on antecedents $c_1, \ldots, c_n$? In that case, in order to conclude *b* from the *a*'s excepting a_i, one will need to derive a_i, which means that one will need to know all of the *c*'s as well. Modus ponens is the following rule of inference, where $d = a_i$:

$$\frac{\begin{array}{r} a_1 \wedge \ldots \wedge a_m \rightarrow b \\ c_1 \wedge \ldots \wedge c_n \rightarrow d \end{array}}{a_1 \wedge \ldots \wedge \hat{a}_i \wedge \ldots \wedge a_m \wedge c_1 \wedge \ldots \wedge c_n \rightarrow b}$$

Modus ponens is sound.

4.1.2.2.3 Resolution Rule

The very powerful resolution rule of inference introduced by Robinson in 1965 is commonly implemented in theorem-proving AI programs [Giarratano, Riley, 1998]. In fact, resolution is the primary rule of inference in the programming language PROLOG. Instead of many different inference rules of limited applicability, such as modus ponens, modus tollens, merging, chaining, and so forth, PROLOG uses the one general purpose inference rule of resolution. This application of resolution makes automatic theorem provers such as PROLOG practical tools for solving problems. Instead of having to try different rules of inference and hoping one succeeds, the single rule of resolution can be applied. This approach can greatly reduce the search space.

The resolution procedure is a symbol-level procedure for determining whether certain sets of formulas are satisfiable. [Brachman, Levesque, 2004] first discuss a propositional version of resolution, the causal representation it depends on, and how it can be used to compute entailments. They then generalize this account to deal with variables and quantifiers, and show how special answer predicates can be used to find bindings for variables in queries. Finally, they review the computational difficulties inherent in resolution, and show some of the refinements to resolution that are used in practice to deal with them. [Russell, Norvig, 1995] also provide an in depth discussion of the resolution procedure.

4.1.2.2.4 Proof Theory

The record of operation of a sound inference procedure is called a *proof* [Russell, Norvig, 1995]. A logical proof consists of a sequence of applications of inference rules, starting with sentences initially in the KB, and culminating in the generation of the sentence whose proof is desired. Stated slightly differently, a proof can be thought of as a sequence of FOL sentences, starting with those known to be true in the KB (or surmised as part of the assumptions dictated by the query), that proceeds logically using other facts in the KB and the rules of logic until a suitable conclusion is reached [Brachman, Levesque, 2004].

The application of inference rules is simply a question of matching their premise patterns to the sentences in the KB and then adding their (suitably instantiated) conclusion patterns [Russell, Norvig, 1995]. The job of an inference procedure, then, is to construct proofs by finding appropriate sequences of applications of inference rules. If one formulates the process of finding a proof as a search process (this is further discussed in a subsequent subsection), then it would have to be a pretty smart program to find the proof without following any wrong paths.

Note that a different style of proof is based on negating the desired conclusion and showing that this leads to a contradiction [Brachman, Levesque, 2004].

By examining the semantics of logical languages, one can extract what is called the proof theory of the language, which specifies the reasoning steps that are sound [Russell, Norvig, 1995].

4.1.2.3 Completeness

A further desirable property of a formal system is completeness [Giarratano, Riley, 1998]. A reasoning process is considered logically complete if it is guaranteed to produce α whenever α is entailed [Brachman, Levesque, 2004]. This rules out the possibility of missing some entailments, for example, when their status is too difficult to determine.

An inference procedure $\vdash$ will be called *complete* (the following definitions are variations on a theme):

- if it can find a proof for any sentence that is entailed [Russell, Norvig, 1995].
- if whenever $p \models q$, it is also the case that $p \vdash q$ [Ginsberg, 1993].
- if it can derive all true conclusions from a set of premises [Russell, Norvig, 1995].
- if it is capable of finding every valid consequence of an initial database *KB* [Ginsberg, 1993].

A proof procedure is *incomplete* when there are sentences entailed by the knowledge base that the procedure cannot infer [Russell, Norvig, 1995].

The truth-table method of inference is complete, because it is always possible to enumerate the 2^n rows of the table for any proof involving n proposition symbols [Russell, Norvig, 1995]. Although modus ponens is sound, it isn't complete [Ginsberg, 1993].

A set of axioms is complete if every well-formed formula (wff) can be either proved or refuted [Giarratano, Riley, 1998]. The term *refute* means to prove that some assertion is false. In a complete system, every logically valid wff is a theorem.

The notion of completeness has a link with the notion of decidability discussed next in subsection 4.1.2.4.

4.1.2.3.1 The Haystack and Needle Analogy

In understanding entailment and proof, it may help to think of the set of all consequences of *KB* as a haystack and α as a needle. Entailment is like the needle being in the haystack; proof is like finding it [Russell, Norvig, 1995]. For real haystacks, which are finite in extent, it seems obvious that a systematic examination can always decide whether the needle is in the haystack; this is the question of completeness. For many knowledge bases, the haystack of consequences is infinite, and completeness becomes an important issue.

4.1.2.3.2 Completeness Theorem (Gödel)

From the completeness theorem (as stated by Gödel), one can say that if a sentence follows, then it can be proved [Russell, Norvig, 1995]. The completeness theorem is like saying that a procedure for finding a needle in a haystack does exist. For first-order logic, any sentence that is entailed by another set of sentences can be proved from that set, i.e., one can find inference rules that allow a complete proof procedure.

4.1.2.4 Decidability

Unfortunately, there is one problem that is a real nuisance with the completeness theorem [Russell, Norvig, 1995]. Normally, one does not know until the proof is done that a sentence does follow. What if a sentence *q* is not a consequence of what's in the database? What happens when the sentence *q* doesn't follow? Can one tell? Well, for first-order logic, it turns out that one cannot. One might just go merrily along, applying rules of inference to conclude more and more things, but never either stopping or actually concluding *q* [Ginsberg, 1993]. The proof procedure can go on and on, but one will not know if it is stuck in a hopeless loop or if the proof is just about to pop out [Russell, Norvig, 1995]. There is no general method of proof like truth tables for all predicate calculus wffs [Giarratano, Riley, 1998].

So what one has is a procedure that is guaranteed to eventually stop successfully if *q* is derivable, but might not terminate otherwise [Ginsberg, 1993]. If this is the best that one can do, it would mean that the entailment problem is semi-decidable. A problem is decidable if there is a procedure that is guaranteed to terminate with an answer whether that answer is “yes” or “no”; a problem is semi-decidable if one can guarantee termination in one of these case, but not in both [Ginsberg, 1993].

All propositional well-formed formulas (wffs) can be proved by the truth-table method, since there is only a finite number of interpretations for wffs, and so the propositional calculus is decidable [Giarratano, Riley, 1998]. Entailment in first-order logic is semi-decidable, i.e., one can show that sentences follow from premises, if they do, but one cannot always show if they do not

[Russell, Norvig, 1995]. Consistency of sets of sentences (the question of whether there is a way to make all the sentences true) is also semi-decidable. There are two consequences of this semi-decidability [Ginsberg, 1993]:

1. It makes a lot of problems not decidable at all; these are problems that depend on the failure of an attempt to prove some sentence *q*.

2. People spend a lot of time trying to figure out how to get away from it.

Planning problems are typically not even semi-decidable [Ginsberg, 1993]. When attacking a non-decidable problem, one needs some way to decide how much of the computational resources should be allocated to solving it [Ginsberg, 1993]; at what point does one simply give up and work with the best answer found so far?

4.1.3 Computational Difficulties with Logical Reasoning Procedures

At the knowledge level, the specification for an idealized deductive procedure is clear [Brachman, Levesque, 2004]:

- Given a knowledge base KB and a sentence α, one would like a procedure that can determine whether or not $KB \models \alpha$.

- If $\beta[x_1, \ldots, x_n]$ is a formula with free variables among the x_i, one wants a procedure that can find terms t_i, if they exist, such that $KB \models \beta[t_1, \ldots, t_n]$.

This is idealized; no computational procedure can fully satisfy this specification. No automated reasoning process for FOL can be both sound and complete in general. What one is really after, in the end, is a procedure that does deductive reasoning in as sound and complete a manner as possible, and in a language as close as possible to that of full FOL [Brachman, Levesque, 2004].

The resolution procedure could in principle be used to calculate entailments of any FOL KB [Brachman, Levesque, 2004]. However, in its most general form, resolution runs into serious computational difficulties. Resolution is not a panacea. For knowledge representation purposes, one would like to be able to produce entailments of a KB for immediate action, but determining the satisfiability of clauses may simply be too difficult computationally for this purpose. Although refinements to resolution can help, the problem can never be completely eliminated. This is a consequence of the fundamental computational intractability of first-order entailment.

One may need to consider some other options. One option, as alluded in the previous subsection, is to give more control over the reasoning process to the user. Another option is to consider the possibility of using representation languages that are less expressive than full FOL or even full propositional logic [Brachman, Levesque, 2004].

4.1.3.1 Introducing Language Restrictions

It is sometimes possible to limit oneself to only a certain interesting subset of FOL, where the resolution procedure becomes much more manageable. Some people look for subsets of first-

order logic that restrict the language in some way that makes it decidable [Ginsberg, 1993]. From a representation standpoint, the subset in question is still sufficiently expressive for many purposes [Brachman, Levesque, 2004].

Note that even in fragments of first-order logic where entailment is decidable, it may still be exponentially difficult, so people continue to work, finding still smaller fragments of the language in which the difficulty of any particular entailment question is low-order polynomial [Ginsberg, 1993]. Much of the research in knowledge representation and reasoning can be seen as attempts to deal with this issue [Brachman, Levesque, 2004].

The main idea in the design of useful "limited" languages is that there are reasoning tasks that can be easily formulated in terms of FOL entailment, i.e., in terms of whether or not KB $\models \alpha$, but that can also be solved by special-purpose methods because of restrictions on the KB or on α [Brachman, Levesque, 2004].

4.1.3.1.1 Horn Sentences/Clauses and Horn Databases

A simple example of this is Horn clause entailment [Brachman, Levesque, 2004]. There is a useful class of sentences for which a polynomial-time inference procedure exists [Russell, Norvig, 1995]. This is the class called *Horn sentences*. Such a sentence has the form:

$$P_1 \wedge P_2 \wedge ... \wedge P_n \Rightarrow Q$$

where P_i and Q are non-negated atoms. A database will be called *Horn* if it is equivalent to a set of sentences of the form $a_1 \wedge \ldots \wedge a_m \rightarrow b$, where b and each a_i are atoms [Ginsberg, 1993].

One could obviously use full resolution to handle Horn clauses, but there is no need to, because SLD resolution offers a much more focused search [Brachman, Levesque, 2004]. SLD resolution is the basic inference rule used in logic programming [Wikipedia, 2008]. It is a refinement of resolution, which is both sound and refutation complete for Horn clauses. The name "SLD resolution" was given by Maarten van Emden for the unnamed inference rule introduced by Robert Kowalski. Its name is derived from SL resolution, which is both sound and refutation complete for the unrestricted clausal form of logic. "SLD" stands for "SL resolution with Definite clauses". In both, SL and SLD, "L" stands for the fact that a resolution proof can be restricted to a linear sequence of clauses. In both SL and SLD, "S" stands for the fact that the only literal resolved upon in any clause is one that is uniquely selected by a selection rule or selection function.

In the propositional case, it is known that there is a procedure guaranteed to run in linear time for doing the reasoning, whereas a full resolution procedure need not and likely would not do as well [Brachman, Levesque, 2004]. In the propositional case, one can determine if a Horn KB entails an atom in a linear number of steps.

Modus ponens is complete for Horn databases [Ginsberg, 1993]. Not every knowledge base can be written as a collection of Horn sentences, but for those who can, one can use a simple inference procedure: apply Modus Ponens wherever possible until no new inferences remain to be made [Russell, Norvig, 1995].

Horn databases and the modus ponens rule are thus important for two reasons [Ginsberg, 1993]. First, a lot of the knowledge that one has about the world can be described in Horn form. Second, Horn databases have attractive computational properties. In a database without function constants and where every sentence is of the restricted form $a_1 \wedge \ldots \wedge a_m \rightarrow b$, the time needed to determine whether or not a particular atom follows is of order nd, where n is the maximum number of premises in any implication and d is the size of the database.

Because of the simplicity of working with Horn databases, much of the AI work on reasoning restricts itself to this case [Ginsberg, 1993]. Unfortunately, even with Horn clauses, one still has the possibility of a procedure that runs forever for the first-order case [Brachman, Levesque, 2004]. The problem of determining whether a set of first-order Horn clauses entails an atom remains undecidable. So no procedure can be guaranteed to always work, despite the fact that the propositional case is so easy. This is not too surprising because PROLOG, which is in many ways a theorem prover for first-order logic, is a full programming language, and being able to decide if an atom is entailed would imply being able to decide if a PROLOG program would halt.

As with non-Horn clauses, the best that can be expected in the first-order case is to give control of the reasoning to the user to help avoid redundancies and infinite branches [Brachman, Levesque, 2004]. Unlike the non-Horn case, however, Horn clauses are much easier to structure and control in this way.

4.1.3.1.2 Description Logics as Another Example of a "Limited" Language

A less obvious example of a "limited" language is provided by description logics in general (cf. subsection 4.1.6) [Brachman, Levesque, 2004]. It is not hard to formulate subsumption (cf. subsection 4.1.6.1) in terms of FOL entailment. One can imagine introducing predicate symbols for concept expressions and writing meaning postulate for them in FOL. So, if one wanted to, one could use full resolution to calculate concept subsumption. But, for some description logic languages, there are very good subsumption procedures. It would be extremely awkward to try to coax efficient structure-matching behaviour out of a general-purpose resolution procedure.

4.1.3.1.3 Special-Purpose Reasoning Procedures and Reasoning by Cases

This idea of limited languages obviously generalizes: It will always be advantageous to use a special-purpose reasoning procedure when one exists even if a general-purpose procedure like resolution is applicable [Brachman, Levesque, 2004].

When does one expect not to be able to use a specialized procedure to advantage? Special-purpose reasoning methods will not help if one is forced to reason by cases and invoke these procedures exponentially often [Brachman, Levesque, 2004]. But can one avoid this type of case analysis? Unfortunately, it seems to be demanded by languages like FOL. The constructs of FOL are ideally suited to expressing incomplete knowledge. The logical operators of FOL allow one to express knowledge in a way that does not force him/her to answer all questions. In fact, one can understand the expressiveness of FOL not in terms of what it allows him/her to say, but in terms of what it allows him/her to leave unsaid.

The trouble with cases is that they multiply together, and so very quickly there are too many of them to enumerate [Brachman, Levesque, 2004]. This is not to suggest that one is required to enumerate the cases to reason correctly. Indeed, whether one needs a reasoning procedure that scales with the number of cases remains open, and is perhaps the deepest open problem in computer science. Not too surprisingly then, the limited languages (e.g., Horn clauses, description logics) do not allow some form of incomplete knowledge to be represented [Brachman, Levesque, 2004].

This then suggests a general direction to pursue to avoid intractability [Brachman, Levesque, 2004]: Restrict the contents of a KB somehow so that reasoning by cases is not required. One natural question along these lines is this: Is complete knowledge sufficient to ensure tractability? That is, if for every sentence α one cares about the KB entails α or the KB entails $\neg\alpha$, can one efficiently determine which? The answer unfortunately is no.

4.1.3.2 Theorem Proving as a Search Problem

Why is it that determining whether or not $KB \models q$ is so computationally difficult? If one is trying to determine whether or not *q* follows from *KB*, there will typically be many available rules of inference (such as modus ponens), and one needs to decide which one to use [Ginsberg, 1993]. Moreover, there will typically be many possible ways to apply modus ponens in a particular situation. Here is the reduction of a problem in theorem proving to a problem in search.

Because there are typically many ways in which rules of inference can be applied in any particular situation, deduction is a search problem [Ginsberg, 1993]. The initial node is the information with which the system is supplied, and the goal nodes are those in which the desired conclusion has been derived. The operators that generate the successors of a given node are those that draw a new conclusion by applying some rule of inference. More broadly, the intended role of knowledge representation in artificial intelligence is to reduce problems of intelligent action to search problems.

Provided that $\vdash$ is complete, applying the rules of inference to sentences derived as early as possible is a rough analog to breadth-first search [Ginsberg, 1993]. It is therefore guaranteed to be complete as well: if *q* is a consequence of the material in the database, this approach will eventually find it. Of course, since search problems are exponentially difficult, one may not find a proof of *q* quickly using this idea.

Actually, in spite of the parallels, there are also differences between the sorts of search problems generated by theorem-proving tasks and those that arise in other areas [Ginsberg, 1993]. In theorem proving, for example, it is generally the case that the result of performing two inference steps is independent of the order in which those inferences are performed; it doesn't matter if one derives *a* first and then *b* or vice versa. A consequence of this is that the associated search space is not a tree, but a densely connected graph, since there are typically a variety of paths to any particular intermediate or terminal node. Another difference is a consequence of the fact that the depth of the goal nodes is almost invariably unknown in proof problems. Inference problems often give rise to conjunctive sub-goals, where an attempt to derive a single conclusion leads to a collection of sub-problems, all of which must be solved in order for the search to terminate successfully.

It is worth observing that in some applications of resolution it is reasonable to wait for answers, even for a very long time [Brachman, Levesque, 2004]. The best one can hope for in such applications of resolution is not a guarantee of efficiency or even of termination, but a way to search for derivations that eliminates unnecessary steps as much as possible. [Brachman, Levesque, 2004] consider strategies that can be used to improve the search in this sense.

4.1.4 Logic Programming Systems

The declarative approach has many advantages for building intelligent systems [Russell, Norvig, 1995]. Logic programming tries to extend these advantages to all programming tasks. Any computation can be viewed as a process of making explicit the consequences of choosing a particular program for a particular machine and providing particular inputs. Logic programming views the program and inputs as logical statements about the world, and the process of making consequences explicit as a process of inference.

The relation between logic and algorithm is summed up in Robert Kowalski's equation [Russell, Norvig, 1995]:

Algorithm = Logic + Control

A logic programming language makes it possible to write algorithms by augmenting logical sentences with information to control the inference process. Logic programming languages typically restrict the logic, disallowing full treatment of negation, disjunction, and/or equality. They may include some non-logical features of programming languages (such as input and output).

Examples of logic programming languages are PROLOG, MRS, LIFE [Russell, Norvig, 1995]. PROLOG is by far the most widely used logic programming language. PROLOG and most other logic programming languages are backward chaining.

4.1.5 Theorem Provers

Theorem provers (also known as automated reasoners) use resolution (or some other complete inference procedure) to prove sentences in full first-order logic, often for mathematical and scientific reasoning tasks [Russell, Norvig, 1995]. They can also be used to answer questions: the proof of a sentence containing variables serves as an answer to a question because it instantiates the variables.

Theorem provers need control information to operate efficiently, but this information is kept distinct from the knowledge base, rather than being part of the knowledge representation itself. Most of the research in theorem provers is in finding control strategies that are generally useful. Examples of theorem provers are SAM, AURA, OTTER [Russell, Norvig, 1995].

4.1.5.1 Theorem Provers Versus Logic Programming Language

Theorem provers differ from logic programming languages in two ways [Russell, Norvig, 1995]. First, most logic programming languages only handle Horn clauses, whereas theorem provers

accept full first-order logic. Second, PROLOG programs (for example) intertwine logic and control [Russell, Norvig, 1995].

4.1.5.2 Theorem Provers as Assistants

A use of theorem provers is as an assistant, providing advice [Russell, Norvig, 1995]. In this mode, the user acts as a supervisor, mapping out the strategy for determining what to do next and asking the theorem prover to fill in the details. This alleviates the problem of semi-decidability to some extent, because the supervisor can cancel a query and try another approach if the query is taking too much time. A theorem prover can also act as a proof-checker, where the proof is given by a human as a series of fairly large steps; the individual inferences required to show that each step is sound are filled in by the system.

4.1.6 Description Logic Systems

Description logic systems have been discussed in [Roy, Auger, 2008-A]. The definition of entailment in description logics (DLs) is exactly like it is in FOL. There are two basic sort of reasoning one is concerned with in DLs [Brachman, Levesque, 2004]:

- determining whether or not some constant c satisfies a certain concept d, and
- determining whether or not a concept d_1 is subsumed by another concept d_2.

Both of these involve calculating entailments of a KB [Brachman, Levesque, 2004]:

- In the first case, one needs to determine if the KB entails ($c \rightarrow d$).
- In the second case, one needs to determine if the KB entails ($d_1 \subseteq d_2$).

where the symbol "$\subseteq$" (representing subsumption) is defined in the next subsection. So, as in FOL, reasoning in description logic means calculating entailments. Note that in some cases the entailment relationship will hold because the sentences themselves are valid. In more typical cases, the entailment relationship will depend on the sentences in the KB. Note also that the first case depends on being able to handle the second. Actually, computing whether an individual denoted by a constant satisfies a concept is very similar to computing subsumption between two concepts.

4.1.6.1 Subsumption

Hence, the basic inference on concept expressions in description logics is *subsumption* (i.e., the principal inference tasks in description logic systems are subsumption tests among concepts [Russell, Norvig, 1995]). Typically, for two concepts C and D, subsumption is written as [Nardi, Brachman, 2003]:

$$C \subseteq D$$

To subsume is to include or place within something larger or more comprehensive [Merriam-Webster, 2003]. Determining subsumption is the problem of checking whether the concept denoted by D (the *subsumer*) is considered more general than the one denoted by C (the

subsumee) [Nardi, Brachman, 2003]. In other words, subsumption checks whether the first concept always denotes a subset of the set denoted by the second one. For example, one might be interested in knowing whether *Woman* $\subseteq$ *Mother*.

Note that the subsumption tests can also be viewed as checking if one category is a subset of another, based on their definitions and classification, or checking if an object belongs to a category [Russell, Norvig, 1995]. A problem instance is solved by describing it and asking if it is subsumed by one of several possible solution categories.

In order to verify this kind of relationship one has in general to take into account the relationships defined in the terminology [Nardi, Brachman, 2003]. Under appropriate restrictions, one can embody such knowledge directly in concept expressions, thus making subsumption over concept expressions the basic reasoning task.

4.1.6.1.1 Subsumption and TBoxes

The basic deduction service for TBoxes ([Roy, Auger, 2008-A]) can be viewed as *logical implication* and it amounts to verifying whether a generic relationship (for example a subsumption relationship between two concept expressions) is a logical consequence of the declarations in the TBox.

4.1.6.1.2 Subsumption and ABoxes

The presence of individuals in a knowledge base makes reasoning more complex from a computational viewpoint, and may require significant extensions of some TBox reasoning techniques [Nardi, Brachman, 2003]. The basic reasoning task in an ABox ([Roy, Auger, 2008-A]) is *instance checking*, which verifies whether a given individual is an instance of (belongs to) a specified concept. Although other reasoning services are usually considered and employed, they can be defined in terms of instance checking. Among them one finds:

- *knowledge base consistency*, which amounts to verifying whether every concept in the knowledge base admits at least one individual,
- *realization*, which finds the most specific concept an individual object is an instance of, and,
- *retrieval*, which finds the individuals in the knowledge base that are instances of a given concept.

These can all be accomplished by means of instance checking.

4.1.6.2 Satisfiability

From the subsumption tests, description logics systems derive concept satisfiability and consistency in the models represented [Gómez-Pérez et al, 2004]. Concept satisfiability is the problem of checking whether a concept expression does not necessarily denote the empty concept [Nardi, Brachman, 2003]. In fact, it is a special case of subsumption, with the subsumer being the empty concept, meaning that a concept is not satisfiable.

4.1.6.3 Subsumption Algorithms

Although the meaning of concepts had already been specified with a logical semantics, the design of inference procedures in description logics was influenced for a long time by the tradition of semantic networks, where concepts were viewed as nodes and roles as links in a network [Nardi, Brachman, 2003]. Subsumption between concept expressions was recognized as the key inference and the basic idea of the earliest subsumption algorithms was to transform two input concepts into labeled graphs and test whether one could be embedded into the other; the embedded graph would correspond to the more general concept (the subsumer). This method is called *structural comparison*, and the relation between concepts being computed is called *structural subsumption*. However, a careful analysis of the algorithms for structural subsumption shows that they are *sound*, but not always *complete* in terms of the logical semantics: whenever they return "yes" the answer is correct, but when they report "no" the answer may be incorrect [Nardi, Brachman, 2003]. In other words, structural subsumption is in general weaker than logical subsumption.

The need for complete subsumption algorithms is motivated by the fact that in the usage of knowledge representation systems it is often necessary to have a guarantee that the system has not failed in verifying subsumption [Nardi, Brachman, 2003]. Consequently, new algorithms for computing subsumption have been devised that are no longer based on a network representation, and these can be proven to be complete. Description logic systems provide efficient automatic classifiers for the subsumption test among concepts [Gómez-Pérez et al, 2004]. These classifiers are commonly built by means of tableaux calculus and constraint systems. Algorithms have been developed by specializing classical settings for deductive reasoning to the DL subsets of first-order logics, as done for tableau calculi, and also by more specialized methods [Nardi, Brachman, 2003].

4.1.6.4 Tractability of Inference / Subsumption

Perhaps the most important aspect of description logics is the emphasis on tractability of inference [Russell, Norvig, 1995]. As already briefly discussed in [Roy, Auger, 2008-A], there is a trade-off between the expressiveness of a representation language and the difficulty of reasoning over the representations built using that language [Nardi, Brachman, 2003]. In other words, the more expressive the language, the harder the reasoning.

In standard first-order logic systems, predicting the solution time is often impossible [Russell, Norvig, 1995]. It is often left to the user to engineer the representation to detour around set of sentences that seem to be causing the system to take several weeks to solve a problem. The thrust in description logics, on the other hand, is to ensure that subsumption-testing can be solved in time polynomial in the size of the problem description. This sounds wonderful in principle, until one realizes that it can only have one of two consequences: hard problems either cannot be stated at all, or require exponentially large descriptions.

An example of the trade-off discussed above can be generated by analyzing the language $\mathcal{FL}^-$ (Frame Language), which includes intersection of concepts, value restrictions and a simple form of existential quantification [Nardi, Brachman, 2003]. It can be showed that for such a language the subsumption problem could be solved in polynomial time, while adding a construct called role

restriction to the language makes subsumption a coNP-hard problem (the extended language was called $\mathcal{FL}$).

Work on the trade-off between expressiveness and reasoning efficiency introduced at least two new ideas [Nardi, Brachman, 2003]:

- "efficiency of reasoning" over knowledge structures can be studied using the tools of computational complexity theory;
- different combinations of constructs can give rise to languages with different computational properties.

An immediate consequence of the above observations is that one can study formally and methodically the trade-off between the computational complexity of reasoning and the expressiveness of the language, which itself is defined in terms of the constructs that are admitted in the language. After some initial work, a number of results on this trade-off for concept languages were obtained, and these results allow to draw a fairly complete picture of the complexity of reasoning for a wide class of concept languages. Moreover, the problem of finding the optimal trade-off, namely the most expressive extensions of $\mathcal{FL}^-$ with respect to a given set of constructs that still keep subsumption polynomial, has been studied extensively.

One of the assumptions underlying this line of research is to use worst-case complexity as a measure of the efficiency of reasoning in description logics (and more generally in knowledge representation formalisms) [Nardi, Brachman, 2003]. Such an assumption has sometimes been criticized as not adequately characterizing system performance or accounting for more average-case behavior. While this observation suggests that computational complexity alone may not be sufficient for addressing performance issues, research on the computational complexity of reasoning in description logics has most definitely led to a much deeper understanding of the problems arising in implementing reasoning tools.

[Nardi, Brachman, 2003] briefly address some of the contributions of this body of work. First of all, the study of the computational complexity of reasoning in description logics has led to a clear understanding of the properties of the language constructs and their interaction. The tractability results do shed light on what sorts of constructs cause problems, and thus help the user to understand how different representations behave [Russell, Norvig, 1995]. This is not only valuable from a theoretical viewpoint, but gives insight to the designer of deduction procedures, with clear indications of the language constructs and their combinations that are difficult to deal with, as well as general methods to cope with them [Nardi, Brachman, 2003]. For example, description logics usually lack negation and disjunction [Russell, Norvig, 1995]. These both force first-order logical systems to essentially go through an exponential case analysis in order to ensure completeness. With disjunctive descriptions, nested definitions can lead easily to an exponential number of alternative routes by which one category can subsume another.

Secondly, the complexity results have been obtained by exploiting a general technique for satisfiability-checking in concept languages, which relies on a form of tableau calculus [Nardi, Brachman, 2003]. Such a technique has proved extremely useful for studying both the correctness and the complexity of the algorithms. More specifically, it provides an algorithmic framework that is parametric with respect to the language constructs. The algorithms for concept satisfiability

and subsumption obtained in this way have also led directly to practical implementations by application of clever control strategies and optimization techniques. The most recent knowledge representation systems based on description logics adopt tableau calculi.

Thirdly, the analysis of pathological cases in this formal framework has led to the discovery of incompleteness in the algorithms developed for implemented systems [Nardi, Brachman, 2003]. This has also consequently proven useful in the definition of suitable test sets for verifying implementations. For example, the comparison of implemented systems has greatly benefitted from the results of the complexity analysis.

4.1.6.5 Very Expressive Description Logics

After the tradeoff between expressiveness and tractability of reasoning was thoroughly analyzed and the range of applicability of the corresponding inference techniques had been experimented with, there was a shift of focus in the theoretical research on reasoning in description logics [Nardi, Brachman, 2003]. Interest grew in relating description logics to the modeling languages used in database management. In addition, the discovery of strict relationships with expressive modal logics stimulated the study of so-called very expressive description logics. These languages, besides admitting very general mechanisms for defining concepts (for example cyclic definitions, addressed in the next section), provide a richer set of concept-forming constructs and constructs for forming complex role expressions. For these languages, the expressiveness is great enough that the new challenge became enriching the language while retaining the decidability of reasoning. It is worth pointing out that this new direction of theoretical research was accompanied by a corresponding shift in the implementation of knowledge representation systems based on very expressive DL languages.

4.1.6.6 Examples of Description Logic Systems

Some examples of first and second generation description logic systems include KL-ONE, CLASSIC, and LOOM [Russell, Norvig, 1995]. Next generation of description logic systems are represented by FaCT, DLP, and RACER [Möller, Haarslev, 2003].

4.2 Rule-Based Systems

Rule-based systems (often called “production systems”) have been discussed in [Roy, Auger, 2008-A]. Like logic programming languages, rule-based systems use implications (essentially modus ponens) as their primary representation. The consequent of each implication is interpreted as an action recommendation, rather than simply a logical conclusion. The typical production system has these features [Russell, Norvig, 1995]:

- The system maintains a knowledge base called the working memory. This contains a set of positive literals with no variables.
- The system also maintains a separate rule memory. This contains a set of inference rules, each of the form $p_1 \wedge p_2 \ldots \Rightarrow act_1 \wedge act_2 \ldots$, where the p_i are literals, and the act_i are actions to take when the p_i are all satisfied. Allowable actions are adding and deleting elements from the working memory, and possibly other actions (such as printing a value).

The inference engine in a rule-based system determines which rule antecedents, if any, are satisfied by the facts [Giarratano, Riley, 1998]. Testing a rule premise can be as simple as matching a symbolic pattern in the rule to a similar pattern in the assertion base [Turban, Aronson, 1998]. This activity is called pattern matching. Every rule in the knowledge base can be checked to see whether its premise can be satisfied by previously made assertions. The true (or false) values of the antecedents can also be obtained by querying the user or by checking other rules.

When the inference engine notices that a fact satisfies the conditional part of a rule, it puts this rule on the agenda [Giarratano, Riley, 1998]. If a rule has multiple patterns, then all of its patterns must be simultaneously satisfied for it to be placed on the agenda. Some patterns may even be satisfied by specifying the absence of certain facts in working memory. A rule whose patterns are all satisfied is said to be activated or instantiated. Multiple activated rules may be on the agenda at the same time, in which case the inference engine must select one rule for firing.

Following the THEN part of a rule is a list of actions to be executed when the rule fires [Giarratano, Riley, 1998]. Specific actions usually include the addition or removal of facts from the working memory (or insertions and deletions from the knowledge base) as well as input and output (e.g., printing results). A conclusion drawn is stored in the assertion base, and it could then be used to satisfy the premises of other rules [Turban, Aronson, 1998].

Examples of production systems are OPS-5, CLIPS, SOAR [Russell, Norvig, 1995].

4.2.1 Forward and Backward Chaining in Rule-Based Systems

The inference process may be done in one of two directions, forward or backward, and will continue until no more rules can fire or until a goal is achieved [Turban, Aronson, 1998]. There are thus two approaches for controlling inference in rule-based systems: forward chaining and backward chaining (each of which has several variations). The word chaining in rule-based systems signifies the linking of a set of pertinent rules.

Using the forward chaining approach, the rule-based system analyzes the problem by looking for the facts that match the IF portion of its IF-THEN rules [Turban, Aronson, 1998]. As each rule is tested, the program works its way toward one or more conclusions. This forward-chaining approach can be appropriate for the design of a rule-based agent; on each cycle, one adds the percepts to the knowledge base and run the forward chainer, which chooses an action to perform according to a set of condition-action rules [Russell, Norvig, 1995]. Theoretically, one could implement a production system with a theorem prover, using resolution to do forward chaining over a full first-order knowledge base. A more restricted language, on the other hand, can provide greater efficiency because the branching factor is reduced.

Using the backward chaining approach, the control program starts with a goal to be verified as either true or false [Turban, Aronson, 1998]. Then it looks for a rule that has that goal in its conclusion. It then checks the premise of that rule in an attempt to satisfy this rule. It checks the assertion base first. If the search fails there, the rule-based system looks for another rule whose conclusion is the same as that of the first rule. An attempt is then made to satisfy the second rule. The process continues until all the possibilities that apply are checked, or until the initially

checked rule (with the goal) is satisfied. If the goal is proven false, then the next goal is tried. In some inference, even if the goal is proven true, the rest of the goals may be tried in succession.

Inference with rules involves implementation of modus ponens, which is reflected in the search mechanism [Turban, Aronson, 1998]. Going either backward or forward, one can use heuristics to make the search more efficient.

4.2.2 The Rule Interpreter

The execution of forward or backward chaining in rule-based systems is done with the aid of a rule interpreter [Turban, Aronson, 1998]. Its function is to examine production rules to determine which are capable of being fired, and then to fire the rules. The control strategy of the rule interpreter (for example, backward chaining) determines how the appropriate rules are found and when to apply them.

4.2.3 Inference Cycle

The inference engine operates in cycles [Giarratano, Riley, 1998]. Various names have been given to describe the cycle, such as:

- recognize-act cycle
- select-execute cycle
- situation-response cycle
- situation-action cycle

The inference engine will repeatedly execute a group of tasks until certain criteria cause execution to cease [Giarratano, Riley, 1998]. The tasks of a typical inference cycle are listed in Table 4.

Table 4: Tasks of a typical inference cycle [Giarratano, Riley, 1998]

Match phase	The system computes the subset of rules whose left-hand side is satisfied by the current contents of the working memory. That is, the system updates the agenda by checking whether the LHSs of any rules are satisfied. If so, the system activates them. The system removes activations if the LHSs of their rules are no longer satisfied.
Conflict resolution phase	The system decides which of the rules should be executed. If there are activations, then the system selects the one with the highest priority, else done.
Act phase	The system executes the action(s) in the chosen rule(s). The system sequentially performs the actions on the RHS of the selected activation. Those that change the working memory have an immediate effect in this cycle. The system removes the activation that has just fired from the agenda.

Check for halt	If a halt action is performed or a break command given, then done.

Multiple rules may be activated and put on the agenda during one cycle [Giarratano, Riley, 1998]. Also, activations will be left on the agenda from previous cycles unless they are deactivated because their LHS is no longer satisfied. Thus, the number of activations on the agenda will vary as execution proceeds.

4.2.4 Incorrect Rules?

Depending on the program, an activation may always be on the agenda but never selected for firing [Giarratano, Riley, 1998]. Likewise, some rules may never become activated. In these cases, the purpose of these rules should be re-examined because either the rules are unnecessary or their patterns were not correctly designed.

4.2.5 Priority Schemes

The inference engine executes the actions of the highest priority activation on the agenda, then the next highest priority activation, and so on until no activations are left [Giarratano, Riley, 1998]. Various priority schemes have been designed into rule-based system tools (also called "shells"). Generally, all shells let the knowledge engineer define the priority of rules.

4.2.5.1 Markov Algorithms

Markov specified a control structure for production systems [Giarratano, Riley, 1998]. A Markov algorithm is an ordered group of productions (with higher priority rules ordered first) that are applied in order of priority to an input string.

There is a definite control strategy to Markov algorithms [Giarratano, Riley, 1998]. As long as the highest priority rule applies, it is used. If the highest priority rule is not applicable, then the next one is applied (i.e., the Markov algorithm tries lower priority rules), and so forth. The Markov algorithm terminates if either (1) the last production is not applicable to a string, or (2) a production that ends with a period is applied. Markov algorithms can also be applied to substrings of a string, starting from the left.

4.2.6 The Rete Algorithm

Although the Markov algorithm can be used as the basis of a rule-based system, it is highly inefficient for systems with many rules [Giarratano, Riley, 1998]. The problem of efficiency is of major importance if one wants to create rule-based systems for real problems containing hundreds or thousands of rules. What is really needed is an algorithm that knows about all the rules and can apply any rule without having to try each one sequentially. A solution of this problem is the Rete algorithm.

The Rete algorithm is a fast pattern matcher that obtains its speed by storing information about rules in a network [Giarratano, Riley, 1998]. Instead of having to match facts against every rule on every inference cycle, the Rete algorithm looks only for changes in matches on every cycle.

This greatly speeds up the matching of facts to antecedents since the static data that don't change from cycle to cycle can be ignored.

4.2.7 Conflict Resolution

Agenda conflicts occur when different activations have the same priority and the inference engine must decide on one rule to fire [Giarratano, Riley, 1998]. Different shells have different ways of dealing with this problem.

A conflict resolution mechanism is also necessary to decide which action to take when more than one is recommended [Russell, Norvig, 1995].

4.2.8 Refraction

Rule-based systems are built using refraction in order to prevent trivial loops [Giarratano, Riley, 1998]. Using refraction, no input can cause the rule to fire again for a short time period. Various methods have been invented to provide refraction. In one of these methods, each fact is given a unique identifier called a *timetag* when it is entered in working memory. After a rule has fired on a fact, the inference engine will not fire on that fact again because it time stamp has been used.

4.2.9 Top-Level (Command Interpreter & System Development)

Eventually, control is returned to the top-level command interpreter for the user to give further instructions to the rule-based system shell [Giarratano, Riley, 1998]. The top-level is the default mode in which the user communicates with the rule-based system. It is the top-level that accept the new command. The top-level is also the user interface to the shell while a rule-based system application is under development.

4.2.10 The Inference Tree of a Rule-Based System

The chaining process in a rule-based system can be described graphically by an inference tree [Turban, Aronson, 1998]. This inference tree, also called a goal tree or logical tree, provides a schematic view of the inference process. It is similar to a decision tree and influence diagram.

Note that each rule is composed of a premise and a conclusion [Turban, Aronson, 1998]. In building the inference tree, the premises and conclusions are shown as nodes. The branches connect the premises and the conclusions. The operators AND and OR are used to reflect the structures of the rules. The inference tree is constructed upside down: the root is at the top (end) and the branches point downward. The tree starts with “leaves” at the bottom.

There is no deep significance to the construction of such trees; they just provide a better insight into the structure of the rules [Turban, Aronson, 1998]. Inference trees map the knowledge in a convenient manner. By using the tree, one can visualize the process of inference and movement along the branches of the tree. This is called tree traversal. To traverse an AND node, one must traverse all the nodes below it. To traverse an OR node, it is sufficient to traverse just one of the nodes below.

Inference trees are basically composed of clusters of goals [Turban, Aronson, 1998]. Each goal may have sub-goals (children) and a super-goal (parent). Single inference trees are always a mixture of AND nodes and OR nodes; they are often called AND/OR trees (note that the NOT operation is allowed). The AND node signifies a situation in which a goal is satisfied only when all its immediate sub-goals are satisfied. The OR node signifies a situation in which a goal is satisfied when any of its immediate sub-goals is satisfied. When enough sub-goals are satisfied to achieve the primary goal, the tree is said to be satisfied.

The inference engine contains procedures for expressing this process as backward or forward chaining [Turban, Aronson, 1998]. These procedures are organized as a set of instructions involving inference rules. They aim at satisfying the inference tree and collectively contribute to the process of goal (problem) reduction.

The inference tree has another big advantage: It provides a guide for answering the *why* and *how* questions in the explanation process [Turban, Aronson, 1998]. The *how* question is asked by users when they want to know how a certain conclusion has been reached. The computer follows the logic in the inference tree, identifies the goal (conclusion) involved in it and the AND/OR branches, and reports the immediate sub-goals. The *why* question is asked by users when they want to know why the computer requests certain information as input (in backward chaining). To deal with why questions, the computer identifies the goal involved with the computer-generated query, and reports the immediate sub-goals.

4.2.11 Cognitive Architectures

Production systems are also popular in cognitive architectures, i.e., models of human reasoning, such as ACT and SOAR [Russell, Norvig, 1995]. In such systems, the "working memory" of the system models human short-term memory, and the productions are part of long-term memory. Both ACT and SOAR have sophisticated mechanisms for conflict resolution, and for saving the results of expensive reasoning in the form of new productions.

4.2.12 A Final Remark on Rule-Based Reasoning

Inference with rules can be very effective, but there are some obvious limitations [Turban, Aronson, 1998]. Everyone knows the familiar axiom that "there is an exception to every rule".

4.3 Frame-Based Systems

Frame-based systems have been discussed in [Roy, Auger, 2008-A]. The procedures attached to frames give a flexible, organized framework for computation [Brachman, Levesque, 2004]. Reasoning within a frame system usually starts with the system "recognizing" an object as an instance of a generic frame, and then applying procedures triggered by that recognition. Such procedure invocations can then produce more data or changes in the knowledge base that can cascade to other procedure calls. When no more procedures are applicable, the system halts.

More specifically, the basic reasoning loop in a frame system has these three steps [Brachman, Levesque, 2004]:

- A user or external system using the frame system as its knowledge representation declares that an object or situation exists, thereby instantiating some generic frame.
- Any slot fillers that are not provided explicitly but can be inherited by the new frame instance are inherited.
- For each slot with a filler, any IF-ADDED procedure that can be inherited is run, possibly causing new slots to be filled, or new frames to be instantiated, and the cycle repeats.

If the user, the external system, or an attached procedure requires the filler of a slot, then one gets the following behaviour [Brachman, Levesque, 2004]:

- If there is a filler stored in the slot, then that value is returned.
- Otherwise, any IF-NEEDED procedure that can be inherited is run, calculating the filler for the slot, but potentially also causing other slots to be filled, or new frames to be instantiated.

If neither of these produces a result, then the value of the slot is considered to be unknown. Note that in this account, the inheritance of property values is done at the time the individual frame is created, but IF-NEEDED procedures, which calculate property values, are only invoked as required. Other schemes are possible.

This comprises the local reasoning involving a single frame. When constructing a frame knowledge base, one would also think about the global structure of the KB and how computation should produce the desired overall reasoning [Brachman, Levesque, 2004]. Typically, generic frames are created for any major object-type or situation-type required in the problem solving. Any constraints between slots are expressed by the attached IF-ADDED and IF-NEEDED procedures. It is up to the designer to decide whether reasoning should be done in a data-directed or goal-directed fashion.

The slot provides a mechanism for a kind of reasoning called expectation-driven processing Turban, Aronson, 1998]. The frame-based reasoning process is essentially the seeking of confirmation of various expectations. This often just amounts to filling in the slots and verifying that they match the current situation. Empty slots can be filled, subject to certain conditioning, with data that confirm expectations.

In this account, default values are filled in whenever they are available on slots [Brachman, Levesque, 2004]. Actually, the simplest way to specify slot values is by default [Turban, Aronson, 1998]. The default value is attached loosely to the slot so as to be easily displaced by a value that meets the assignment condition. In the absence of information, however, the default value remains attached and expressed. It is worth nothing that in the original, psychological view that first gave rise to frames, defaults were considered to play a major role in scene, situation, and object recognition; it was felt that people were prone to generalize from situations they had seen before, and that they would assume that objects and situations were “typical”, had key aspects taking on their normal default values, unless specific features in the individual case were noticed to be exceptional [Brachman, Levesque, 2004].

Overall, given the constraints between slots that are enforced by attached procedures, one can think of a frame knowledge base as a symbolic “spreadsheet”, with constraints between the objects one cares about being propagated by attached procedures [Brachman, Levesque, 2004].

But the procedures in a frame KB can do a lot more, including invoking complex actions by the system.

4.4 Case-Based Reasoning Systems

The basic idea of case-based reasoning (CBR) is to adapt solutions that were used to solve old problems and use them to solve new problems [Turban, Aronson, 1998]. One variation of this approach is the rule induction method. In rule induction, the computer examines historical cases and generates rules, which then can be chained (forward or backward) to solve problems. Case-based reasoning, on the other hand, follows a different process [Turban, Aronson, 1998]:

- It finds the cases in memory that solved problems similar to the current problem.
- It adapts the previous solution or solutions to fit the current problem, taking into account any differences between the current and previous situations.

A case is the primary knowledge-base element for a case-based reasoning application [Turban, Aronson, 1998]. It is an example of a problem that includes the given information, the methods used, and the results obtained [Stefik, 1995]. It defines a situation or problem, and associates with this situation a proper action. Finding relevant cases involves [Turban, Aronson, 1998]:

- Characterizing the input problem by assigning appropriate features to it.
- Retrieving the cases from memory with those features.
- Picking the case or cases that match the input best.

Case-based reasoning has been proposed as a more psychologically plausible model of the reasoning of an expert than a rule-based model [Turban, Aronson, 1998]. It is basically a processing of the right information retrieved at the right time. So the central problem is the identification of pertinent information whenever needed. This is done in case-based reasoning with the aid of scripts.

Scripts describe a well-known sequence of events [Turban, Aronson, 1998]. In many cases, it is possible to say that reasoning is no more than applying scripts. The more scripts available, the less thinking one needs to do. All that is necessary is to find the right script to use. It has been postulated that given a choice between thinking hard (reworking the problem) and applying an old script, people will choose the script every time.

Scripts can be constructed from historical cases that reflect human experience [Turban, Aronson, 1998]. The experience can be that of the decision makers, or that of others. Case-based reasoning is the essence of how people reason from experience.

4.4.1 The Process of Case-Based Reasoning

The process of case-based reasoning is shown graphically in Fig. 5.

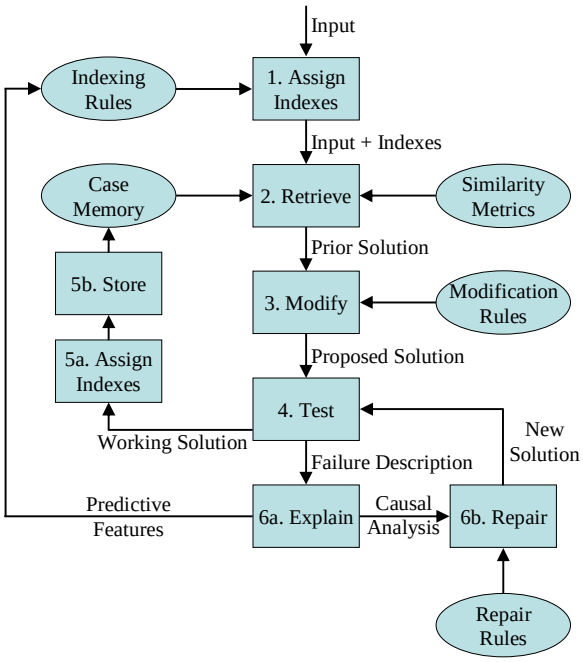


Figure 5: Case-based reasoning flow chart [Turban, Aronson, 1998]

Boxes represent processes and ovals represent knowledge structures. The major steps in the process are [Turban, Aronson, 1998]:

1. Assign indexes. Features of the new event are assigned as indexes characterizing the event.

2. Retrieve. The indexes are used to retrieve a similar past case from memory. The past case contains the prior solution.

3. Modify. The old solution is modified to conform to the new situation, resulting in a proposed solution.

4. Test. The proposed solution is tried out. It either succeeds or fails.

5. Assign and store. If the solution succeeds, then assign indexes and store a working solution. The successful plan is then incorporated into the case memory.

6. Explain, repair and test. If the solution fails, then explain the failure, repair the working solution, and test again. The explanation process identifies the source of the problem. The predictive features of the problem are incorporated into the indexing rules to anticipate this problem in the future. The failed plan is repaired to fix the problem and the revised solution is then tested.

In support of this process are the following types of knowledge structures, represented by ovals in Fig. 5 [Turban, Aronson, 1998]:

- Indexing rules. Indexing rules identify the predictive features in the input that provide appropriate indexes in the case memory. Determining the significant input features is a persistent problem.
- Case memory. Case memory is the episodic memory, which comprises the database of experience.
- Similarity metrics. If more than one case is retrieved from episodic memory, the similarity metrics can be used to decide which case is more like the current situation.
- Modification rules. No old case is going to be an exact match for a new situation. The old case must be modified to fit. One requires knowledge about what kinds of factors can be changed and how to change them.
- Repair rules. Once one identifies and explains an expectation failure, one must try to alter the plan to fit the new situation. Again, one has rules for what kinds of changes are permissible.

4.4.2 Comparison of Case-Based and Rule-Based Reasoning

Table 5 gives a comparison of case-based and rule-based reasoning.

Table 5: Comparison of case-based and rule-based reasoning [Turban, Aronson, 1998]

Criterion	Rule-Based Reasoning	Case-Based Reasoning
Knowledge unit	Rule	Case
Knowledge acquisition units	Rules, hierarchies	Cases, hierarchies
Granularity	Fine	Coarse
Explanation mechanism	Backtrack of rule firings	Precedent cases
Characteristic output	Answer and confidence measures	Answer and precedent cases
Knowledge transfer across problems	High, if backtracking; low if deterministic	Low
Speed as a function of knowledge base size	Exponential, if backtracking; linear, if deterministic	Logarithmic, if index tree balanced
Domain requirements	Domain vocabulary Good set of inference rules Either few rules or rules apply sequentially Domain mostly obeys rules	Domain vocabulary (Large) database of example cases Stability (a modified good solution is probably still good) Many exceptions to rules
Advantages	Flexible use of knowledge	Rapid response

Criterion	Rule-Based Reasoning	Case-Based Reasoning
	Potentially optimal answers	Rapid knowledge acquisition Explanation by examples
Disadvantages	Computationally expensive Long development time Black-box answers	Suboptimal solutions Redundant knowledge base

4.4.3 Advantages of Case-Based Reasoning

Case-based reasoning has several potential benefits [Turban, Aronson, 1998]:

- Knowledge acquisition is improved: easier to build, simpler to maintain, less expensive to develop and support.
- System development time is faster.
- Existing data and knowledge are leveraged.
- Complete formalized domain knowledge (as is required with rules) is not required.
- Experts feel better discussing concrete cases (not general rules).
- Explanation becomes easier. Rather than showing many rules, a logical sequence can be shown.
- Acquisition of new cases is easy (can be automated).
- Learning can occur from both successes and failures.

4.4.4 Case-Based Reasoning Uses and Issues

Case-based reasoning can be used on its own or it can be combined with other reasoning paradigms. Case-based reasoning can be used when [Turban, Aronson, 1998]:

- Domain cannot be formalized with rules because:
  - Domain has weak or unknown causal model.
  - Domain has underdefined terms.
  - Contradictory rules apply in different situations.
  - Domain formalization requires too many rules.
- Application requires complex output (such as battle plans).
- Domain is already precedent-based.
- Domain is dynamic, requiring rapid generation of solutions to new problem types.
- Domain task benefits from records of past solutions, to reuse successful ones and avoid bad ones.

The following issues and questions regarding case-based implementation have been raised [Turban, Aronson, 1998]:

- What makes up a case? How can one represent case memory?
- Automatic case-adaptation rules can be very complex.
- How is memory organized? What are the indexing rules?
- The quality of the results is heavily dependent on the indexes used.
- How does memory function in relevant information retrieval?
- How can one perform efficient search (knowledge navigation) of the cases?
- How can one organize (cluster) the cases?
- How can one design the distributed storage of cases?
- How can one adapt old solutions to new problems? Can one simply adapt the memory for efficient query, depending on context? What are the similarity metrics and the modification rules?
- How can one factor errors out of the original cases?
- How can one learn from mistakes? That is, how does one repair and update the case base?
- The case base may need to be expanded as the domain model evolves, yet much analysis of the domain may be postponed.
- How can one integrate CBR with other knowledge representation and inference mechanisms?
- Are there better pattern-matching methods than the ones currently being used?
- Are there alternative retrieval systems that match the CR schema?

4.4.5 Success Factors for a Case-Based Reasoning System

[Turban, Aronson, 1998] provide a detailed discussion on the success factors for a case-based reasoning system.

5. Non-Monotonic Reasoning

A knowledge base is not of much use to an agent unless it can be expanded [Russell, Norvig, 1995]. Normally, the addition of new axioms to a logic system means that more theorems can be proved because there are more axioms from which theorems can be derived [Giarratano, Riley, 1998]. This property of increasing theorems with increasing axioms is known as monotonicity. As new facts are generated, which is analogous to theorems being proved, a monotonic expert system would keep building up facts.

The use of inference rules to draw conclusions from a knowledge base relies implicitly on this general monotonicity property of certain logics (including propositional and first-order logics) [Russell, Norvig, 1995]. Suppose that a knowledge base *KB* entails some set of sentences. A logic is monotonic if when one adds some new sentences to the knowledge base, all the sentences entailed by the original *KB* are still entailed by the new larger knowledge base. Formally, one can state the property of monotonicity of a logic as follow [Russell, Norvig, 1995]:

$$\text{if } KB_1 \models \alpha \text{ then } (KB_1 \cup KB_2) \models \alpha$$

This is true regardless of the contents of KB_2, i.e., it can be irrelevant or even contradictory to KB_1.

In systems based on first-order logic, one enjoys this property of monotonicity, i.e., if *P* follows from *KB*, then it still follows when *KB* is augmented by *S* [Russell, Norvig, 1995]:

$$\text{if } KB \vdash P \text{ then } (KB \cup S) \vdash P$$

5.1 Retractions and Defeasibility

However, most logical reasoning systems, regardless of their implementation, also have to deal with retractions. There are three reasons for retracting a sentence [Russell, Norvig, 1995]:

1. It may be that a fact is no longer important, and one wants to forget about it to free up space for other purposes.

2. It may be that the system is tracking the current state of the world (without worrying about past situations) and that the world changes.

3. It may be that the system assumed (or determined) that a fact was true, but now wants to assume (or comes to determine) that it is actually false.

Another realistic situation is if a newly introduced axiom partially or completely contradicts a previous axiom.

Unfortunately, a major problem can occur in a monotonic system, in these cases of contradictory facts or if one or more facts become false, as the theorems that had been proved may no longer be

valid [Giarratano, Riley, 1998]. Monotonic systems cannot deal with changes in the truth of axioms and theorems.

A non-monotonic system allows the retraction of facts [Giarratano, Riley, 1998]. In a non-monotonic system, the theorems do not necessarily increase as the number of axioms increases. Inheritance with exceptions is non-monotonic [Russell, Norvig, 1995].

One can switch from first-order logic to a non-monotonic logic that allows one to say that a proposition *P* should be treated as true until additional evidence allows one to prove that *P* is false [Russell, Norvig, 1995]. The term *defeasible* means that any inferences are tentative and may have to be withdrawn as new information becomes available [Giarratano, Riley, 1998].

In order to provide for non-monotonicity, it is necessary to attach a justification or dependency to each fact and rule that explains the reason for belief in it [Giarratano, Riley, 1998]. If a non-monotonic decision is made, then the inference engine can examine the justification of each fact and rule to see if it is still believed, and also to possibly restore excised rules and retracted facts that are believed again.

5.2 Truth-Maintenance Systems (TMS)

In the context of non-monotonic systems, one wants to be able to retract a sentence from the knowledge base without introducing any inconsistencies, and one would like the interaction with the knowledge base as a whole to be efficient [Russell, Norvig, 1995]. The problem of maintaining the correctness or truth of a system when one retracts a proposition *P*, and the process of keeping track of which additional propositions need to be retracted is called *truth maintenance*. It is essential for keeping the correctness or truth by retracting unjustified facts [Giarratano, Riley, 1998]. The simplest approach to truth maintenance is to use chronological backtracking [Russell, Norvig, 1995].

A Truth Maintenance System (TMS) is a program that keeps track of dependencies between sentences so that retraction (and some other operations) will be more efficient [Russell, Norvig, 1995]. It includes sophisticated mechanisms to manipulate the reasons that sentences are believed [Ginsberg, 1993]. A TMS actually performs four important jobs [Russell, Norvig, 1995]:

1. Enabling dependency-directed backtracking, to avoid the inefficiency of chronological backtracking

2. Providing explanations of propositions.

3. Doing default reasoning.

4. Dealing with inconsistencies.

5.2.1 Providing Explanations of Propositions in a TMS

A proof is one kind of explanation; if one asks "Explain why you believe *P* is true", then a proof of *P* is a good explanation [Russell, Norvig, 1995]. If a proof is not possible, then a good explanation is one that involves assumptions.

Technically, an explanation *E* of a sentence *P* is defined as a set of sentences such that *E* entails *P* [Russell, Norvig, 1995]. The sentences in *E* must either be known to be true (i.e., they are in the knowledge base), or they must be known to be assumptions that the problem-solver has made. To avoid having the whole knowledge base as an explanation, one will insist that *E* is minimal, i.e., that there is no proper subset of *E* that is also an explanation. The ability to deal with assumptions and explanations is critical for doing default reasoning.

5.2.2 Dealing with Inconsistencies in a TMSs

If adding *P* to the knowledge base results in a logical contradiction, a TMS can help pinpoint an explanation of what the contradiction is [Russell, Norvig, 1995].

5.2.3 Justification-Based Truth Maintenance Systems (JTMS)

There are several types of TMSs [Russell, Norvig, 1995]. The simplest is the justification-based TMS (JTMS). In a JTMS, each sentence in the knowledge base is annotated with a justification that identifies the sentences from which it was inferred, if any. Some sentences will have more than one justification. Justifications are used to do selective retractions.

In most JTMS implementations, it is assumed that sentences that are considered once will probably be considered again, so rather than removing a sentence from the knowledge base when it loses all justification, one merely marks the sentence as being out of the knowledge base [Russell, Norvig, 1995]. If a subsequent assertion restores one of the justifications, then one marks the sentence as being back in. In this way, the JTMS retains all of the inference chains that it uses, and need not re-derive sentences when a justification becomes valid again.

5.2.4 Assumption-Based Truth Maintenance Systems (ATMS)

The JTMS was the first type of TMS, but the most popular type is the Assumption-Based Truth Maintenance Systems (ATMS) [Russell, Norvig, 1995]. The difference is that a JTMS represents one consistent state of the world at a time. The maintenance of justifications allows one to quickly move from one state to another by making a few retractions and assertions, but at any time only one state is represented. An ATMS represents all the states that have ever been considered at the same time. Whereas a JTMS simply labels each sentence as being in or out, an ATMS keeps track, for each sentence, of which assumptions would cause the sentence to be true. In other words, each sentence has a label that consists of a set of assumption sets. The sentence holds just in those cases where all the assumptions in one of the assumption sets hold.

To solve problems with an ATMS, one can make assumptions in any order one likes [Russell, Norvig, 1995]. Instead of retracting assumptions when one line of reasoning fails, one just asserts all the assumptions one is interested in, even if they contradict each other. One then can check a

particular sentence to determine the conditions under which it holds. On one hand, a sentence that has the empty set as one of its assumption sets is necessarily true; it is true with no assumptions at all. On the other hand, a sentence with no assumption sets is just false.

5.2.5 TMS Computational Complexity

The algorithms used to implement TMSs are complicated [Russell, Norvig, 1995]. The computational complexity of the truth maintenance problem is at least as great as that of propositional inference, i.e., NP-hard. Therefore, one should not expect truth maintenance to be a panacea (except for trivially small problems). But when used carefully (for example, with an informed choice about what is an assumption and what is a fact that can not be retracted), a TMS can be an important part of a logical system.

6. Conclusion

When adopting a knowledge-centric view of situation analysis and information fusion (SAIF), one eventually has to care about knowledge representation and reasoning. This memorandum presented reasoning and inference processes, methods and systems, and discussed the main aspects of this important field.

The memorandum didn't present “solutions”, or findings and results of completed activities. The intent was more to provide a fair description of the technologies for automated reasoning, in order to contribute to setting up a foundational R&D framework for two projects that are just starting under the Applied Research Program (ARP) at Defence R&D Canada: SATAC and CKE-4-MDA. Among other things, the notion of an inference engine and other important concepts such as logical arguments, inference chains and theorems were discussed. Forward chaining and backward chaining were briefly described. The three basic categories of reasoning, i.e., deduction, induction, and abduction were presented, along with other reasoning methods including analogical, generate-and-test, model-based, qualitative, default, and autoepistemic reasoning. Many of the reasoning/inference systems developed and used over the years to achieve automated reasoning in computer systems were discussed. Emphasis was given to logic, rule-based, frame-based, and case-based reasoning systems, as these paradigms and systems are the most relevant to the development of knowledge-based situation analysis support systems. Finally, non-monotonic systems that allow the retraction of facts and truth maintenance systems were presented.

SAIF has a critical role to play in the C2 and intelligence processes. For this reason, numerous ongoing DND and DRDC projects have an important SAIF component. However, despite the importance given to SAIF in many DND strategic documents and projects, the current systems and the associated technology often fail to meet the demanding requirements of the operational decision makers; major science and technology advancements are required to really achieve the full potential of the related enabling technologies and to best serve the operational communities. The effort reported here is one step in this direction; it contributes to the establishment of some solid basis on which a long-term R&D program should be built.

Clearly, adopting a knowledge-centric view of SAIF and developing a complete support system along the approach and framework presented in [Roy, 2007-B] is a very challenging, multidisciplinary enterprise. Project teams will certainly have to seriously deepen the aspects described in this memorandum along the way. Along this line of thought, R&D activities have been and are still currently being conducted to further investigate knowledge representation [Roy, Auger, 2008-A], and also reasoning processes, methods and systems for use in knowledge-based SAIF support systems. Adopting the knowledge-centric view of SAIF requires that knowledge (i.e., expertise) be eventually acquired from the subject matter experts (SMEs) of the different military and public security application domains. In this regard, knowledge and ontological engineering techniques have been and are still being investigated [Roy, Auger, 2008-B]. Knowledge acquisition and validation sessions with SMEs are about to be conducted.

References

[Brachman, Levesque, 2004], Brachman, R. J. and Levesque, H. J., Knowledge Representation and Reasoning, Elsevier, Morgan Kaufmann, San Francisco, CA, 2004.

[Giarratano, Riley, 1998], Giarratano, J. and Riley, G., Expert Systems - Principles and Programming, PWS Publishing Company, Boston, 1998.

[Ginsberg, 1993], Ginsberg, M., Essentials of Artificial Intelligence, Morgan Kaufmann, San Francisco, California, USA, 1993.

[Lambert, 2001], Lambert, D. A., An Exegesis of Data Fusion, accepted for publication in Soft Computing in Measurement and Information Acquisition, Edited L. Reznik and V. Kreinovich. Studies in Fuzziness and Soft Computing, Physica Verlag.

[Merriam-Webster, 2003], Merriam-Webster's 11th Collegiate Dictionary, Software, Version 3.0, 2003.

[Möller, Haarslev, 2003], Möller, R. and Haarslev V., Description Logic Systems, in "Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., The Description Logic Handbook: Theory, Implementation, and Applications", Cambridge University Press, 2003.

[Nardi, Brachman, 2003], Nardi, D. and Brachman, R. J., An Introduction to Description Logics, in "Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., The Description Logic Handbook: Theory, Implementation, and Applications", Cambridge University Press, 2003.

[Roy, 2001], Roy, J., From Data Fusion to Situation Analysis, Proceedings of the Fourth International Conference on Information Fusion (FUSION 2001), Montreal, Canada, August 7-10, 2001.

[Roy, 2007-A], Roy, J., A Knowledge-Centric View of Situation Analysis Support Systems, DRDC Valcartier Technical Report, TR 2005-419, 2007.

[Roy, 2007-B], Roy, J., Holistic Approach and Framework for the Building of Knowledge-Based Situation Analysis Support Systems, DRDC Valcartier Technical Report, TR 2005-420, 2007.

[Roy, Auger, 2008-A], Roy, J. and Auger, A., Knowledge Representation Concepts, Paradigms and Techniques for Use in Knowledge-Based Situation Analysis Support Systems, DRDC Valcartier Technical Memorandum, TM 2006-755, 2008.

[Roy, Auger, 2008-B], Roy, J. and Auger, A., Knowledge and Ontological Engineering Techniques for Use in Developing Knowledge-Based Situation Analysis Support Systems, DRDC Valcartier Technical Memorandum, TM 2006-757, 2008.

[Russell, Norvig, 1995], Russell, S. and Norvig, P., Artificial Intelligence - A Modern Approach, Prentice Hall, New Jersey, 1995.

[Stefik, 1995], Stefik, M., Introduction to Knowledge Systems, Morgan Kaufmann Publishers, San Francisco, California, 1995.

[Turban, Aronson, 1998], Turban, E. and Aronson, J.E., Decision Support Systems and Intelligent Systems, Fifth Edition, Prentice Hall, New Jersey, 1998.

[Waltz, 2003], Waltz, E., Knowledge Management in the Intelligence Enterprise, Artech House, Norwood, MA, 2003.

[White, 1987], White, F. E. Jr., Data Fusion Lexicon, Joint Directors of Laboratories, Technical Panel for C^3, Data Fusion Sub-Panel, Naval Ocean Systems Center, San Diego, 1987.

[Wikipedia, 2008], Wikipedia, SLD_Resolution, http://en.wikipedia.org/wiki/SLD_resolution, 2008.

List of symbols/abbreviations/acronyms/initialisms

AI	Artificial Intelligence
AKAMIA	Advanced Knowledge Acquisition for Maritime Information Awareness
ANS	Artificial Neural System
ARP	Applied Research Program
ASCAT	Analyse de la situation pour le commandant d'armée tactique
ASFI	Analyse de la situation et fusion d'information
ATMS	Assumption-Based Truth Maintenance Systems
C2	Command and Control
C2	Commandement et contrôle
C4ISR	Command and Control, Communication, Computer, Intelligence, Surveillance, and Reconnaissance
CBR	Case-Based Reasoning
CKE-4-MDA	Collaborative Knowledge Exploitation for Maritime Domain Awareness
DL	Description Logic
DND	Department of National Defence
DRDC	Defence R&D Canada
ECCESDM	Exploitation collaborative de la connaissance pour l'éveil situationnel du domaine maritime
ED	Expert du domaine
FOL	First-Order Logic
IA	Intelligence artificielle
IKM	Information and Knowledge Management
INCOMMANDS	Innovative Naval Combat Management Decision Support
JCDS 21	Joint Command Decision Support for the 21st Century
JIIFC	Joint Information and Intelligence Fusion Capability
JTMS	Justification-Based Truth Maintenance Systems
KB	Knowledge Base
KBS	Knowledge-Based System
KR	Knowledge Representation
LF ISTAR	Land Force Intelligence, Surveillance, Target Acquisition and Reconnaissance
LHS	Left-Hand-Side
MDN	Ministère de la Défense Nationale

MSOC	Marine Security Operations Centres
PRA	Programme de recherches appliquées
QR	Qualitative reasoning
R	Relation
RC	Représentation de la connaissance
R&D	Research & Development
RHS	Right-Hand-Side
SA	Situation Analysis
SAIF	Situation Analysis and Information Fusion
SASS	Situation Analysis Support Systems
SATAC	Situation Analysis for the Tactical Army Commander
SAW	Situation Awareness
SME	Subject Matter Expert
SSAS	Systèmes de soutien à l'analyse de la situation
S&T	Science and Technology
TDP	Technology Demonstration Program
TMS	Truth Maintenance System
TM	Technical Memorandum
wff	well-formed formula

Distribution list

Document No.: DRDC Valcartier TM 2006-756

LIST PART 1 − Internal Distribution by Centre (All copies to be distributed in PDF format):

1 - Director General
3 - Document Library
1 - Head/ Intelligence & Information Section
1 - Head/C2 Decision Support Systems Section
1 - Head/System of Systems Section
1 - Head/ Spectral and Geospatial Exploitation Section
1 - Head/ Tactical Surveillance and Reconnaissance Section
1 - Mr Claude Roy (SDA)
1 - Mr Jean Roy (author)
1 - Dr Alain Auger (author)
1 - Mrs Régine Lecocq
1 - Mrs Valérie Lavigne
1 - Mr Yaïves Ferland
1 - Dr Michel Gagnon
1 - Mr Denis Gouin
1 - Dr Luc Pigeon
1 - Mr Alain Bergeron
1 - Mr François Lemieux
1 - Mr Marc-André Morin
1 - Mr Alexandre Bergeron-Guyard
1 - Mr Réjean Lebrun
1 - Dr Pierre Valin
1 - Dr Hengame Irandoust
1 - Dr Patrick Maupin
1 - Mr François Réhaume
1 - Dr Abderezak Benaskeur
1 - Dr Adel Guitouni
1 - Mrs Micheline Bélanger
1 - Dr Anne-Claire Boury-Brisset
1 - Mr Abder Sahi
1 - Mrs Catherine Daigle
1 - Mr Martin Blanchette
1 - Dr Richard Breton
1 - LCol Michel Gareau
1 - Mr Jean-Claude St-Jacques
1 - Dr Marielle Mokhtari
1 - Mr François Bernier
1 - Mr Michel Lizotte
1 - Mr Mario Couture

1 - Mr David Ouellet
1 - Mr Gaetan Thibault
1 - Mr Éric Dorion
1 - Mr Dany Dessureault

TOTAL LIST PART 45

LIST PART 2 − External Distribution by DRDKIM (All copies to be distributed in PDF format)

1 - Director - R&D Knowledge and Information Management

Library and Archives Canada
395 Wellington Street
Ottawa, ON K1A 0N4

1 - Library and Archives Canada

National Defence Headquarter (NDHQ)
101 Colonel By Drive
Ottawa ON K1A 0K2

1 - ADM(S&T) / Associate DG RDP
1 - ADM(S&T) / Directorate Science and Technology (Air)
1 - ADM(S&T) / Directorate Science and Technology (Land)
1 - ADM(S&T) / Directorate Science and Technology (Maritime)
1 - ADM(S&T) / Directorate Science and Technology (Policy)
1 - ADM(S&T) / Directorate Science and Technology (Human Performance)
1 - ADM(S&T) / Directorate Science and Technology (C4ISR)
1 - ADM(S&T) / Centre for Security Science (CSS); Attn: Dr Anthony Ashley – DG

DRDC Atlantic
9 Grove Street
Dartmouth NS B2Y 3Z7

1 - Dr Ross Graham – DG
1 - Mr Jim S. Kennedy
1 - Cdr Siegfried Richardson-Prager (SMO)
1 - Dr Mark McIntyre
1 - Mr Anthony Isenor
1 - Mrs Francine Desharnais
1 - Mrs Liesa Lapinski
1 - Mr Bill Campbell

DRDC Ottawa
3701 Carling Avenue
Ottawa ON K1A 0Z4

1 - Mr Dan Brookes
1 - Mr Chris Helleur, DRDC Ottawa
1 - Mr David DiFilippo

DRDC Toronto
1133 Sheppard Ave W.
Toronto, ON, M3M 3B9

1 - Mrs Carol McCann

TOTAL LIST PART 22

TOTAL COPIES REQUIRED (IN PDF FORMAT): 67

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada - Valcartier, 2459 Pie-XI Blvd. North, Quebec (Quebec), Canada, G3J 1X5

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

Unclassified

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C, R or U) in parentheses after the title.)

Reasoning Processes, Methods and Systems for Use in Knowledge-Based Situation Analysis Support Systems

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Roy, J., Auger, A.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
October 2008	66	18

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

N/A

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
Projects 11bh, 11hg, and 12of.	N/A

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Valcartier TM 2006-756	N/A

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited announcement

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Adopting a knowledge-centric view of situation analysis and information fusion ultimately requires that one cares about knowledge representation and reasoning, i.e., the area of artificial intelligence concerned with how knowledge can be represented symbolically and manipulated in an automated way by programs simulating reasoning. This memorandum reviews reasoning and inference processes, methods and systems. Aspects being discussed include the notion of an inference engine and other related concepts such as logical arguments, inference chains and theorems. Forward chaining and backward chaining are briefly described. Three basic categories of reasoning, i.e., deduction, induction, and abduction are presented, along with other reasoning methods including analogical, generate-and-test, model-based, qualitative, default, and autoepistemic reasoning. Many of the reasoning/inference systems developed and used over the years to achieve automated reasoning in computer systems are discussed. Emphasis is given to logic, rule-based, frame-based, and case-based reasoning systems, as these paradigms and systems are the most relevant to the development of knowledge-based situation analysis support systems. Finally, non-monotonic systems that allow the retraction of facts and truth maintenance systems are presented. The memorandum provides a comprehensive description of reasoning, thereby contributing to the development of a foundational R&D framework for two projects that are just starting at Defence R&D Canada.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Knowledge-Based Systems, Knowledge Representation, Reasoning, Inference, Logic, Rule-Based Systems, Frame-Based Systems, Case-Based Reasoning Systems.

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE R&D DÉFENSE

www.drdc-rddc.gc.ca