Inference engine

From Wikipedia, the free encyclopedia

An Inference Engine is a tool from artificial intelligence. The first inference engines were components of expert systems. The typical expert system consisted of a knowledge base and an inference engine. The knowledge base stored facts about the world. The inference engine applied logical rules to the knowledge base and deduced new knowledge. This process would iterate as each new fact in the knowledge base could trigger additional rules in the inference engine. Inference engines work primarily in one of two modes: forward chaining and backward chaining. Forward chaining starts with the known facts and asserts new facts. Backward chaining starts with goals, and works backward to determine what facts must be asserted so that the goals can be achieved.[1]

Contents

Architecture

The logic that an inference engine uses is typically represented as IF-THEN rules. The general format of such rules is IF <logical expression> THEN <logical expression>. Prior to the development of expert systems and inference engines artificial intelligence researchers focused on more powerful theorem prover environments that offered much fuller implementations of First Order Logic. For example, general statements that included universal quantification (for all X some statement is true) and existential quantification (there exists some X such that some statement is true). What researchers discovered is that the power of these theorem proving environments was also their drawback. It was far too easy to create logical expressions that could take an indeterminate or even infinite time to terminate. For example, it is common in universal quantification to make statements over an infinite set such as the set of all natural numbers. Such statements are perfectly reasonable and even required in mathematical proofs but when included in an automated theorem prover executing on a computer may cause the computer to fall into an infinite loop. Focusing on IF-THEN statements (what logicians call Modus Ponens) still gave developers a very powerful general mechanism to represent logic but one that could be used efficiently with

computational resources. What is more there is some psychological research that indicates humans also tend to favor IF-THEN representations when storing complex knowledge.[2]

A simple example of Modus Ponens often used in introductory logic books is "If you are human then you are mortal". This can be represented in pseudocode as:

Rule1: Human(x) => Mortal(x)

A trivial example of how this rule would be used in an inference engine is as follows. In forward chaining, the inference engine would find any facts in the knowledge base that matched Human(x) and for each fact it found would add the new information Mortal(x) to the knowledge base. So if it found an object called Socrates that was Human it would deduce that Socrates was Mortal. In Backward Chaining the system would be given a goal, e.g. answer the question is Socrates Mortal? It would search through the knowledge base and determine if Socrates was Human and if so would assert he is also Mortal. However, in backward chaining a common technique was to integrate the inference engine with a user interface. In that way rather than simply being automated the system could now be interactive. In this trivial example if the system was given the goal to answer the question if Socrates was Mortal and it didn't yet know if he was human it would generate a window to ask the user the question "Is Socrates Human?" and would then use that information accordingly.

This innovation of integrating the inference engine with a user interface led to the second early advancement of expert systems: explanation capabilities. The explicit representation of knowledge as rules rather than code made it possible to generate explanations to users. Both explanations in real time and after the fact. So if the system asked the user "Is Socrates Human?" the user may wonder why she was being asked that question and the system would use the chain of rules to explain why it was currently trying to ascertain that bit of knowledge: i.e., it needs to determine if Socrates is Mortal and to do that needs to determine if he is Human. At first these explanations were not much different than the standard debugging information that developers deal with when debugging any system. However, an active area of research was utilizing natural language technology to ask, understand, and generate questions and explanations using natural languages rather than computer formalisms.[3]

An inference engine cycles through three sequential steps: match rules, select rules, and execute rules. The execution of the rules will often result in new facts or goals being added to the knowledge base which will trigger the cycle to repeat. This cycle continues until no new rules can be matched.

In the first step, match rules, the inference engine finds all of the rules that are triggered by the current contents of the knowledge base. In forward chaining the engine looks for rules where the antecedent (left hand side) matches some fact in the knowledge base. In backward chaining the engine looks for antecedents that can satisfy one of the current goals.

In the second step select rules, the inference engine prioritizes the various rules that were matched to determine the order to execute them. In the final step, execute rules, the engine executes each matched rule in the order determined in step two and then iterates back to step one again. The cycle continues until no new rules are matched.[4]

Implementations

Early inference engines focused primarily on forward chaining. These systems were usually implemented in the Lisp programming language. Lisp was a frequent platform for early AI research due to its strong capability to do symbolic manipulation. Also, as an interpreted language it offered productive development environments appropriate to debugging complex programs. A necessary consequence of these benefits was that Lisp programs tended to be slower and less robust than compiled languages of the time such as C. A common approach in these early days was to take an expert system application and repackage the inference engine used for that system as a re-usable tool other researchers could use for the development of other expert systems. For example, MYCIN was an early expert system for medical diagnosis and EMYCIN was an inference engine extrapolated from MYCIN and made available for other researchers.[1]

As expert systems moved from research prototypes to deployed systems there was more focus on issues such as speed and robustness. One of the first and most popular forward chaining engines was OPS5 which used the Rete algorithm to optimize the efficiency of rule firing. Another very popular technology that was developed was the Prolog logic programming language. Prolog focused primarily on backward chaining and also featured various commercial versions and optimizations for efficiency and robustness.[5]

As Expert Systems prompted significant interest from the business world various companies, many of them started or guided by prominent AI researchers created productized versions of inference engines. For example, Intellicorp was initially guided by Edward Feigenbaum. These inference engine products were also often developed in Lisp at first. However, demands for more affordable and commercially viable platforms eventually made Personal Computer platforms very popular.

See also

- Action selection mechanism
- Backward chaining
- Expert system
- Forward chaining
- Inductive inference

References

1. Hayes-Roth, Frederick; Donald Waterman; Douglas Lenat (1983). Building Expert Systems.

Addison-Wesley. ISBN 0-201-10686-8.
2. Feigenbaum, Edward; Avron Barr (September 1, 1986). The Handbook of Artificial Intelligence, Volume I. Addison-Wesley. p. 195. ISBN 0201118114.
3. Barzilayt, Regina; Daryl McCullough*, Owen Rambow* Jonathan DeCristofaro$, Tanya Korelsky*, Benoit Lavoie*. "A NEW APPROACH TO EXPERT SYSTEM EXPLANATIONS". USAF Rome Laboratory Report. Cite uses deprecated parameter |coauthors= (help)
4. Griffin, N.L. "A Rule-Based Inference Engine which is Optimal and VLSI Implementable" (PDF). http://www.cs.uky.edu. University of Kentucky. Retrieved 6 December 2013. External link in |work= (help)
5. Sterling, Leon; Ehud Shapiro (1986). The Art of Prolog. Cambridge, MA: MIT. ISBN 0-262-19250-0.

Categories: Expert systems | Decision theory | Inference
