This document discusses ontological engineering and knowledge representation, focusing on general and flexible representations for complex real-world domains. It covers several key concepts:

- **Ontological Engineering:** The process of representing general concepts like events, time, and physical objects, forming an "upper ontology" with general concepts at the top and more specific ones below.
- **Characteristics of General-Purpose Ontology:** Such an ontology should be applicable across various specialized domains, unifying different areas of knowledge for reasoning and problem-solving.
- **Categories and Objects:** Emphasizes organizing objects into categories for knowledge representation, which is crucial for reasoning and making predictions. It explains how categories can be represented using predicates or as reified objects in first-order logic, and how inheritance simplifies knowledge bases through taxonomic hierarchies.
- **Types of Facts about Categories:** Illustrates how to state facts about categories, including object membership, subcategory relations, properties of category members, and properties of a category as a whole.
- **Disjoint Categories, Exhaustive Decompositions, and Partitions:** Defines these concepts for managing relationships between categories, especially when they are not subclasses of each other.
- **Physical Composition:** Discusses the "PartOf" relation for representing that one object is a component of another, noting its transitivity and reflexivity. It also introduces the concept of "bunches" for composite objects with definite parts but no particular structure, especially when dealing with properties like weight.
- **Measurements:** Explains how to represent quantitative measures (like length, diameter, cost) using units and axioms for conversion. It also addresses qualitative measures (like difficulty or beauty), emphasizing their orderability even without numerical values.
- **Objects (Stuff vs. Things):** Distinguishes between "things" (discrete objects like a zebra) and "stuff" (portions of reality that defy obvious individuation, like butter). It highlights that intrinsic properties (density, color) are retained upon subdivision of stuff, while extrinsic properties (weight, shape) are not.
- **Events:** Introduces event calculus as an alternative to situation calculus to handle continuous and simultaneous actions. It explains how fluents and events are reified, and how predicates like Happens, Initiates, and Terminates are used to describe events and their effects on fluents over time intervals.

Lecture 27

This document discusses Truth Maintenance Systems (TMS) and related concepts in artificial intelligence, focusing on how knowledge representation systems handle non-monotonic reasoning and belief revision.

Here's a summary of the key topics:

- **Processes and Time Intervals:** It introduces the distinction between discrete events (with a definite beginning, middle, and end) and process categories (which hold true for any subinterval). It also defines different types of time intervals (moments and extended intervals) and explains interval relations like "Meet," "Before," and "Overlap."
- **Mental Events:** The document explores the need for agents to have knowledge about their own and other agents' beliefs and reasoning processes. It highlights the problem of "referential transparency" in standard logic when dealing with propositional attitudes like

"Believes" and "Knows," and the desire for "referential opacity" in such cases.

- **Non-monotonic Reasoning:** This section explains that human common-sense reasoning often involves "jumping to conclusions" by default, which can be retracted with new evidence. This behavior is called non-monotonicity, where beliefs don't grow monotonically over time.
- **Circumscription:** One approach to non-monotonic reasoning, circumscription, allows specifying predicates to be "as false as possible." It enables drawing default conclusions (e.g., "birds fly") unless specific exceptions are known. It's viewed as a model preference logic where a sentence is entailed if it's true in all preferred models (those with fewer abnormal objects). The "Nixon diamond" example illustrates how it handles conflicting defaults.
- **Default Logic:** Another formalism for non-monotonic reasoning, default logic uses default rules of the form "P: J1,...,Jn/C," meaning if P is true and J1 to Jn are consistent with the knowledge base, then C can be concluded by default. The "Nixon diamond" is also used to demonstrate default logic, leading to multiple possible "extensions" (maximal sets of consequences).
- **Truth Maintenance System (TMS):** This section details the core concept of TMS. It addresses the problem of **belief revision**, where inferred facts might be wrong and need to be retracted without introducing inconsistencies. TMSs are designed to manage these complications efficiently, especially when a retracted sentence P has led to other inferences (Q), and those inferences might have other independent justifications.
- **Simple TMS Approach:** A basic method involves tracking the order of sentences. Retracting a sentence means reverting the system to the state before that sentence was added, then re-adding subsequent sentences. This is simple but impractical for large knowledge bases due to the computational cost.
- **Justification-Based Truth Maintenance System (JTMS):** A more efficient approach where each sentence is annotated with a justification (the set of sentences from which it was inferred). Retracting a sentence P in JTMS only deletes sentences for which P is a member of *all* their justifications. Sentences are marked "in" or "out" rather than entirely deleted to retain inference chains.
- **Assumption-Based Truth Maintenance System (ATMS):** This system enhances efficiency for context switching between hypothetical worlds. Unlike JTMS which represents one state at a time, ATMS represents all considered states simultaneously. For each sentence, it tracks which sets of assumptions would cause it to be true (a label consisting of assumption sets). ATMS also helps in generating explanations for a sentence, which can include assumptions.
- **TMS Node States and Justifications:** In a general TMS, each statement or rule is a "node" and is either "IN" (believed true) or "OUT" (not believed true). IN nodes have at least one valid justification, while OUT nodes have none. Two types of justifications are described:
    - **Support List (SL):** A statement is valid if all "in-nodes" are IN and all "out-nodes" are OUT.
    - **Conditional Proof (CP):** Represents hypothetical arguments, where the consequent is IN if "in-hypothesis" nodes are IN and "out-hypothesis" nodes are OUT.
- **Examples:** The document provides examples of TMS usage, including simple logical deductions and a scheduling scenario involving contradictions.

In essence, the document explains that TMSs are crucial for managing knowledge in systems where beliefs can change, especially in default reasoning and scenarios requiring belief revision, with JTMS and ATMS offering increasingly sophisticated and efficient methods for handling these complexities.
Lecture 28
This file, "Lecture 28: Representation Schemes - 1," discusses various knowledge representation schemes, focusing on Semantic Networks and introducing Description Logics.

Key points include:

- **Knowledge Representation Issues:** Covers topics like generality, specificity, definitions, exceptions, causality, uncertainty, and the choice of schemes (Semantic networks, scripts, frames, stochastic methods, connectionist) and implementation media (Prolog, Lisp, C, Java).
- **Semantic Networks:**
    - Represent knowledge as a graph with labeled nodes (objects/categories) and arcs (relations/associations).
    - Facilitate inheritance reasoning, where properties are inherited from categories to members.
    - Handle default values, allowing for exceptions to general rules.
    - Their simplicity and efficiency for certain inferences are a main attraction, especially compared to logical theorem proving.
    - Limitations include difficulty in representing full first-order logic (negation, disjunction, nested function symbols, existential quantification).
    - Procedural attachment can be used to fill in gaps in expressive power.
    - Examples illustrate how semantic networks can represent human information storage, response times, and properties of snow and ice.
    - Early implementations were used in machine translation (e.g., Quillian's semantic network).
    - Inferences involve searching for relationships between words by finding common concepts or intersection nodes.
- **Description Logics:**
    - Evolved from semantic networks to formalize their meaning while retaining the emphasis on taxonomic structure.
    - Designed to describe definitions and properties of categories more easily than first-order logic.
    - Principal inference tasks include subsumption (checking if one category is a subset of another), classification (checking if an object belongs to a category), and consistency checking.
    - Emphasize tractability of inference, aiming for polynomial-time solutions for subsumption testing.
    - Often lack negation and general disjunction to maintain tractability.
    - The CLASSIC language is presented as a typical example, demonstrating how complex descriptions can be constructed and translated into equivalent first-order sentences.

Lecture 29
This document, "Lecture 29: Representation Schemes - 2," covers several knowledge

representation schemes, including:

- **Description Logics:** These are formal notations designed for describing categories and their properties, emphasizing tractability of inference (like subsumption and classification). The CLASSIC language is given as an example, showcasing its syntax and how it differs from first-order logic.
- **Case Frames:** A verb-oriented approach where links define roles of nouns/phrases in a sentence's action (e.g., agent, object, instrument). An example sentence, "Sarah fixed the chair with glue," is used to illustrate its representation.
- **Conceptual Dependency:** Schank's theory, which uses a set of primitive conceptualizations (ACTs, PPs, AAs, PAs) to build word meaning. It outlines various primitive ACTs (like ATRANS, PTRANS, INGEST) and conceptual syntax rules, with examples of how complex English sentences are represented.
- **Scripts:** Designed by Schank in 1974, these are structured representations describing stereotyped sequences of events in a particular context. They organize conceptual dependency structures and consist of entry conditions, results, props, roles, and scenes. A detailed example of a "RESTAURANT" script is provided.
- **Frames:** Proposed by Minsky in 1975, frames capture implicit connections of information into explicitly organized data structures, similar to classes in object-oriented programming. They consist of slots for identification, relationships to other frames, requirements, procedural information, default information, and new instance information. An example of a "hotel room" frame is given.
- **Conceptual Graphs:** These are finite, connected, bipartite graphs with concept nodes (boxes) and conceptual relation nodes (ellipses). The document explains types, individuals, names, and how to represent various sentences, including those with multiple names for a person or actions involving multiple parts. It also covers operations like generalization, specification (copy, restrict, join, simplify), and how inheritance is implemented. Finally, it touches on representing propositional concepts and logic (conjunction, negation, disjunction) within conceptual graphs.

Lecture 30
This document, "Lecture 30: Acting under uncertainty," discusses how to handle uncertainty in artificial intelligence systems, focusing on utility theory, probability, and Bayes' rule.

Here's a breakdown of the key concepts:

- **Difficulty in handling uncertainty:** Uncertainty arises from partial observability and non-determinism. Traditional logical agents struggle with this due to the complexity of belief states and the need to consider unlikely contingencies, leading to large and complex representations.
- **Utility theory:** This theory addresses the need for agents to have preferences between possible outcomes of plans. It states that every state has a degree of usefulness (utility) to an agent, and agents prefer states with higher utility.
- **Decision theory:** This combines probability theory and utility theory. A decision-theoretic agent chooses actions that yield the highest expected utility, averaged over all possible outcomes. This is known as the principle of maximum expected utility (MEU).
- **Probabilities:**
    - Probabilistic assertions assign probabilities to possible worlds within a sample

space.
  - **Unconditional (prior) probabilities** refer to beliefs in propositions without any other information (e.g., P(Total=11)).
  - **Conditional (posterior) probabilities** refer to beliefs in propositions given some evidence (e.g., P(doubles | Die1 = 5)).
  - The definition of conditional probability is given by: $P(a|b) = P(a \land b) / P(b)$.
  - The **product rule** is derived from this: $P(a \land b) = P(a|b)P(b)$.
- **Inference using full joint probability:**
  - A full joint distribution is a "knowledge base" that can answer all probabilistic queries for discrete variables.
  - **Marginalization (summing out)** is the process of summing probabilities for each possible value of other variables to get the distribution over a subset of variables (e.g., $P(Cavity) = \Sigma P(Cavity, z)$).
  - **Conditioning** is a variant of marginalization using conditional probabilities: $P(Y) = \Sigma P(Y|z)P(z)$.
  - Conditional probabilities like P(cavity | toothache) can be computed by finding the ratio of $P(cavity \land toothache)$ to P(toothache).
  - **Normalization (α)** is a useful shortcut to make computations easier, especially when some prior probabilities are not available. It ensures that the calculated probabilities sum to 1.
  - While powerful, inference using full joint distributions doesn't scale well for large numbers of variables due to exponential complexity (O(2^n)).
- **Independence:**
  - Two variables or propositions are independent if the occurrence of one does not affect the probability of the other (e.g., $P(a|b) = P(a)$ or $P(a \land b) = P(a)P(b)$).
  - Independence can significantly reduce the amount of information needed to specify the full joint distribution, as the distribution can be factored into smaller, independent distributions.
  - However, clean separation of entire sets of variables by independence is rare.
- **Bayes' Rule:**
  - Derived from the product rule: $P(b|a) = [P(a|b)P(b)] / P(a)$.
  - It's useful for computing a diagnostic probability (P(cause | effect)) from causal probabilities (P(effect | cause)), prior probabilities of the cause (P(cause)), and prior probabilities of the effect (P(effect)).
  - In medical diagnosis, for instance, doctors often know P(symptoms | disease) and want to find P(disease | symptoms).
  - Bayes' rule, especially with normalization, provides robustness because causal knowledge (P(symptoms | disease)) is often more stable and unaffected by changes in prior probabilities (like an epidemic) than directly assessed diagnostic probabilities.