Unit 2: Game Playing and Symbolic AI

Syllabus:

Game Playing: Overview and Example Domain, Min-max Search, Adding Alpha-Beta Cutoffs.

First Order Predicate Logic: Unification – Forward Chaining - Backward Chaining – Resolution – **Knowledge Representation:** Ontological Engineering-Categories and Objects – Events –
Mental Events and Mental Objects – Reasoning Systems for Categories – Reasoning with Default Information

Game Playing

5.1 Games

Games represent competitive environments where agents' goals conflict, leading to adversarial search problems.

A game can be defined as a search problem with these elements:

- **Initial State (S_0)**: Setup of the game at the start.
- **PLAYER(s)**: Identifies which player has the current move.
- **ACTIONS(s)**: Returns the set of legal moves available.
- **RESULT(s, a)**: Defines the outcome of a move.
- **Terminal Test (TERMINAL-TEST(s))**: Indicates if the game has ended.
  - **Terminal States**: States where the game concludes.
- **Utility Function (UTILITY(s, p))**: Assigns numeric values based on the game's outcome for each player. In chess, these values are +1, 0, or -1 for win, draw, and loss, respectively.

5.2 Optimal Decisions In Games

Since the optimal solution involves considering the opponent's responses. MAX must develop a contingent strategy, determining its moves based on all possible responses from MIN.

Mirrors AND-OR search algorithm, where MAX represents OR and MIN represents AND.

MAX prefers to move to a state of maximum value, MIN prefers minimum.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax Algorithm

Computes the optimal decision from the current state through recursive calculations of minimax values for each successor state.

Explores the entire depth of the game tree and backs up the values from terminal to root node.

- **Time Complexity**: $O(b^m)$, where b is the branching factor and m is the maximum depth.
- **Space Complexity**: $O(b^m)$ if generating all actions at once, or $O(m)$ if actions are generated one at a time.

5.3 Alpha Beta Pruning

Alpha–Beta Pruning is a technique designed to improve the efficiency of the minimax algorithm by eliminating unnecessary branches in the game tree.

Allows us to disregards branches that can’t affect final decision.

How Alpha–Beta Pruning Works

The key idea behind alpha–beta pruning is to maintain two values:

- **α** (alpha): The best value that the maximizer (MAX) can guarantee at that level or above.
- **β** (beta): The best value that the minimizer (MIN) can guarantee at that level or above.

5.3.1 Move Ordering

The effectiveness of alpha–beta pruning significantly depends on the order in which moves are evaluated. If the best moves are evaluated first, more branches can be pruned, resulting in a much shallower search. Can reduce the number of nodes evaluated to $O(b^{m/2})$ instead of $O(b^m)$, where b is the branching factor and e is the depth of the tree.

Strategies for Move Ordering

1. **Killer Moves**: These are moves that have previously proven effective against a certain position and can be prioritized in future searches.
2. **Transposition Tables**: Storing previously evaluated positions allows for efficient reuse of information. If a position is encountered again, the stored value can be used instead of recalculating it.
3. **Iterative Deepening**: Performing a series of depth-limited searches can inform future searches by highlighting promising moves.

First Order Logic

Built around objects and relations.

The primary difference between propositional and first-order logic lies in the **ontological commitment** (what exists in the world) made by each language - that is, what it assumes about the nature of reality.

- PL assumes there are facts that are true/false.
- FOL assumes objects and certain relations between them, which hold true or not.

A logic can also be characterized by its **epistemological commitments** (what an agent believes about facts) - the possible states of knowledge that it allows with respect to each fact.

Syntax and Semantics

Domain: is the set of objects or domain elements it contains.

Relations: set of tuples of objects that are related.

Total functions: there must be a value for every input tuple.

Term: Logical expression that refers to an object (constant symbols are terms)

Atomic Sentence: formed from a predicate symbol optionally followed by a parenthesized list of terms.

Complex Sentence: Can be made by atomic sentences and logical connectives.

Three kinds of symbols:

- **Constant symbols**: which stand for objects
- **Predicate symbols**: which stand for relations
- **Function symbols**: which stand for functions

$$\begin{aligned}
\text{Sentence} &\rightarrow \text{AtomicSentence} \mid \text{ComplexSentence} \\
\text{AtomicSentence} &\rightarrow \text{Predicate} \mid \text{Predicate}(\text{Term}, \ldots) \mid \text{Term} = \text{Term} \\
\text{ComplexSentence} &\rightarrow (\text{Sentence}) \mid [\text{Sentence}] \\
&\mid\ \neg\text{Sentence} \mid\ \text{Sentence} \land \text{Sentence} \\
&\mid\ \text{Sentence} \lor \text{Sentence} \mid \text{Sentence} \Rightarrow \text{Sentence} \\
&\mid\ \text{Sentence} \Leftrightarrow \text{Sentence} \\
&\mid\ \text{Quantifier Variable}, \ldots\ \text{Sentence} \\
\text{Term} &\rightarrow \text{Function}(\text{Term}, \ldots) \mid \text{Constant} \mid \text{Variable} \\
\text{Quantifier} &\rightarrow \forall \mid \exists \\
\text{Constant} &\rightarrow A \mid X_1 \mid \text{John} \mid \ldots \\
\text{Variable} &\rightarrow a \mid x \mid s \mid \ldots \\
\text{Predicate} &\rightarrow \text{True} \mid \text{False} \mid \text{After} \mid \text{Loves} \mid \text{Raining} \mid \ldots \\
\text{Function} &\rightarrow \text{Mother} \mid \text{LeftLeg} \mid \ldots \\
\textbf{Operator Precedence:} &\ \neg, =, \land, \lor, \Rightarrow
\end{aligned}$$

Quantifiers

To express properties of entire collections instead of enumerating them by name.

- **Universal quantification** ($\forall$): Universal quantification makes statements about every object
  - $\forall x King(x) \Rightarrow Person(x)$
- **Existential quantification** ($\exists$): Similarly, we can make a statement about **some** object in the universe without naming it, by using an existential quantifier.
  - $\exists x Crown(x) \land OnHead(x, John)$

9.2.2 Unification

Process of finding a substitution that makes two logical expressions the same.

The UNIFY algorithm takes two sentences and returns a unifier if it exists.

$$UNIFY(p, q) = \theta where SUBST(\theta, p) = SUBST(\theta, q).$$

Example usage

$$\begin{gathered}
UNIFY(Knows(John, x), Knows(John, Jane)) = \{x/Jane\} \\
UNIFY(Knows(John, x), Knows(y, Bill)) = \{x/Bill, y/John\} \\
UNIFY(Knows(John, x), Knows(y, Mother(y))) = \{y/John, x/Mother(John)\} \\
UNIFY(Knows(John, x), Knows(x, Elizabeth)) = fail.
\end{gathered}$$

The last unification fails because you can't assign two values to x.

This issue can be avoided by **standardizing apart** one of two sentences i.e assign a different variable.

$$UNIFY(Knows(John, x), Knows(x17, Elizabeth)) = x/Elizabeth, x17/John.$$

9.3 Forward Chaining

Start with atomic sentences in the knowledge base and apply **Modus Ponens,** until no further inferences can be made.

Modus ponens is a rule of inference that states that if P and P implies Q are true, then Q must also be true. It is a deductive rule of inference, meaning that it is guaranteed to produce true conclusions from true premises.

9.4 Backward Chaining

Knowledge Representation

12.1 Ontological Engineering

Representing abstract concepts like Events, Time, Physical Objects and Beliefs

Ontology: categorization of entities and specifying/identifying relationships between them

Upper Ontology: the general framework of concepts. (Because of drawing graphs with general concepts at top and more specific concept below)

Difficulty in any generalization is that there are excepts or they hold true only to a degree.

Characteristics of General Purpose Ontology:

- Should be applicable in any special purpose domain, so no representation issues can be finessed/ignored
- Different areas of knowledge should be unified

Ontologies can be created in four ways:

1. By a team of ontologist/logicians
2. Importing categories, attributes and values from existing dbs
3. Parsing text docs and extracting info from them
4. Enticing unskilled amateur to enter common sense knowledge

12.2 Categories and Objects

Categories: are essential in organizing objects and reasoning about them. Interaction takes place object level, but reasoning takes place at the level of categories.

In first-order logic, categories can be represented as **predicates** or **objects**.

For instance, instead of just saying $Basketball(b)$, we can reify the category as an object $Basketballs$ and express membership as $b \in Basketballs$. Categories can also have relations like subcategories (e.g., $Basketballs \subset Balls$).

Inheritance: simplifies knowledge representation.

For example, by stating that all food is edible and that apples are a subclass of food, we infer that all apples are edible. This is an example of **taxonomic hierarchy**, where subclass relations create a structured hierarchy, such as the Dewey Decimal system in libraries.

In first-order logic, facts about categories can be expressed as:

- **Object membership in a category**: $BB_9 \in Basketballs$
- **Subclass relations**: $Basketballs \subset Balls$
- **Properties of all members**: $x \in Basketballs \Rightarrow Spherical(x)$
- **Recognition of members by properties**: $Orange(x) \land Round(x) \land Diameter(x) = 9.5 \Rightarrow x \in Basketballs$
- Category properties: $Dogs \in DomesticatedSpecies$

Disjoint categories have no members in common, and **exhaustive decomposition** occurs when categories completely divide a larger group.

A **partition** is a disjoint exhaustive decomposition, such as $Partition(Males, Females, Animals)$.

Categories can also be defined by **necessary and sufficient conditions**. For example, a bachelor is defined as an unmarried adult male:

$x \in Bachelors \Leftrightarrow Unmarried(x) \land x \in Adults \land x \in Males$

12.2.1 Physical Composition

Objects are often parts of other objects, forming $PartOf$ hierarchies similar to category hierarchies. For example, $PartOf(Bucharest, Romania)$ and $PartOf(Romania, Europe)$. This relation is **transitive** (if one object is part of another, and that object is part of a third, then the first is part of the third) and **reflexive** (an object is part of itself).

Composite objects are characterized by their parts. For example, a **biped** has two legs and a body:

$$\begin{aligned}\text{Biped}(a) \Rightarrow \exists l_1, l_2, b\, &\text{Leg}(l_1) \wedge \text{Leg}(l_2) \wedge \text{Body}(b) \wedge \\ &\text{PartOf}(l_1, a) \wedge \text{PartOf}(l_2, a) \wedge \text{PartOf}(b, a) \wedge \\ &\text{Attached}(l_1, b) \wedge \text{Attached}(l_2, b) \wedge \\ &l_1 \neq l_2 \wedge [\forall l_3\, \text{Leg}(l_3) \wedge \text{PartOf}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)]\end{aligned}$$

Part Partition: objects without structured parts, like a **bunch** of apples, are handled by the concept of $BunchOf$. For instance, the apples in a bag form a bunch, distinct from the abstract set $Apples$.

$BunchOf(Apple1, Apple2, Apple3)$ denotes the composite object made from these apples.

12.2.2 Measurements

Measures: are values assigned to properties of objects. They can be ordered and compared using $>$, $<$ $etc.$

Measures can be used to describe objects as follows

$$\begin{gathered} Diameter(Basketball12) = Inches(9.5) \\ ListPrice(Basketball12) = \$(19) \\ d \in Days \Rightarrow Duration(d) = Hours(24) \end{gathered}$$

12.2.3 Objects: Things and Stuff

The real world contains primitive objects and composite objects (made from primitives).

Individuation: division into distinct objects.

Stuff: The reality that prevents further individuation. $(Eg.\ Butter)$

Properties:

- Intrinsic: Belongs to the very substance of the object.
- Extrinsic: Belongs to the object as a whole.

The category stuff is the most general substance category, specifying on intrinsic properties.

The category thing is the most general discrete object category, specifying on extrinsic properties.

12.3 Events

Situation Calculus: Describes a world in which actions are discrete, instantaneous and happens one at a time. It cannot describe two actions happening simulataneously.

Event Calculus: Based on points of time rather than situation.

In event calculus, **fluents** (facts that can change over time) and **events** are reified (treated as objects).

Fluents: are defined based on events that initiate or terminate them.

A fluent like $At(Shankar, Berkeley)$ refers to Shankar being in Berkeley, but to assert its truth at a specific time, we use the predicate $T(fluent, time)$ (e.g., $T(At(Shankar, Berkeley), t)$).

A fluent doesn't hold true if it is terminated by an event and not made true (restored) by another event.

$$\begin{gathered} Happens(e, (t1, t2)) \wedge Initiates(e, f, t1) \wedge \neg Clipped(f, (t1, t)) \wedge t1 < t \Rightarrow T(f, t) \\ Happens(e, (t1, t2)) \wedge Terminates(e, f, t1) \wedge \neg Restored(f, (t1, t)) \wedge t1 < t \Rightarrow \neg T(f, t) \end{gathered}$$

where Clipped and Restored are defined by

$$\begin{gathered} Clipped(f, (t1, t2)) \Leftrightarrow \exists e, t, t3 Happens(e, (t, t3)) \wedge t1 \leq t < t2 \wedge Terminates(e, f, t) \\ Restored(f, (t1, t2)) \Leftrightarrow \exists e, t, t3 Happens(e, (t, t3)) \wedge t1 \leq t < t2 \wedge Initiates(e, f, t) \end{gathered}$$

A fluent holds over an interval if it holds on every point within the interval.

$$T(f, (t1, t2)) \Leftrightarrow [\forall t (t1 \leq t < t2) \Rightarrow T(f, t)]$$

Events: are instances of event categories.

For example, the event of Shankar flying from San Francisco to Washington can be represented as:

$$E1 \in Flyings \land Flyer(E1, Shankar) \land Origin(E1, SF) \land Destination(E1, DC)$$

Key predicates in event calculus include:

- $T(fluent, time)$: Fluent is true at time t.
- $Happens(event, interval)$: Event occurs over the interval i.
- $Initiates(event, fluent, time)$: Event causes fluent to become true at time $t1$.
- $Terminates(event, fluent, time)$: Event causes fluent to cease being true at time $t1$.
- $Clipped(fluent, interval)$: Fluent ceases to be true during interval i.
- $Restored(fluent, interval)$: Fluent becomes true during interval i.

Event calculus can be extended to represent:

- **Simultaneous events:** Ex. Two people on the sea saw
- **Continuous events:** Ex. Level of water rising in a bucket
- **Exogenous events** : Events caused by external factors, like wind moving an object.

12.3.1 Processes

Discrete events have a clear structure with a beginning, middle, and end. For example, Shankar's trip from San Francisco to Washington is a discrete event. If interrupted, it becomes a different event.

Process categories, or **liquid events**, are different. They can happen over any subinterval of time. For example, during any small interval of Shankar's flight, the event is still part of the broader category of $Flyings$.

This is analogous to the distinction between **spatial substances or stuffs** (like butter) and **individual objects** or things (like a chair). Liquid events are also referred to as **temporal substances**.

12.3.2 Time Intervals

Event calculus enables reasoning about **time intervals**, which can be **moments** (zero duration) or **extended intervals**.

For example, a moment is defined as having zero duration: $i \in Moments \Leftrightarrow Duration(i) = Seconds(0)$

Time intervals are represented by a start and end time. Functions like $Begin(i)$ and $End(i)$ pick the earliest and latest moments in an interval, and $Duration(i)$ gives the length of the interval.

Allen's Interval Relations describe how intervals relate to one another. Some key relations include:

- $Meet(i, j)$: Interval i ends when interval j begins.
- $Before(i, j)$: Interval i ends before interval j begins.
- $After(j, i)$: Interval j ends after interval i begins.
- $During(i, j)$: Interval i occurs entirely within interval j.
- $Overlap(i, j)$: Interval i begins before, and ends after, the start of interval j.
- $Begins(i, j)$: Interval i begins at the same time as interval j.
- $Finishes(i, j)$: Interval i ends at the same time as interval j.
- $Equals(i, j)$: Interval i begins and ends at the same times as interval j.

$$\begin{gathered} Interval(i) \Rightarrow Duration(i) = (Time(End(i)) - Time(Begin(i))) \\ Time(Begin(AD1900)) = Seconds(0) \\ Time(Begin(AD2001)) = Seconds(3187324800) \\ Time(End(AD2001)) = Seconds(3218860800) \\ Duration(AD2001) = Seconds(31536000) \end{gathered}$$

For example, to say that the reign of George VI immediately preceded Elizabeth II: $Meets(ReignOf(GeorgeVI), ReignOf(ElizabethII))$

12.3.3 Fluents and Objects

Objects can be seen as **generalized events** that span space and time.

For instance, the **USA** can be thought of as an event that began in 1776 and continues today.

Fluents can describe properties of these objects that change over time, like the **President of the USA**.

To represent changing properties, we can use expressions (fluents) like:
$T(Equals(President(USA), GeorgeWashington), AD1790)$

This asserts that George Washington was President in 1790. The function $Equals$ is used instead of logical identity ($=$) because identity doesn't change over time—rather, it is specific to time intervals. Thus, the **President(USA)** is a single object that consists of different people (Washington, Adams, etc.) over different periods.

12.4 Mental Events and Mental Objects

Agents can have **beliefs** and deduce new beliefs, but they typically lack knowledge about their own beliefs and reasoning processes. This self-awareness is important for controlling inference.

Agents also need knowledge of **other agents' knowledge.** To manage this, a model is needed for **mental objects** (knowledge, beliefs) and **mental processes** (deduction, reasoning). This model doesn't need precise details but should allow reasoning about knowledge and beliefs.

Propositional attitudes: represent the relationships/attitudes agents have toward mental objects, such as **Believes**, **Knows**, **Wants**, or **Informs**. These attitudes do not behave like typical predicates in logic.

Referential Transparency: Terms do not matter, the object that the term names does.

For instance, if Lois knows that Superman can fly ($Knows(Lois, CanFly(Superman))$), and if Superman is Clark Kent, regular logic would incorrectly conclude that Lois knows Clark can fly.

This is because regular logic follows **referential transparency**, where the specific term referring to an object doesn't matter. However, for beliefs and knowledge, the **terms do matter**—Lois may know Superman can fly but not that Clark can.

Modal logic: addresses the issue of referential opacity by introducing **modal operators**.

Modal: A word that expresses a modality - qualifies a statement

For example, "A knows P" is written as $K_A P$, where K_A is the operator for knowledge. Modal logic works with **possible worlds**, representing different states of reality. In this framework, an agent's knowledge is modeled by **accessibility relations** between possible worlds. For example, if an agent knows a certain fact in world $w0$, then all worlds accessible from $w0$ must agree with that fact.

In modal logic, a knowledge statement $K_A P$ is true in world w if P is true in **every world** accessible from w. For example, if Lois doesn't know if Superman is Clark Kent, we can express that she knows that Clark knows his own identity:

$$K_{Lois}[K_{Clark} Identity(Superman, Clark) \lor K_{Clark} \neg Identity(Superman, Clark)]$$

Ambiguity in Knowledge Statements

Modal logic helps resolve ambiguities in statements about knowledge. For example, the sentence "Bond knows that someone is a spy" can be interpreted in two ways:

- **Specific spy known**: There is a specific individual Bond knows to be a spy ($\exists x K_{Bond} Spy(x)$).
- **General knowledge**: Bond knows that there is at least one spy, without knowing who the spy is ($K_{Bond} \exists x Spy(x)$).

Modal logic includes axioms for knowledge:

- **Deduction**: If an agent knows P and P implies Q, then the agent knows Q: $(K_A P \land K_A(P \Rightarrow Q)) \Rightarrow K_A Q$
- **Truth**: If an agent knows P, then P must be true: $K_A P \Rightarrow P$
- **Introspection**: If an agent knows something, they also know that they know it: $K_A P \Rightarrow K_A(K_A P)$

A challenge in modal logic is the assumption of **logical omniscience**—the idea that agents know all logical consequences of what they know. This is problematic because real-world agents are limited in their reasoning capabilities.

12.5 Reasoning Systems For Categories

Categories are fundamental components of large-scale knowledge representation systems. Two key approaches to organizing and reasoning with categories include **semantic networks** and **description logics**.

12.5.1 Semantic Networks

Semantic networks are graphical tools that represent relationships between objects and categories using nodes and edges. Provide a way to visualize a knowledge base and perform efficient reasoning about category membership.

- **Nodes** represent objects or categories.
- **Edges** represent relations between objects or categories (e.g., membership or subset relationships).

Inheritance reasoning is a key feature of semantic networks, allowing objects to inherit properties from categories via links. For example, if the category "Persons" has a property of having two legs, then "Mary," as a member of "Persons," inherits this property.

However, **multiple inheritance** (when an object belongs to more than one category) can complicate this reasoning. This can lead to conflicting properties if multiple categories have different values for the same property.

Advantages

- **Efficiency**: Inheritance-based reasoning is simpler and faster than general logical theorem proving.
- **Graphical Notation**: It is easier to visualize and understand the reasoning process by following links between categories.

Limitations

- **Binary Relations Only**: Links in semantic networks typically represent only binary relations. To represent more complex, n-ary relations (e.g., "Fly(Shankar, NewYork, NewDelhi, Yesterday)"), the proposition must be reified into an event category, as shown in Figure 12.6.
- **Limited Expressiveness**: Semantic networks do not support negation, disjunction, or complex quantification. Extending the notation to include these features would make the system more complex and reduce the benefits of simplicity and efficiency.
- **Default Values**: Semantic networks can represent default values for categories, which can be overridden by specific values. For example, "John" may have only one leg despite being a "Person" with the default property of having two legs.

12.5.2 Description Logics

Description logics (DLs) are formal languages designed to describe the definitions and properties of categories in a more structured way than semantic networks.

They evolved from semantic networks to provide formal semantics while retaining a focus on taxonomic hierarchy and category reasoning.

Key reasoning tasks in description logics include:

- **Subsumption**: Checking if one category is a subset of another by comparing their definitions.
- **Classification**: Checking whether an object belongs to a category.
- **Consistency**: Determining whether a category's membership criteria are logically consistent (i.e., if it's possible for objects to belong to the category).

Syntax of Description Logics

In **CLASSIC**, a common description logic, categories are defined using a structured syntax that allows operations like conjunction (And), role restrictions ($All, AtLeast, AtMost$), and individual constraints ($Fills, SameAs$).

For instance, to define the category of **bachelors** as unmarried adult males, we would write: $Bachelor = And(Unmarried, Adult, Male)$

This is equivalent to the first-order logic statement: $Bachelor(x) \Leftrightarrow Unmarried(x) \land Adult(x) \land Male(x)$

More complex descriptions are also possible. For example, to describe men with at least three sons who are unemployed and married to doctors, and at most two daughters who are professors in physics or math, you would write:

$$\begin{gathered} And(Man, AtLeast(3, Son), AtMost(2, Daughter), \\ All(Son, And(Unemployed, Married, All(Spouse, Doctor))), \\ All(Daughter, And(Professor, Fills(Department, Physics, Math)))) \end{gathered}$$

Tractability and Inference

One of the major focuses of description logics is ensuring that reasoning tasks (like subsumption testing) can be performed efficiently, typically in **polynomial time**. In contrast, reasoning in standard first-order logic can be unpredictable and computationally expensive, often requiring manual optimization by the user.

While description logics provide **efficient subsumption testing** in practice, some descriptions can still lead to **exponential-time** complexity in the worst case.

12.6 Reasoning with Default Information

Defaults are useful for making assumptions when explicit information is unavailable, but they can be overridden by more specific information. Focused on **nonmonotonic reasoning**.

12.6.1 Circumscription and Default Logic

In traditional logic, reasoning is **monotonic**: once something is proven true, it remains true even when new facts are added. However, **nonmonotonic reasoning**—which allows the retraction of conclusions when new evidence is presented—is more reflective of how humans reason in everyday situations.

Circumscription

Circumscription is a form of nonmonotonic reasoning that can be viewed as a more powerful version of the **closed-world assumption**. It assumes that certain predicates are **"as false as possible"**, meaning they are false for all objects except those for which they are explicitly known to be true.

For instance, to express the default rule that **birds fly**, we introduce an abnormality predicate, $Abnormal1(x)$, and express the rule as: $Bird(x) \land \neg Abnormal1(x) \Rightarrow Flies(x)$

Under circumscription, the predicate Abnormal1 is minimized, allowing the conclusion that birds fly unless there is explicit evidence that a specific bird is abnormal (e.g., $Abnormal1(Tweety)$).

Circumscription can be seen as a **model preference logic**, where one model is preferred over another if it has fewer abnormal objects.

For example, in the **Nixon diamond** problem, Richard Nixon is both a $Quaker$ (and thus a pacifist by default) and a $Republican$ (and thus not a pacifist by default).

$$\begin{gathered} Republican(Nixon) \land Quaker(Nixon). \\ Republican(x) \land \neg Abnormal2(x) \Rightarrow \neg Pacifist(x). \\ Quaker(x) \land \neg Abnormal3(x) \Rightarrow Pacifist(x). \end{gathered}$$

Circumscription allows reasoning to remain **agnostic** about Nixon's pacifism unless further information is provided.

Default Logic

Default logic is another formalism for nonmonotonic reasoning that allows the use of **default rules**. A default rule has the form: $Bird(x) : Flies(x)/Flies(x)$

This rule means: if $Bird(x)$ is true and $Flies(x)$ is consistent with the current knowledge base, then conclude $Flies(x)$ by default.

The general form of a default rule is: $P : J1, J2, ..., Jn/C$

Where:

- P: Prerequisite (must be true for the rule to apply).
- $J1, ..., Jn$: Justifications (if any of these are false, the conclusion cannot be drawn).
- C : Conclusion (can be inferred if the justifications hold).

For example, the **Nixon diamond** problem in default logic can be represented by two default rules:

$$\begin{gathered} Republican(Nixon) : \neg Pacifist(Nixon)/\neg Pacifist(Nixon) \\ Quaker(Nixon) : Pacifist(Nixon)/Pacifist(Nixon) \end{gathered}$$

Extensions in default logic represent maximal sets of conclusions that can be inferred from the knowledge base, considering the default rules. In the Nixon example, there would be two possible extensions: one where Nixon is a pacifist and one where he is not.

Challenges with Nonmonotonic Reasoning

- **Nonmodularity**: It is difficult to decide which default rules should be included in the knowledge base.
  - For example, if "Cars have four wheels" is false for a specific car, it raises the question of what it means to have that as a default rule.
- **Decision Making**: Using default beliefs to make decisions involves trade-offs, especially when decisions must be made under uncertainty.
  - For instance, the default belief that "my brakes are always OK" may be reasonable most of the time, but not when driving a heavily loaded truck down a steep hill. This has led to the idea of integrating default reasoning with **probability theory** or **utility theory** to account for decision-making under uncertainty.

12.6.2 Truth Maintenance Systems

When reasoning with defaults and nonmonotonic logic, there are times when previously inferred facts need to be retracted.

This process is managed by **truth maintenance systems (TMS)**, which handle the **belief revision** process—updating or retracting beliefs when new information contradicts previous inferences.

Belief Revision

When a new fact contradicts an existing belief, a **belief revision** process is triggered.

For example, if a knowledge base contains a fact P, and new evidence suggests ¬P, the system must retract P and any conclusions that depend on it. This can be challenging if those conclusions were derived from multiple justifications.

Justification-Based Truth Maintenance Systems (JTMS)

A **Justification-Based Truth Maintenance System (JTMS)** keeps track of the justifications for each fact in the knowledge base.

Each fact is associated with a set of justifications, and if one justification is invalidated, the system re-evaluates whether the fact should remain in the knowledge base.

For example, if Q was inferred from P and $P \Rightarrow Q$, then Q has the justification $P, P \Rightarrow Q$. If P is retracted, the system checks if any other justifications for Q exist before deciding whether to remove Q.

- **Efficiency**: Unlike simpler approaches that revert the knowledge base to a previous state, JTMS only retracts facts that depend entirely on the retracted fact, making it more efficient.
- **Marking Sentences**: Instead of deleting a sentence entirely, JTMS "marks" it as out of the knowledge base. If a justification for the sentence is restored later, the sentence can be reinstated without needing to be re-derived.

JTMS ensures that the retraction of incorrect information is handled efficiently, even in large systems where many facts are interconnected.