AIES — Unit IV Notes

Planning and Expert Systems

Table of Contents

---

1. Planning — Introduction

1. **Planning** is the process by which an AI agent decides the sequence of actions needed to achieve a desired goal — it bridges the gap between the current state of the world and the goal state.
2. A **plan** is an ordered set of actions (operators) that, when executed, transforms the initial state into the goal state; each action has **preconditions** (what must be true before the action) and **effects** (add/delete lists describing what changes).
3. Planning operates on a **state-space representation** — the world is described by a set of logical predicates, and actions change which predicates are true or false.
4. The **STRIPS** (Stanford Research Institute Problem Solver) representation is the classic planning formalism — each action is defined by its precondition list, add list, and delete list.
5. Key planning challenges include handling large state spaces, incomplete knowledge, and selecting the optimal sequence of actions from among many possible plans.

Classic STRIPS Example — Blocks World:

Predicate	Meaning
ON(A, B)	Block A is on Block B
ONTABLE(A)	Block A is on the table
CLEAR(A)	Nothing is on top of Block A
HOLDING(A)	Robot arm is holding Block A
ARMEMPTY	Robot arm holds nothing

PICKUP(x) — Pick up a block from the table:

- Preconditions: ONTABLE(x), CLEAR(x), ARMEMPTY
- Add: HOLDING(x)
- Delete: ONTABLE(x), CLEAR(x), ARMEMPTY

PUTDOWN(x) — Put a block on the table:

- Preconditions: HOLDING(x)
- Add: ONTABLE(x), CLEAR(x), ARMEMPTY
- Delete: HOLDING(x)

STACK(x, y) — Stack block x on block y:

- Preconditions: HOLDING(x), CLEAR(y)
- Add: ON(x, y), CLEAR(x), ARMEMPTY
- Delete: HOLDING(x), CLEAR(y)

UNSTACK(x, y) — Remove block x from block y:

- Preconditions: ON(x, y), CLEAR(x), ARMEMPTY
- Add: HOLDING(x), CLEAR(y)
- Delete: ON(x, y), CLEAR(x), ARMEMPTY

---

2. Forward Planning (Forward Chaining / Data-Driven)

1. **Forward planning** starts from the **known initial state** and applies applicable operators (actions whose preconditions are satisfied) to generate new states, progressing forward until the goal state is reached.
2. At each step the planner identifies all actions whose **preconditions are satisfied** by the current state, selects one (using search strategy or heuristic), applies it, and obtains a new

state — this continues until the goal is achieved.

3. Forward planning is also called **progression planning** — it is equivalent to forward chaining in a rule-based inference engine, where facts drive the firing of rules.
4. **Advantage:** Can discover anything derivable from the current facts; useful when the initial state is well-known and the goal state is loosely defined.
5. **Disadvantage:** It can be expensive and inefficient because it may explore many irrelevant states that do not lead toward the goal; the search space can be enormous.

Algorithm:

```
Repeat:
  Apply all applicable rules to current facts
  Each rule firing adds new facts
Until: Goal state is reached or no new facts can be added
```

Example:

```
Facts:   F1: Ungee gives milk
         F2: Ungee eats meat
         F3: Ungee has hoofs

Rules:   R1: If X gives milk → X is a mammal
         R2: If X is a mammal AND eats meat → X is a carnivore
         R3: If X is a carnivore AND has hoofs → X is an ungulate

Trace:   R1 + F1 → Ungee is a mammal (F4)
         R2 + F4 + F2 → Ungee is a carnivore (F5)
         R3 + F5 + F3 → Ungee is an ungulate ✓
```

---

3. Backward Planning (Backward Chaining / Goal-Driven)

1. **Backward planning** starts from the **desired goal state** and works backwards — it identifies which actions could have produced the goal, then sets up their preconditions as sub-goals to be achieved.
2. The process recursively decomposes goals into sub-goals until all sub-goals match facts already present in the initial state — this is called **regression planning**.
3. Backward planning is efficient when the goal is precisely defined but the initial state is complex, because it focuses search only on facts that are relevant to proving the goal.
4. **Advantage:** Avoids exploring irrelevant states; more focused and often more efficient than forward planning for well-defined goals; the trace of rules directly provides an explanation of the solution.

5. **Disadvantage:** Requires known goals upfront; may get stuck in infinite loops if sub-goals recursively reference each other (needs loop detection); difficult when many rules can achieve the same goal.

Example:

```
Goal:    G1: Is Ungee an ungulate?

G1 matches conclusion of R3 → Sub-goals:
    G2: Ungee is a carnivore
    G3: Ungee has hoofs

G3 matches Fact F3 → TRUE ✓

G2 matches conclusion of R2 → Sub-goals:
    G4: Ungee is a mammal
    G5: Ungee eats meat

G5 matches Fact F2 → TRUE ✓
G4 matches conclusion of R1 → Sub-goal:
    G6: Ungee gives milk
G6 matches Fact F1 → TRUE ✓

All sub-goals satisfied → Ungee IS an ungulate ✓
```

Forward vs Backward Chaining Comparison:

Feature	Forward Chaining	Backward Chaining
Start point	Initial state (known facts)	Goal state
Direction	Data → Goal	Goal → Data
Use when	Goal is unknown; explore all possibilities	Goal is specific
Efficiency	Can be expensive (explores irrelevant states)	More focused, efficient
Explanation	Trace of applied rules	Rule trace provides justification

---

4. Goal Stack Planning

1. **Goal Stack Planning** uses an explicit **stack data structure** to manage goals and sub-goals — goals are pushed onto the stack, and the planner works on the top-most goal first (LIFO order).

2. The planner pops the top goal from the stack; if it is already satisfied by the current state, it is simply discarded; if not, an action that achieves it is selected and its preconditions are pushed as new sub-goals above it.
3. This **depth-first recursive decomposition** continues until the stack is empty, meaning all goals have been achieved in the correct order.
4. Goal stack planning can suffer from the **Sussman Anomaly** — when two goals interfere with each other (achieving goal A undoes goal B, and vice versa), simple goal stack planning loops or fails; this requires interleaving of sub-plans.
5. The key advantage of goal stack planning is its simplicity and the clear hierarchical decomposition of complex goals into manageable sub-goals; it is a precursor to more advanced planning approaches like HTN planning.

Goal Stack Planning — Step-by-Step Process:

```
Initial State: ON(C,A), ONTABLE(A), ONTABLE(B), CLEAR(C), CLEAR(B), ARMEMPTY
Goal: ON(A,B), ON(B,C)   (A on B, B on C)

Step 1: Push [ON(A,B), ON(B,C)] — main goal
Step 2: Pop ON(A,B) — not satisfied, find action STACK(A,B)
Step 3: Push preconditions: HOLDING(A), CLEAR(B) → resolve each
Step 4: Achieve HOLDING(A) → PICKUP(A) → requires ONTABLE(A), CLEAR(A), ARMEMPTY
...continue until stack is empty
```

---

5. Hierarchical Planning

1. **Hierarchical Planning** (also called Hierarchical Task Network or HTN Planning) decomposes high-level abstract tasks into progressively more detailed sub-tasks, eventually reaching primitive actions that can be directly executed.
2. Tasks are organised at **multiple levels of abstraction** — the top level contains the overall goal described in abstract terms; each abstract task is expanded into a network of sub-tasks until only executable primitive actions remain.
3. A key concept is the **Task Network** — a partially ordered set of tasks (abstract or primitive) connected by ordering constraints; the planner refines the network level by level using **decomposition methods**.
4. Hierarchical planning is particularly powerful for **complex real-world domains** because it allows planners to focus on high-level strategy first and then handle low-level details; it is the basis for many practical planning systems used in robotics and game AI.
5. Advantages over flat planning include: dramatically reduced search space (high-level decisions prune low-level choices), natural encoding of domain knowledge, easier human understanding of plans, and modular reuse of sub-plans.

Hierarchical Planning — Levels of Abstraction:

```
Level 0 (Abstract):    Travel from City A to City B
Level 1 (Refined):     Book-Flight → Board-Aircraft → Fly → Land → Collect-Luggage
Level 2 (Primitive):   Go-to-Airport, Check-in, Pass-security, Board, ...
```

Diagram:

```
[Main Goal]
     ↓ decompose
[Sub-task 1]  [Sub-task 2]  [Sub-task 3]
     ↓              ↓              ↓
[Primitives] [Primitives] [Primitives]
```

---

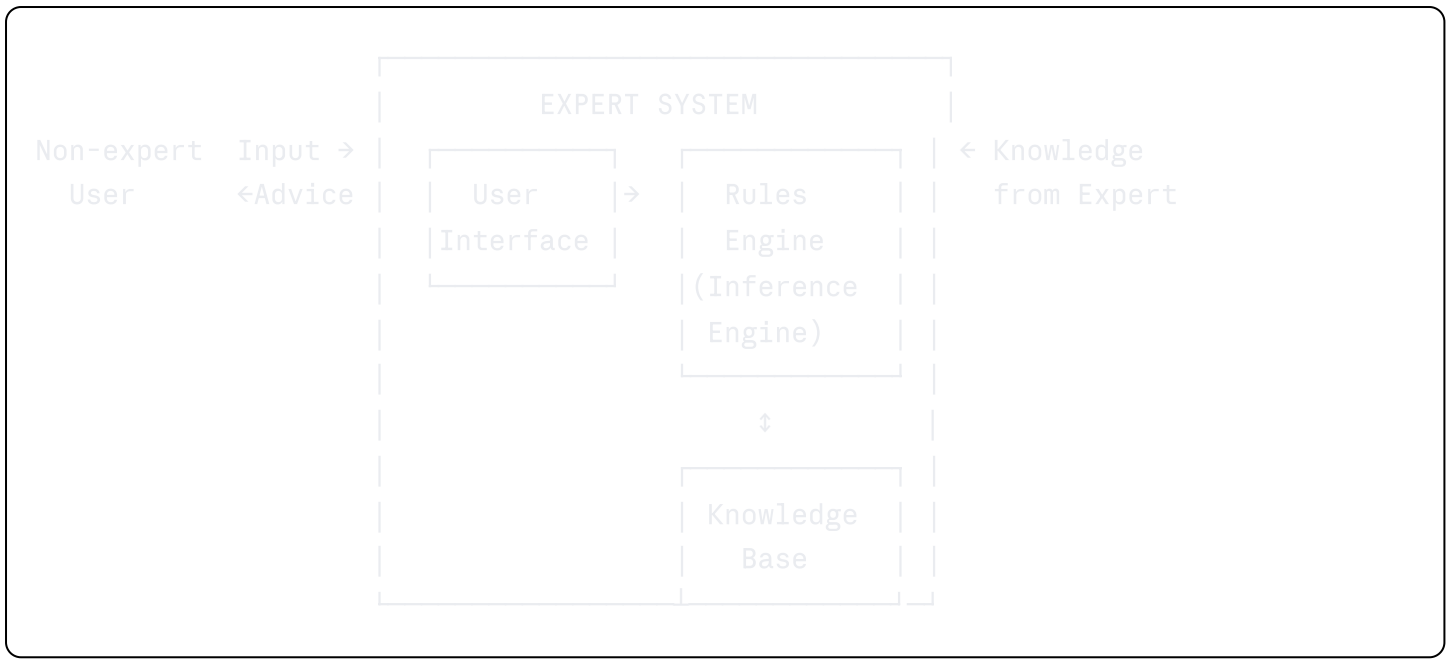
6. Expert Systems — Introduction

1. An **Expert System (ES)** is a computer program designed to solve complex problems and provide decision-making ability equivalent to that of a human expert in a specific domain.
2. It performs this by extracting knowledge from its **Knowledge Base (KB)** using reasoning and inference rules in response to user queries — the quality of the system directly depends on the richness of the KB.
3. Expert systems capture **scarce human expertise** and make it available at any time, at multiple locations simultaneously, without fatigue or emotional bias — this is their core value proposition over human experts.
4. Unlike conventional programs that follow fixed algorithms, expert systems use **heuristic knowledge** (rules of thumb) and handle uncertain, incomplete, or imprecise information naturally.
5. Common real-world examples include: Google search spelling correction, medical diagnosis systems, financial fraud detection systems, and tax advisory software.

---

7. Architecture of Expert Systems

The expert system architecture has the following structure:



1. The **User Interface** accepts queries from the non-expert user in a readable format, passes them to the inference engine, and presents the system's conclusions back to the user.
2. The **Inference Engine** (the "brain") applies inference rules to the knowledge base to derive conclusions — it uses either forward chaining (data-driven) or backward chaining (goal-driven) reasoning.
3. The **Knowledge Base** stores expert knowledge as IF-THEN rules, facts, and heuristics — it is the repository of all domain-specific knowledge that powers the system's reasoning.
4. An optional **Explanation Facility** traces the reasoning steps and explains to the user *why* a particular question is being asked and *how* a particular conclusion was reached.
5. A **Knowledge Acquisition Module** allows domain experts to add, modify, or delete rules in the knowledge base without deep programming knowledge, ensuring the system can be updated as domain knowledge evolves.

---

8. Components of Expert Systems in Detail

8a. User Interface

1. The user interface serves as the bridge between the non-expert user and the complex internal workings of the expert system.
2. It accepts queries as input in readable natural language or structured format and passes them to the inference engine for processing.
3. After receiving the response from the inference engine, it translates the conclusion back into understandable output and presents it to the user.
4. Good user interfaces include synonym dictionaries (so users can phrase queries naturally), automatic typo correction, and the ability to rephrase questions.
5. An interface that allows a non-expert to communicate with an expert system is what makes the technology practically useful — without it, only programmers could use the system.

1. The inference engine is the **main processing unit** — it is the "brain" of the expert system that controls how knowledge is accessed and reasoning is performed.
2. It applies inference rules to the contents of the knowledge base to derive new conclusions or make recommendations in response to user queries.
3. **Deterministic Inference Engine:** conclusions are assumed to be definitely true; based on exact facts and rules with no uncertainty.
4. **Probabilistic Inference Engine:** conclusions carry a degree of uncertainty (confidence factors or probabilities); suitable for domains where evidence is incomplete or noisy.
5. The inference engine uses two main modes: **Forward Chaining** (starts from known facts, applies rules to derive new facts until goal is reached) and **Backward Chaining** (starts from the goal, works backward to find supporting evidence in known facts).

8c. Knowledge Base

1. The knowledge base is a **structured database** that stores all the domain knowledge — facts, rules, heuristics, and relationships — acquired from human experts.
2. It is divided into **Factual Knowledge** (verified facts accepted by knowledge engineers, e.g., "antibiotics treat bacterial infections") and **Heuristic Knowledge** (rules of thumb based on experience, e.g., "if fever > 38°C and stiff neck, consider meningitis").
3. Knowledge is typically represented as **IF-THEN production rules**: IF (conditions are met) THEN (take this action / draw this conclusion).
4. The quality and completeness of the knowledge base directly determines the quality of the expert system — the more comprehensive and accurate the KB, the more precise the system's recommendations.
5. The KB can be represented using: simple relational tables, inheritance hierarchies (inheritable knowledge), formal logic (inferential knowledge), or procedural programs using If-Then rules in languages like LISP or Prolog.

---

9. Role of Expert Systems in AI

1. Expert systems represent the **Knowledge Representation and Reasoning** component of an AI system — they encode domain expertise and apply it logically to new problems.
2. An AI system exhibits intelligent behaviour through five components: **Perception** (acquiring data from environment), **Learning** (learning patterns from data), **Knowledge Representation and Reasoning** (storing and applying knowledge), **Planning** (deciding on actions), and **Execution** (carrying out the plan).
3. Expert systems primarily address the **Knowledge Representation and Reasoning** layer, which is the core of intelligent decision-making — once knowledge is represented and reasoning is correct, planning and execution naturally follow.

4. Expert systems demonstrate that AI can be **practical and domain-specific** without requiring general intelligence — by constraining the problem domain, expert systems can outperform average human practitioners in narrow tasks.
5. The knowledge representation approaches used in expert systems include: **Relational** (tables), **Inheritable** (class hierarchies), **Inferential** (formal logic), and **Procedural** (If-Then rules in code) — each suited to different types of domain knowledge.

---

10. Capabilities of Expert Systems

1. **Advising** — provides expert advice to non-expert users on queries within its domain, making specialised knowledge broadly accessible.
2. **Decision Making** — supports complex decision-making in domains such as medicine, finance, and engineering by evaluating evidence and recommending optimal actions.
3. **Diagnosing** — a medical ES can diagnose diseases using built-in medical rules, often matching or exceeding the accuracy of average human practitioners within the specific domain.
4. **Predicting Results** — can forecast likely outcomes based on current conditions (e.g., predicting credit risk, disease prognosis, or equipment failure).
5. **Consistent Recommendations** — unlike human experts, an ES is never affected by fatigue, emotions, or stress — it gives the same recommendation for the same input every time, ensuring repeatability and reliability.

Additional capabilities:

- Interpreting user input and extracting relevant parameters
- Demonstrating new products and explaining features
- Problem solving across complex multi-factor domains
- Operating in environments hazardous to humans
- Accelerating the learning curve for novices when used as training tools

---

11. Knowledge Acquisition

1. **Knowledge Acquisition** is the process of extracting, structuring, and encoding domain knowledge from human experts and other sources into the knowledge base — it is considered the primary **bottleneck** in building expert systems.
2. Knowledge sources include: textbooks, reports, databases (data mining), case studies, empirical data, and direct person expertise from domain specialists.
3. **Selection of Expert:** Choose an expert with acknowledged high performance, a successful track record, willingness to communicate their knowledge clearly, and availability to invest time in the development process.

4. **Interviewing Techniques** used to extract knowledge from domain experts:
    - **On-site observation** — K.E. watches D.E. solve real problems without interruption
    - **Problem discussion** — K.E. and D.E. explore the data and procedures needed
    - **Problem description** — D.E. describes prototypical problems for each answer category
    - **Problem analysis** — K.E. presents realistic problems; D.E. solves with visible reasoning steps
    - **System refinement** — D.E. gives K.E. problems to solve using acquired rules; errors surface
    - **System examination** — D.E. critiques the prototype system's rules and control structure
5. **Challenges in Knowledge Acquisition:** for some problems no expert exists; some experts cannot articulate what they do; experts may withhold trade secrets; some experts may have poor or biased expertise — making knowledge acquisition inherently difficult.

---

12. Characteristics of Expert Systems

1. **High-level Performance** — capable of responding with competency comparable to a human expert; the quality of advice must be of high integrity.
2. **Domain Specificity** — expert systems are typically highly specialised; the developer must limit scope to exactly what is needed to solve the target problem.
3. **Reliability and Understandability** — the system must be as reliable as a human expert and be able to explain its reasoning steps (explanation capability).
4. **Adequate Response Time** — must respond within time comparable to or better than a human expert; uses symbolic representations (rules, networks, frames) for efficient inference.
5. **Justified Reasoning and Metaknowledge** — allows users to ask the system to justify its advice; expert systems often reason with metaknowledge (knowledge about their own capabilities and limitations).

---

13. Limitations of Expert Systems

1. **Knowledge Dependency** — if the knowledge base contains wrong or outdated information, the system's responses will be incorrect — the system is only as good as its KB.
2. **No Creativity** — unlike human experts, an ES cannot generate creative or novel solutions; it can only work within the bounds of its encoded rules.
3. **High Maintenance Cost** — development and ongoing maintenance costs are very high; updating the KB requires significant effort and expert involvement.

4. **Domain Restriction** — a separate ES must be built for each domain; a medical ES cannot advise on financial matters — there is no generalisation across domains.
5. **No Self-Learning** — traditional expert systems cannot learn from new cases automatically; the KB must be manually updated when domain knowledge evolves.

---

14. Expert System Shells

1. An **Expert System Shell** is a pre-built software environment (framework) that contains the inference engine, user interface, and knowledge acquisition tools — but an **empty knowledge base** — allowing developers to build domain-specific ES by simply adding domain rules.
2. Shells separate the **inference mechanism** from the **domain knowledge** — the same shell can be used for medical diagnosis, financial advisory, or engineering troubleshooting by loading different knowledge bases.
3. **EMYCIN** (Empty MYCIN) was one of the first widely known ES shells — it was extracted from MYCIN by removing its medical knowledge base, leaving the backward-chaining inference engine and explanation system intact for reuse.
4. Key advantages of using shells: reduced development time, no need to implement inference from scratch, built-in explanation facilities, standardised knowledge representation format, and easier maintenance.
5. Examples of well-known ES shells: **CLIPS** (C Language Integrated Production System), **JESS** (Java Expert System Shell), **OPS5**, **ART** (Automated Reasoning Tool), **Prolog-based shells** — all provide the infrastructure; the developer supplies only domain knowledge.

---

15. Applications of Expert Systems

1. **Medical Diagnosis** — the first and most prominent application area; ES like MYCIN diagnose bacterial infections, CADUCEUS handles internal medicine, ONCOCIN supports cancer therapy.
2. **Finance and Banking** — detect fraud and suspicious transactions, evaluate loan applications, advise on investment strategies, perform credit risk assessment.
3. **Designing and Manufacturing** — assist in designing complex physical devices such as camera lenses, automobile components, VLSI chips, and structural engineering designs.
4. **Knowledge/Advisory Domain** — tax advisory systems, legal advisory systems, academic course selection advisors; make specialised expertise widely available to non-experts.
5. **Planning and Scheduling** — automate scheduling of manufacturing tasks, logistics, project planning, and resource allocation to achieve optimal outcomes.

Additional applications: fault diagnosis in industrial equipment, geological survey analysis, military decision support, environmental monitoring, and customer service automation.

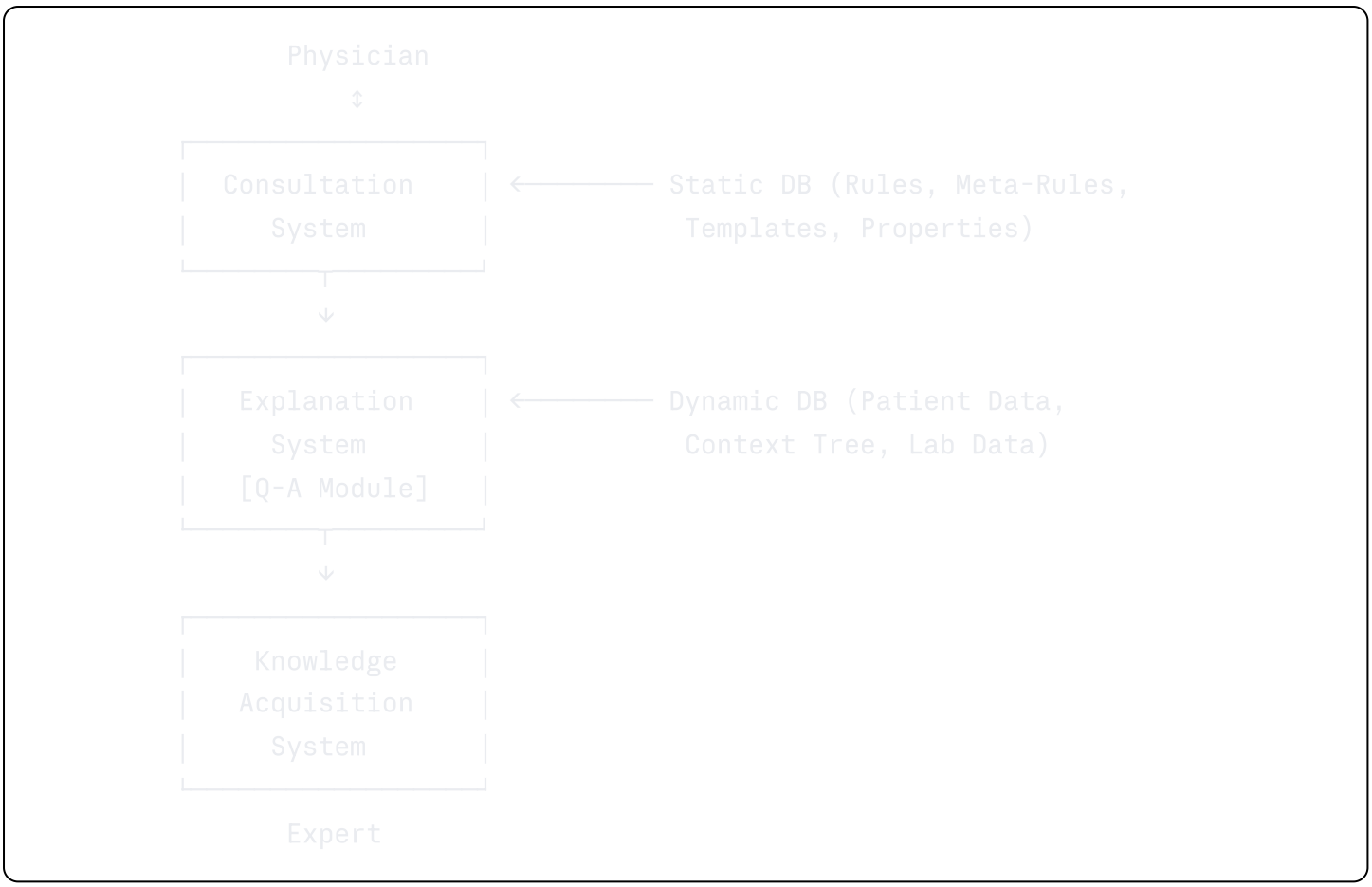
16. Case Study — MYCIN Expert System

16a. Overview and History

1. **MYCIN** was developed at Stanford University (1972–1980) as a PhD thesis project by Edward Shortliffe, with contributions from Davis, Buchanan, and van Melle under the Stanford Heuristic Programming Project.
2. MYCIN's domain was the **diagnosis of bacterial blood infections** (later expanded to meningitis and other ailments) and the **selection of appropriate antibiotic therapy** — advising non-expert physicians facing time constraints and incomplete evidence.
3. MYCIN contained between **50 and 500 production rules** (over 450 at its peak), all acquired from infectious disease experts through structured interviews — each rule represented an independent "nugget" of medical knowledge.
4. Despite proving its clinical competence (performing better than average physicians in controlled evaluations), MYCIN was **never deployed in routine clinical use** — primarily due to concerns about medical liability and the difficulty of integrating it into clinical workflows.
5. MYCIN's lasting contribution was not the system itself but its **methodology** — the use of production rules for knowledge representation, certainty factors for uncertain reasoning, and the architectural separation of knowledge from inference.

16b. MYCIN Architecture

MYCIN consists of three major sub-systems interacting through two databases:



Sub-systems:

1. **Consultation System** — performs diagnosis and therapy selection; uses goal-directed backward-chaining depth-first search; queries the physician when data is missing; ignores rules whose inputs cannot be obtained.
2. **Explanation System** — provides reasoning transparency via two modules: the **Q-A Module** (translates symbolic rules into readable English using translation patterns) and the **Reasoning Status Checker** (allows WHY and HOW queries to navigate up/down the reasoning tree).
3. **Knowledge Acquisition System** — allows domain experts (not programmers) to extend the static database via dialogue; uses IF-THEN formalism which was found to be easy for medical experts to learn; integrates new rules while detecting contradictions.

Databases:

- **Static Database** — stores production rules, meta-rules, rule templates, context type properties, and drug/organism tables; fed by the Knowledge Acquisition System.
- **Dynamic Database** — stores patient-specific data, laboratory results, and the Context Tree (hierarchical structure of patient → cultures → organisms → therapies → drugs); built during consultation, used by the explanation system.

16c. Knowledge Representation — Production Rules

1. MYCIN uses **IF-THEN production rules** — each rule has a premise (conditions) and an action (conclusion), written in the form `<predicate function> <object> <attribute>`

`<value>`.

2. Each rule is **completely modular** — all relevant context is explicitly stated within the rule; no rule implicitly depends on another, making rules easy to add, delete, or modify independently.
3. **Example MYCIN Rule:**

```
IF   the stain of the organism is gram-negative
AND  the morphology is rod
AND  the aerobicity is aerobic
THEN there is strongly suggestive evidence (CF = 0.8)
     that the organism is enterobacteriaceae
```

4. Rules are based on **templates** — standardised structures that allow the Knowledge Acquisition System to "understand" rule syntax and validate new rules automatically.
5. **Meta-Rules** are higher-order rules that control which rules to try first for a given sub-goal — they act as strategy rules that prune the search tree, improving efficiency and sometimes providing cleaner explanations.

16d. Certainty Factors (Inexact Reasoning)

1. MYCIN uses **Certainty Factors (CF)** — numbers between −1 and +1 — to represent the degree of belief in a hypothesis; CF is **not** probability but an intuitive measure of evidential support.
2. The **CF of a rule premise** equals the **minimum CF** of all its conditions (weakest-link principle — the strength of the argument is limited by its weakest component).
3. The **CF of a rule conclusion** = (CF of premise) × (CF of rule), capturing how much the premise's confidence is attenuated by the rule's own reliability.
4. When **multiple rules** support the same conclusion with CFs c_1, c_2, ..., c_n, they are combined using: $\mathbf{c_1 \oplus c_2 = 1 - (1 - c_1)(1 - c_2)}$ — this ensures combined confidence is always higher than either individual value but never exceeds 1.
5. A **positive CF sum** is confirming evidence; a **negative CF sum** is disconfirming evidence — this allows MYCIN to weigh conflicting evidence and still reach reasoned conclusions.

Worked Example with Certainty Factors:

```
F1: Ungee gives milk          CF = 0.9
F2: Ungee eats meat           CF = 0.8
F3: Ungee has hoofs           CF = 0.7
R1: gives milk → mammal       CF = 0.6
R2: mammal AND eats meat → carnivore   CF = 0.5
R3: has hoofs → carnivore     CF = 0.4

Step 1: R1 + F1 → mammal (F4)
        CF(F4) = 0.9 × 0.6 = 0.54

Step 2: R2 + F4 + F2 → carnivore (F5) via R2
        premise CF = min(0.54, 0.8) = 0.54
        CF(F5 via R2) = 0.54 × 0.5 = 0.27

Step 3: R3 + F3 → carnivore (F5) via R3
        CF(F5 via R3) = 0.7 × 0.4 = 0.28

Step 4: Combine F5 from R2 and R3
        CF(F5) = 1 − (1 − 0.27)(1 − 0.28) = 1 − (0.73)(0.72) = 1 − 0.526 = 0.474
```

16e. Consultation Control Structure

1. MYCIN's consultation is controlled by a **goal-directed backward-chaining depth-first search** — it starts from high-level diagnostic goals and recursively establishes sub-goals.
2. The four-step high-level algorithm: (1) determine if the patient has a significant infection; (2) determine the likely identity of significant organisms; (3) identify potentially useful drugs; (4) select the best drug or combination.
3. **Sub-goals** are established whenever a parameter's value is needed — MYCIN prefers to obtain values directly from the user (e.g., lab data) rather than deducing them, saving computation.
4. The **Preview Mechanism** reads rules before invoking them to avoid unnecessary deductive work if the sub-goal has already been tested, and prevents infinite recursive loops from self-referencing sub-goals.
5. **Unity Paths** allow MYCIN to bypass sub-goals by following a path whose certainty is known (CF = 1), enabling definitive conclusions without further reasoning when the evidence is certain.

16f. Therapy Selection

1. After identifying likely organisms, MYCIN proceeds to **therapy selection** using a Plan-Generate-and-Test approach — it generates a list of candidate drug therapies and tests them against multiple criteria.
2. Only organisms whose hypotheses have been deemed **significantly likely** (sufficient CF) are considered; each is assigned an Item Number and given probability scores for identity

and drug sensitivity.

3. Final drug selection is based on maximising **coverage** (treating all likely organisms) while minimising the **number of drugs** (reducing side effects and costs), and screening for **contraindications** specific to the patient.
4. Experts can request **alternate treatments** — therapy selection repeats with previously recommended drugs removed, exploring the next best option.
5. MYCIN's therapy recommendations were shown in clinical evaluations to be **competent** — panels of infectious disease experts rated its recommendations as acceptable or better than those of average physicians, even in cases where the experts themselves disagreed.

16g. Explanation System in Detail

1. The explanation system provides transparency into MYCIN's reasoning — it answers *WHY* (why is this question being asked?) and *HOW* (how was this conclusion reached?).
2. The **Q-A Module** uses translation patterns associated with each predicate function to convert symbolic production rules into readable English sentences automatically.
3. The **Reasoning Status Checker** implements tree traversal over the traced rules: **WHY** moves up the reasoning tree (to the parent goal), and **HOW** moves down (to sub-goals and the rules being applied).
4. The explanation facility was critical for clinical acceptance — physicians needed to understand *why* MYCIN was asking questions and *what evidence* supported its recommendations before trusting its advice.
5. Meta-rules, while improving search efficiency, sometimes created "murky" explanations because they encode search strategy rather than domain knowledge — a trade-off between efficiency and transparency.

WHY Example:

```
MYCIN: Was penicillinase added to CULTURE-1?
Physician: WHY

[Because establishing this will aid in determining whether
ORGANISM-1 is a contaminant.
It has already been established that:
  - the site of CULTURE-1 is blood
  - the gram stain of ORGANISM-1 is grampos
Therefore, if penicillinase was added, there is weakly
suggestive evidence (CF=0.3) that ORGANISM-1 is a contaminant.]
```

16h. Key Results and Legacy of MYCIN

1. **Clinical Performance:** MYCIN performed better than average physicians and comparably to infectious disease experts in controlled evaluations — it achieved correctness ratings of 65% vs 42.5% for non-expert physicians.

2. **Never deployed in practice** — despite clinical competence, medico-legal concerns, workflow integration issues, and computational constraints of the era prevented routine use.
3. **EMYCIN (Empty MYCIN):** The knowledge-free shell extracted from MYCIN became one of the first ES shells — demonstrating the value of separating inference mechanism from domain knowledge.
4. **Key Contributions to AI:**
    - Established production rules as the standard for knowledge representation in ES
    - Introduced certainty factors as a practical approach to uncertain reasoning
    - Pioneered the explanation facility as a requirement for trustworthy AI systems
    - Demonstrated meta-level knowledge (rules about rules) as a powerful search-control mechanism
5. **MYCIN's methodology** — not the system itself — became the template for expert systems development for the next two decades.

Quick Revision — Key Concepts Table

Concept	Key Idea	Exam Focus
Forward Planning	Start from facts, apply rules, reach goal	Algorithm trace, pros/cons
Backward Planning	Start from goal, work backwards to facts	Algorithm trace, comparison with forward
Goal Stack Planning	Stack-based recursive goal decomposition	Step-by-step example, Sussman Anomaly
Hierarchical Planning	Decompose abstract tasks into primitives	Levels of abstraction, advantages
Expert System	AI program mimicking human expert	Definition, components, 5-mark answer
Inference Engine	Brain of ES — forward/backward chaining	Types (deterministic/probabilistic), modes
Knowledge Base	DB of domain facts and IF-THEN rules	Factual vs heuristic, role
Knowledge Acquisition	Extracting expert knowledge — bottleneck	Techniques, challenges
Expert System Shell	ES without KB — reusable framework	EMYCIN, purpose
MYCIN	Medical ES for blood infection diagnosis	Architecture, CF, rules, WHY/HOW

Concept	Key Idea	Exam Focus
Certainty Factors	Non-probabilistic belief measure in MYCIN	Formula: $c_1 \oplus c_2 = 1-(1-c_1)(1-c_2)$
Meta-Rules	Rules that control which rules to try	Purpose, trade-off with explanation

Exam-Style Answer Templates

"Describe Expert System Architecture (5 marks)"

1. Define Expert System (1 line)
2. Draw/describe the architecture diagram showing: User → User Interface → Inference Engine ↔ Knowledge Base, plus Knowledge Acquisition and Explanation modules
3. Explain each component: User Interface, Inference Engine (forward/backward chaining), Knowledge Base (factual + heuristic)
4. Mention Explanation Facility and Knowledge Acquisition module
5. State that performance depends on KB quality

"Explain MYCIN Expert System (5–10 marks)"

1. Overview: Stanford 1972–1980, medical diagnosis ES for bacterial infections
2. Architecture: Consultation System, Explanation System, Knowledge Acquisition System + Static DB + Dynamic DB
3. Knowledge representation: IF-THEN production rules with Certainty Factors
4. Control: backward-chaining depth-first search
5. CF formula + worked example, WHY/HOW explanation
6. Results: competent but never deployed; legacy = EMYCIN shell and production-rule methodology

"Explain Forward and Backward Chaining with example (5–10 marks)"

1. Define forward chaining + algorithm
2. Trace example (Ungee animals)
3. Define backward chaining + algorithm
4. Trace same example backward
5. Comparison table (start point, use case, efficiency, explanation)

"What is Goal Stack Planning? (5 marks)"

1. Define with stack data structure concept
2. Explain push/pop mechanism for goals and sub-goals

3. Give blocks world example with step-by-step stack trace
4. Mention Sussman Anomaly (interference between goals)
5. State advantage: clear hierarchical decomposition

End of AIES Unit IV Notes