CS4710: Artificial Intelligence
Expert Systems

Some processes can only be reasonably done by experts. How do we model this knowledge and build such systems?

Topics

- What is an Expert System / Brief History
- Example: Rule-Based Expert Systems
- Concrete Example: CLIPS
- Software Engineering Process for Expert Systems

WHAT IS AN EXPERT SYSTEM?

- An expert system is a computer program that contains some of the subject-specific knowledge of one or more human experts.

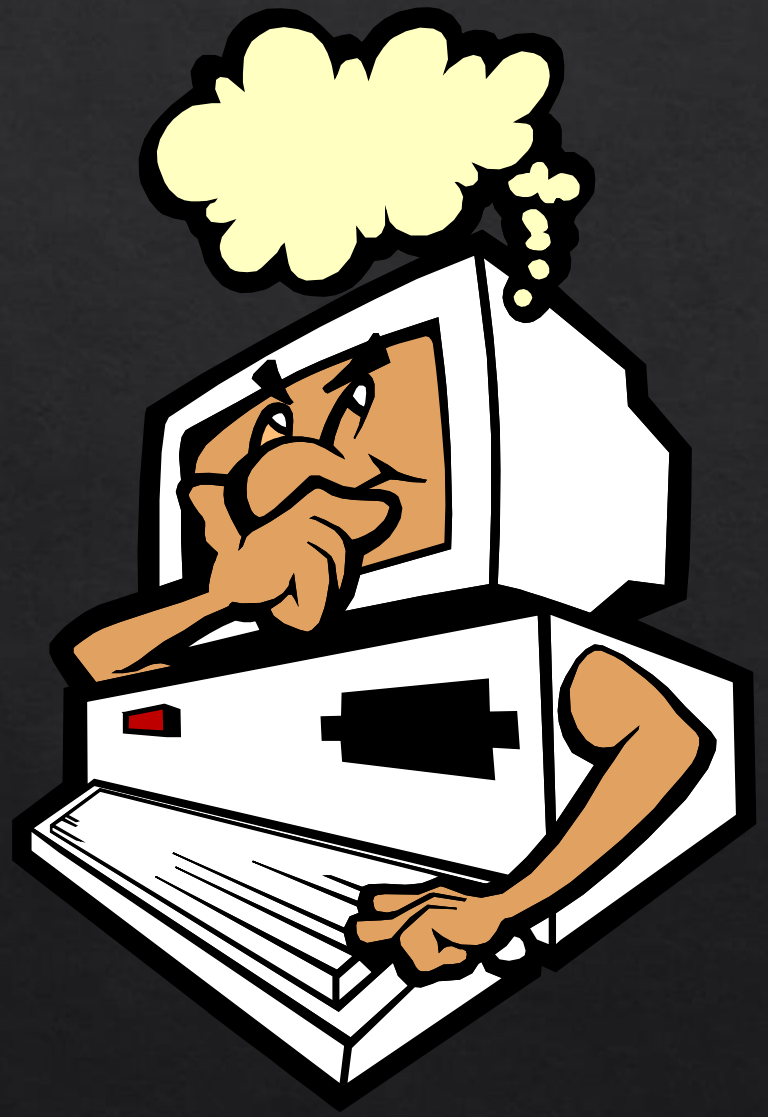
Expert System Problem Categories

- Interpretation: Forming high level conclusions from raw data
  - Example: People who go to UVa tend to get good jobs
- Prediction: Projecting probable consequences of situations
  - Example: “If patient has grave’s disease they will get hyperthyroidism”
- Diagnosis: Determining cause of malfunctions in complex situations based on symptoms
  - Example: “Given blood test results and other symptoms it is likely you have diabetes”
- Design: Finding a configuration of system components that meets a set of goals
  - Example: Smart home. Need certain appliances but want to keep energy usage low.

Expert System Problem Categories

- Planning: Devising a set of steps to reach a set of goals (generated on the fly)
  - Example: “Here are a set of steps that will fix your computer”
- Monitoring: Comparing observed behavior to expected behavior
  - Example: “Warning: oxygen levels coming out of engine are too low, throw check engine error”
- Instruction: Assisting in the education process in technical domains
  - Example: “In order to learn guitar, the optimal next exercise is to practice your scales”
- Control: Governing the behavior of a complex environment
  - Example: Establishing actions of a robotic arm that does something

Expert Systems: Brief History



- Early 70s
- Goal of AI scientists → develop computer programs that could in some sense think .
- In 60s→ general purpose programs were developed for solving the classes of problems but this strategy produced no breakthroughs.
- In 1970→ it was realized that The problem-solving power of program comes from the knowledge it possesses.

To make a program intelligent, provide it with lots of high-quality, specific knowledge about some problem area.

Aside: Discrete Math Review

Predicate Logic

- Powerful way to represent most facts
- Allows for inference
  - Modus Ponens, etc…
- Is relatively flexible, and common in AI systems.

Propositions

- A *proposition* is a variable that represents a statement and has a true/false value
  - P = “Allison Likes Cakes”
  - Q = “Allison Eats Cakes”
  - *P & Q are arbitrary, can use any symbol
  - P & Q can be either true or false

Basic Connectives

- $\wedge$
  - $P \wedge Q$
  - Allison likes cakes AND Allison eats cakes
- $\vee$
  - $P \vee Q$
  - Alisson likes or eats cakes
- $\neg$
  - $\neg P$
  - Allison does not like cakes

Conditionals

- →
  - P → Q
  - If Allison likes cakes then she eats them
- ⇔
  - P ⇔ Q
  - Allison likes cakes iff she eats them

Functional Propositions

- Common for propositions to be functions and have arbitrary parameters.

- cake(C) = “C is a cake”
- likes(T) = “Allison likes T”

- Gives us some more general expressing power

Quantifiers

- ∀
  - For all quantifier
  - ∀c (cake(c) → likes(c))
  - For all cakes, Allison likes them
- ∃
  - Exists quantifier
  - ∃c | cake(c)
  - There exists something that is a cake (Thank God!)

Modus Ponens

- P
- P → Q

- ∴ Q

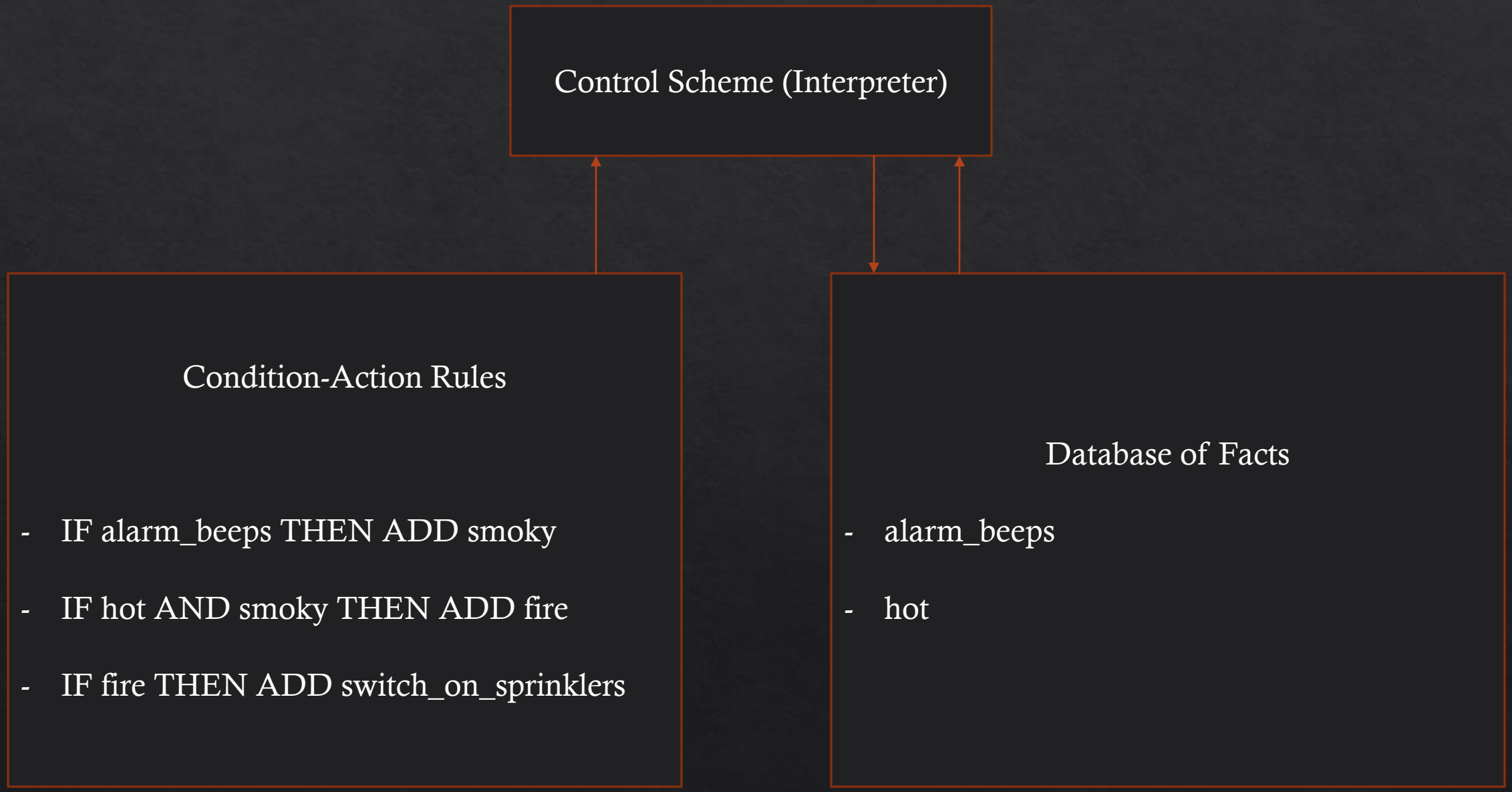
- Similarly:

- ¬Q
- P → Q

- ∴ ¬P

Example:
Rule-Based Expert Systems

Rule-Based Architecture



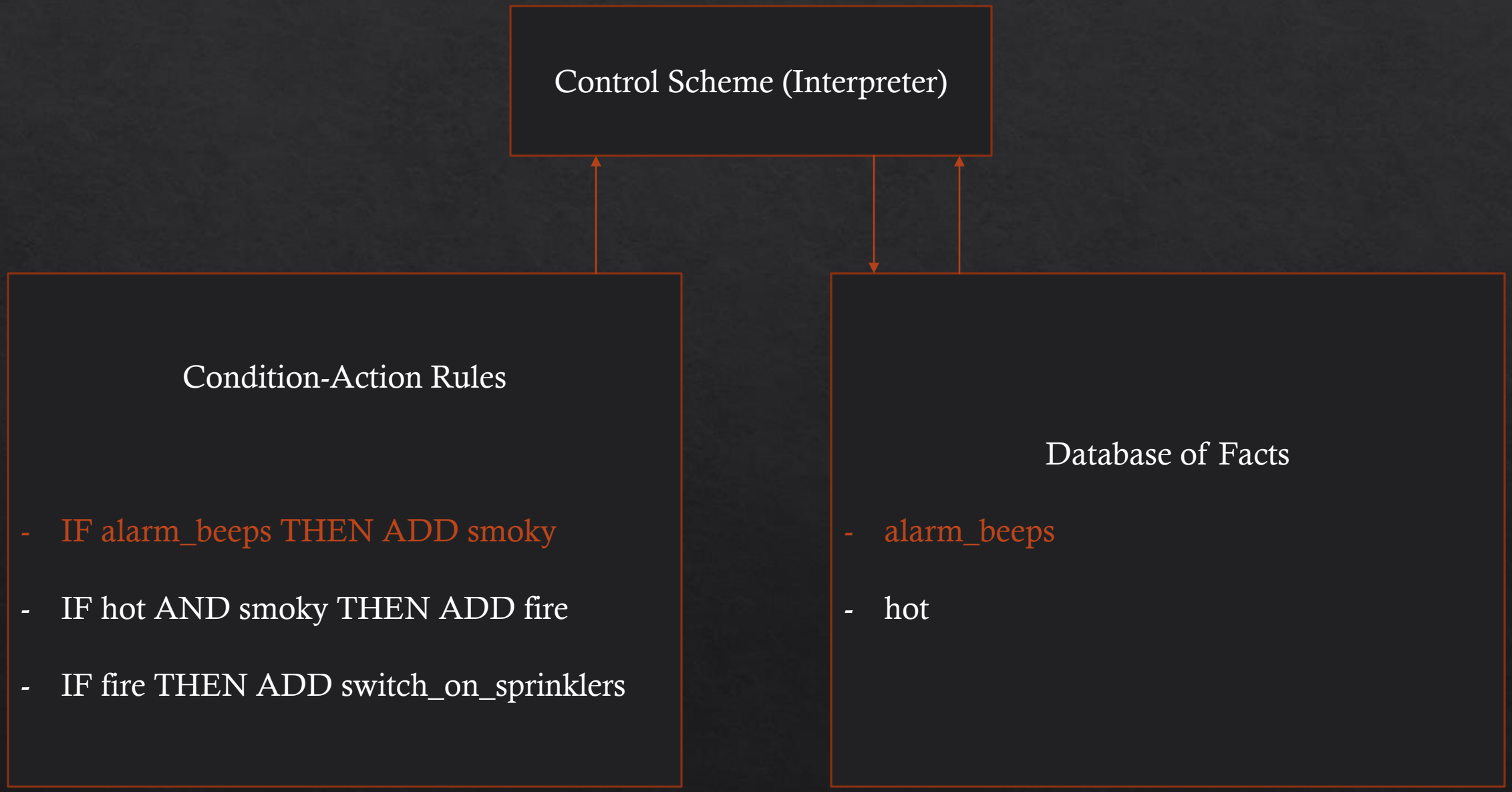
Acquiring New Facts

- By human operator simply telling the system something is true.
  - Example?
- By interacting with the environment directly
  - Temperature sensor determines whether *hot* is set to true or false
- Others?

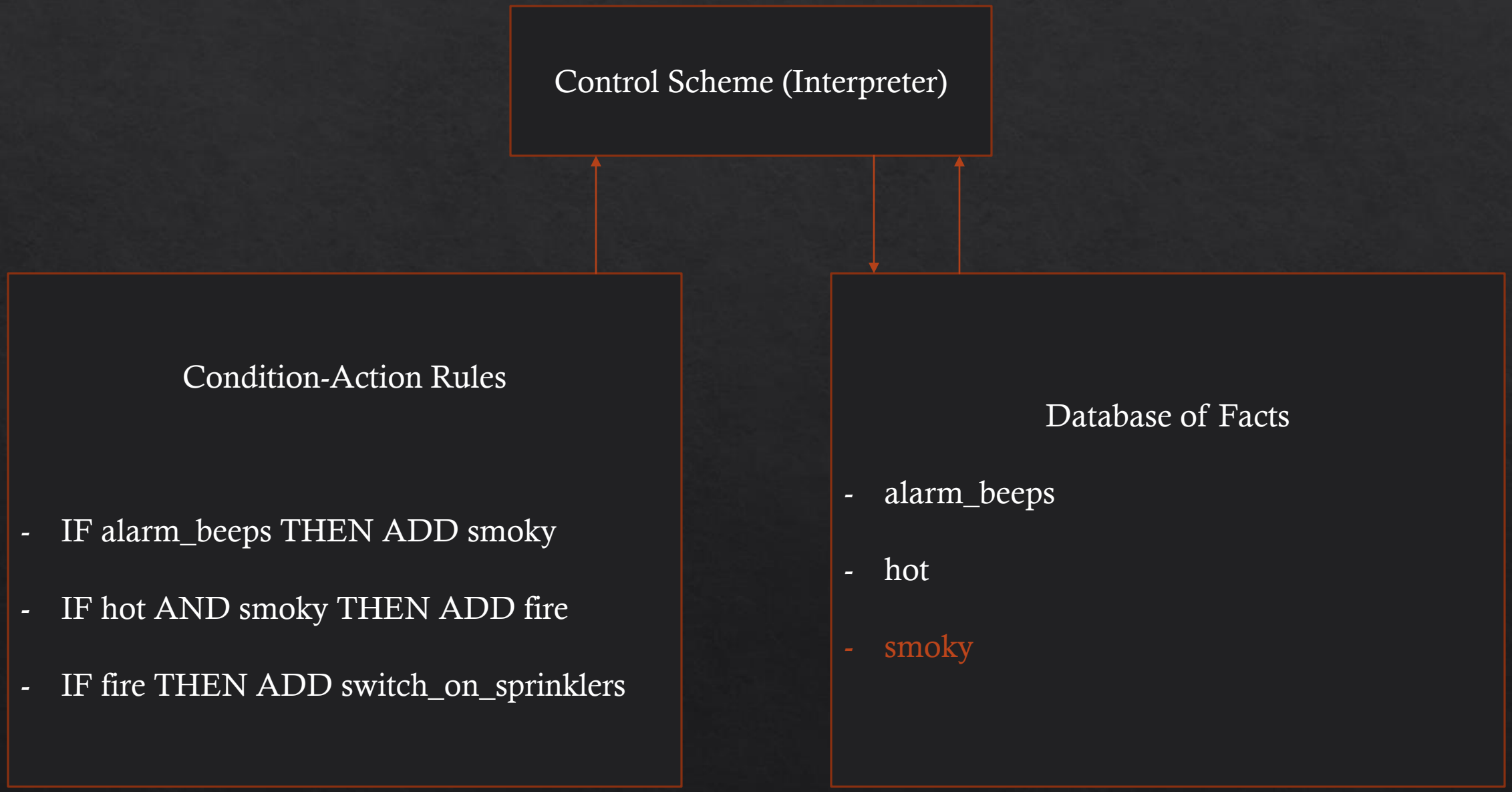
Forward Chaining

- Until special HALT symbol in DB
  - Pick a logical rule that can be applied
  - Add resulting truth statement to DB of facts
  - Continue until no new knowledge can be created
- This continues indefinitely, until a special halt signal is given, or every time a DB fact is updated or changed externally.

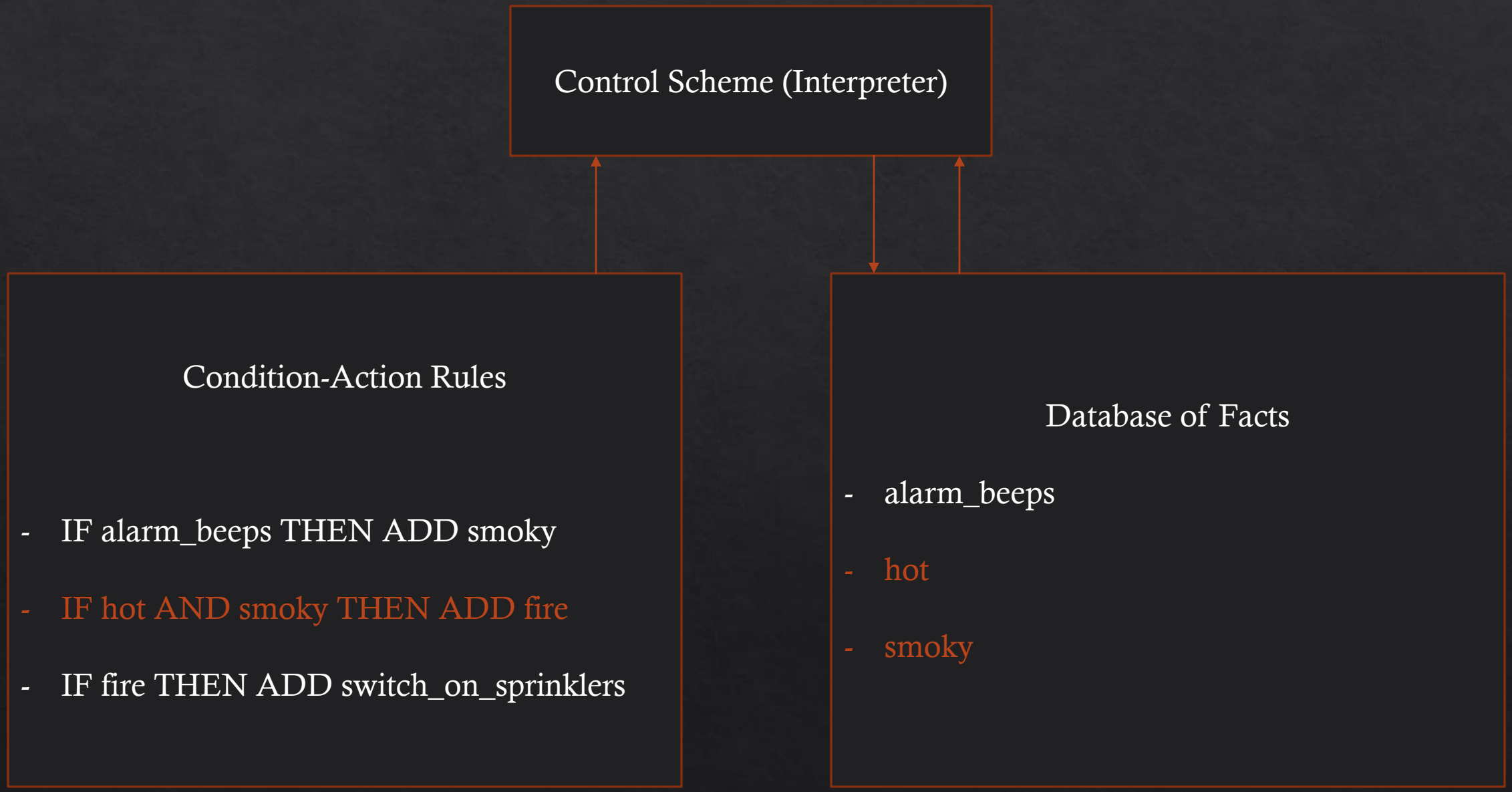
Forward Chaining: Simple Example



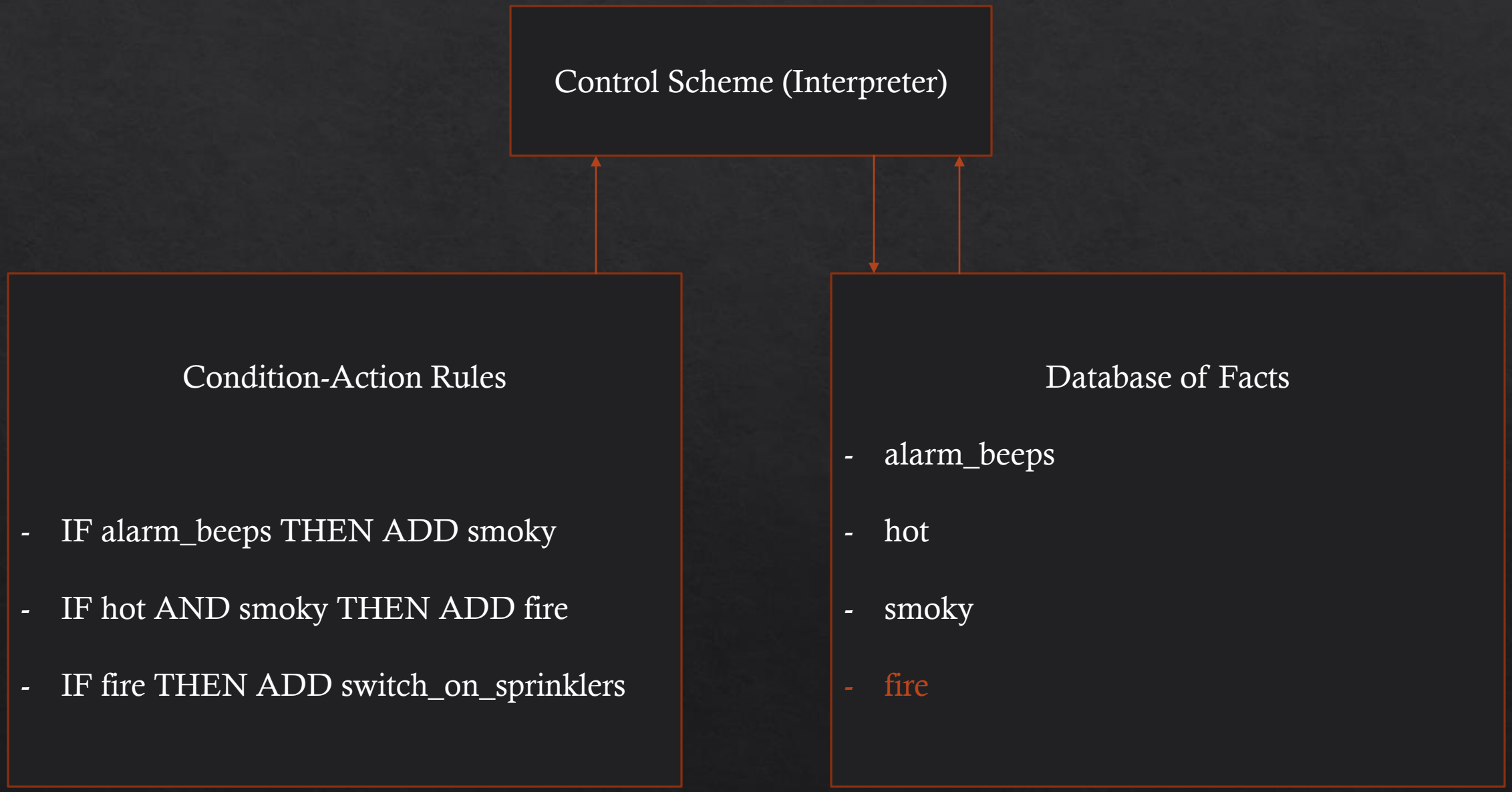
Forward Chaining: Simple Example



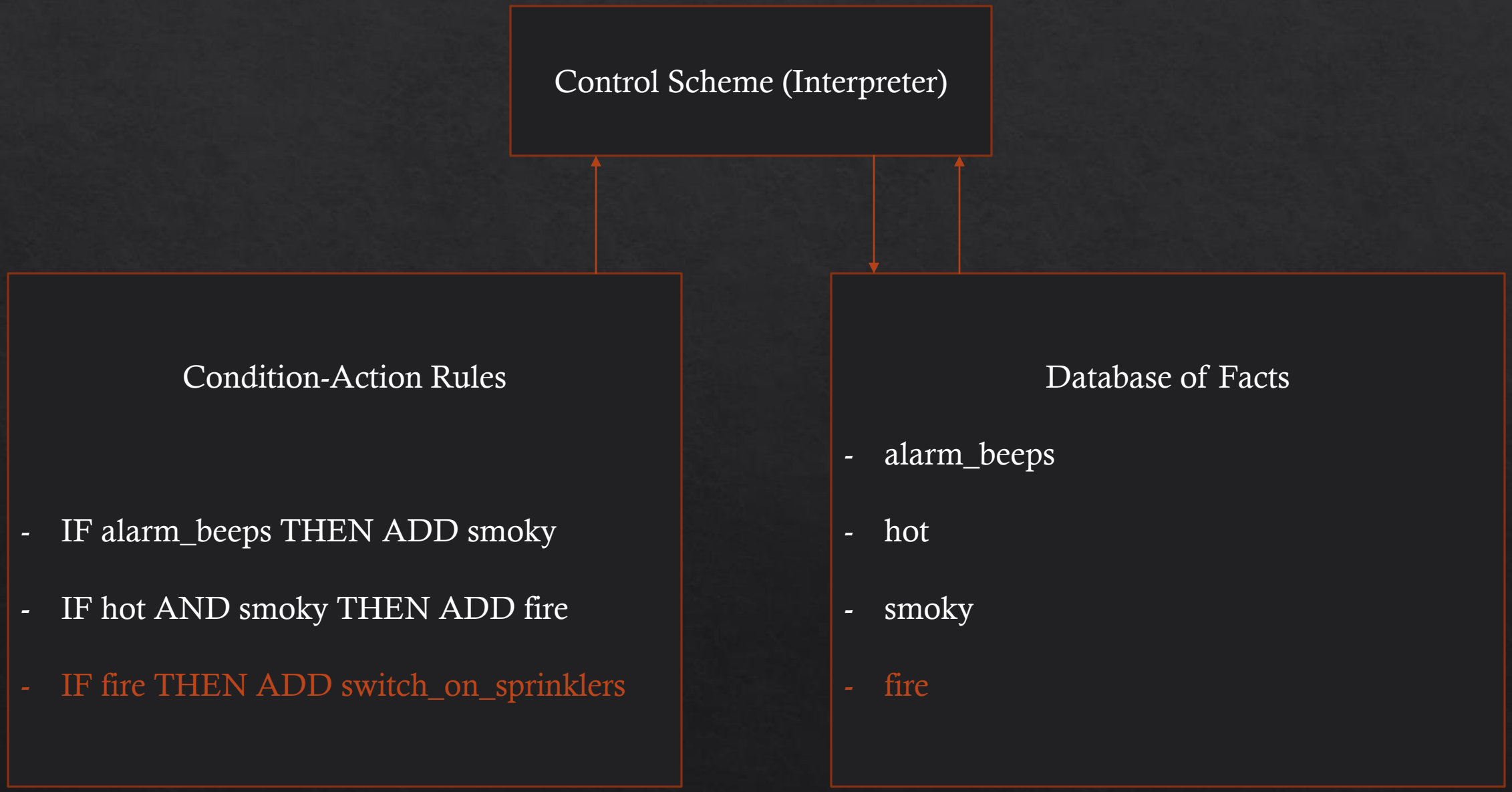
Forward Chaining: Simple Example



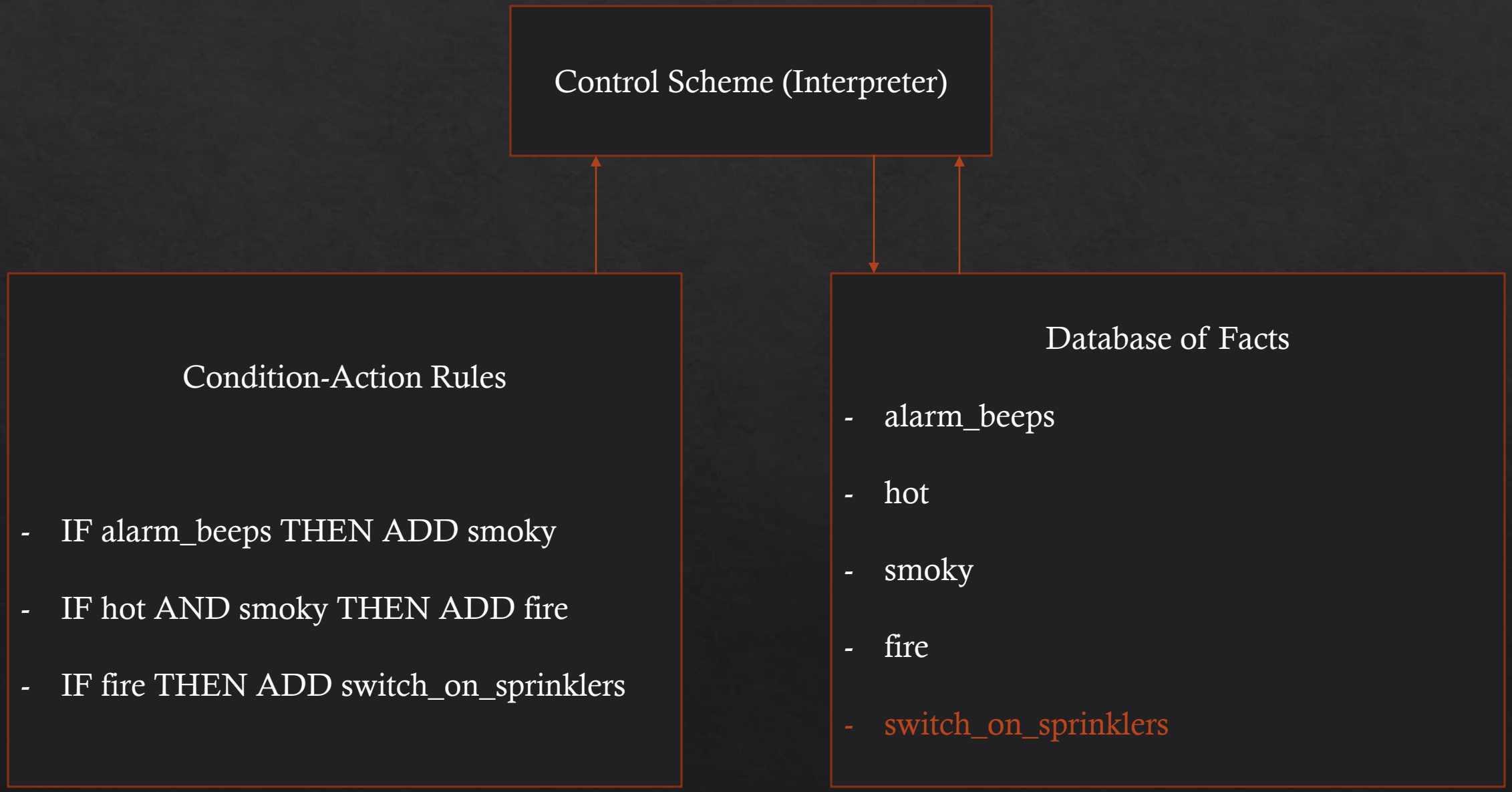
Forward Chaining: Simple Example



Forward Chaining: Simple Example



Forward Chaining: Simple Example



Conflict Resolution

- Which rule to apply when many choices available?
- What are some ways we could deal with this?
  - Discuss!
- Can you devise a situation when the order of rule application might be important?

Reason Maintenance

- What if a fact is removed
  - E.g., if ‘hot’ is removed from the DB of facts.
  - This means some other acquired facts may now be invalid?
- How would you deal with this?

Reason Maintenance

Control Scheme (Interpreter)

Condition-Action Rules

- IF alarm_beeps THEN ADD smoky
- IF hot AND smoky THEN ADD fire
- IF fire THEN ADD switch_on_sprinklers

Database of Facts

- alarm_beeps
- ~~hot~~
- smoky
- fire
- switch_on_sprinklers

Pattern Matching

- Can extend this idea to arbitrary parameters.
- For example:
- IF temperature(R,hot) AND environment(R,smoky)
  - THEN ADD fire_in(R)

Backward Chaining

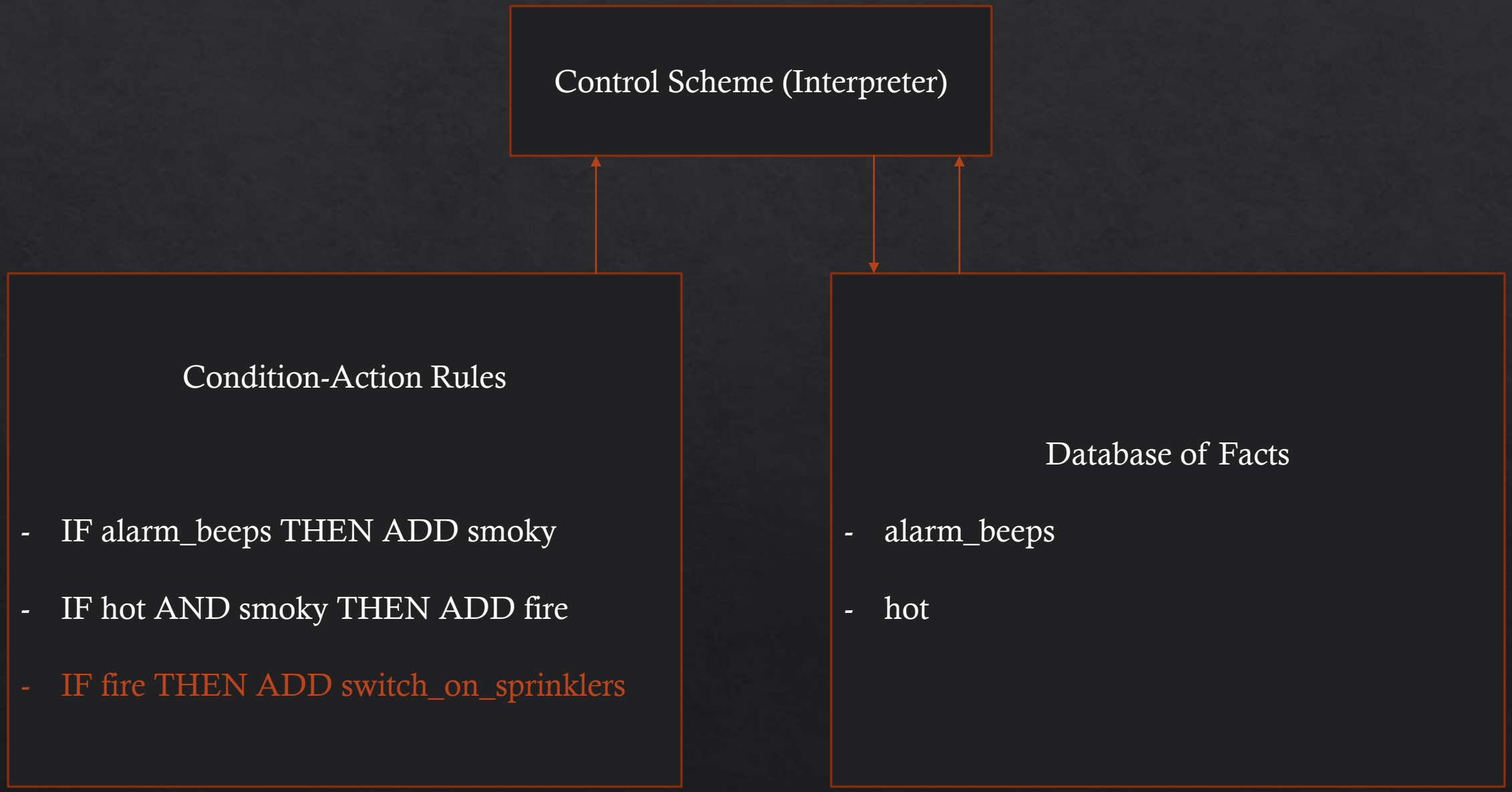
- Good when we want to query the system and see if something is true.
- Suppose I ask the system "is it the case that the room is hot and sprinklers are on"

- Backward Chaining
  - See if proposition(s) already true
  - If not, find rules that can prove them and try to prove those conditionals are true
    - e.g., I don't know if sprinklers are on but I do know that if hot and smoky → sprinklers are on.
    - So…prove hot and smoky instead

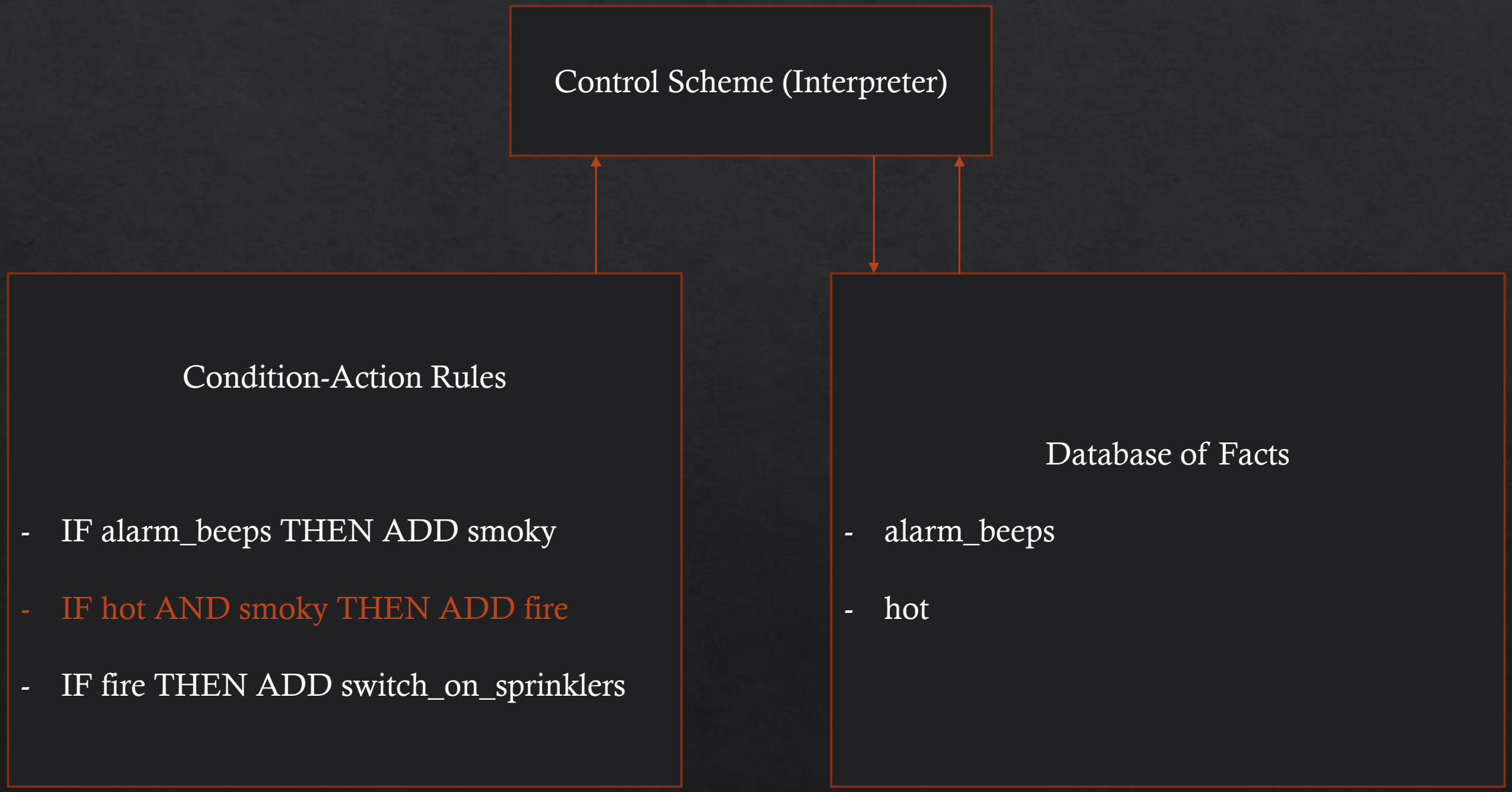
Backward Chaining

- QueryForFact(f)
- Base Case:
  - If f in working memory, return true
- Else:
  - Find rules with f as consequence
  - Recursively call QueryForFact on rule
  - Return true if QueryForFact returns true
- Return false if no rule found and not in working memory

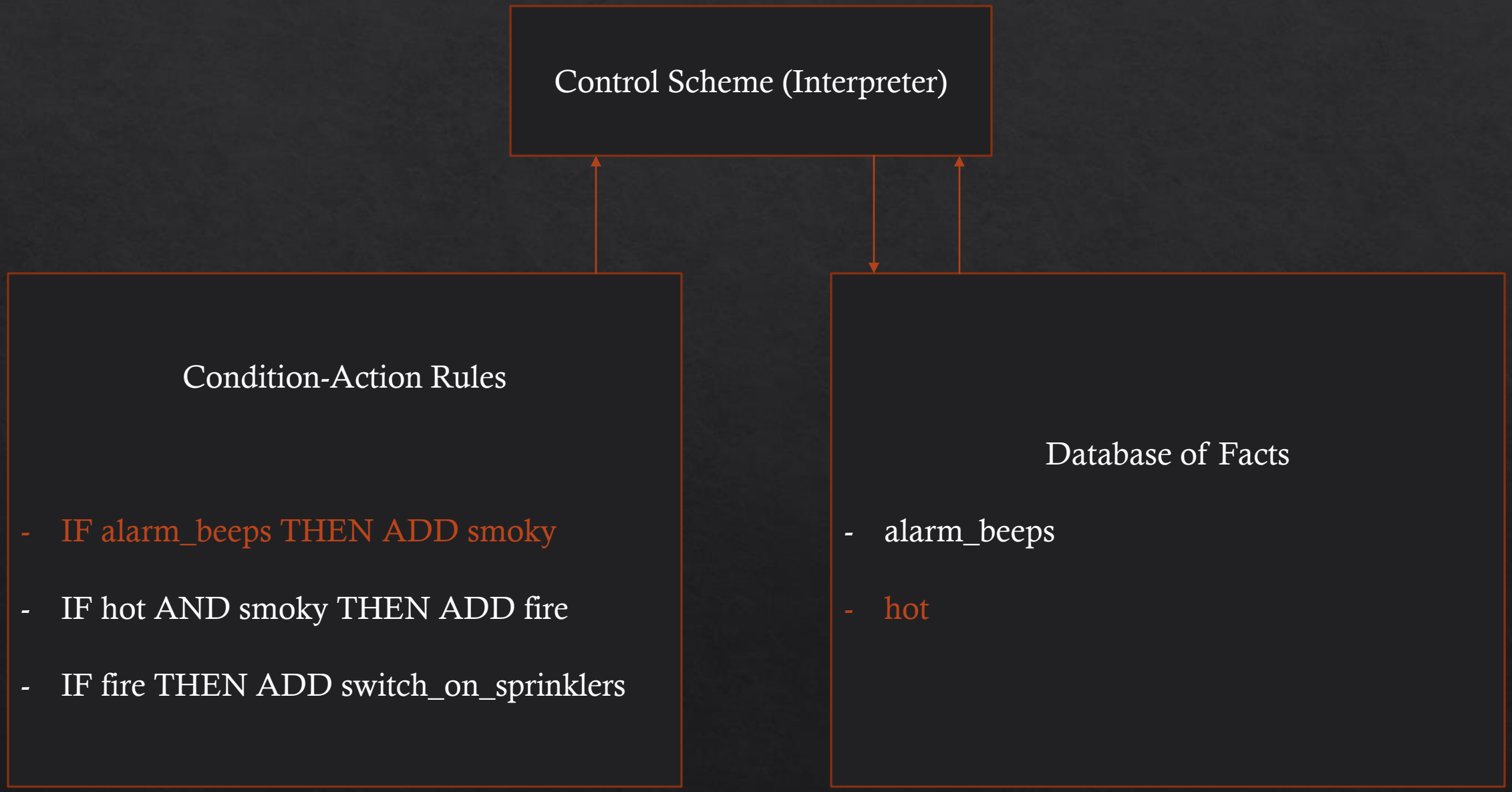
Query(switch_on_sprinklers)



Return Query(fire)



Return Query(hot) && Query(smoky)



Backward Chaining

- QueryForFact(f)
- Base Case:
  - If f in working memory, return true
- Else:
  - Find rules with f as consequence
  - Recursively call QueryForFact on rule
  - Return true if QueryForFact returns true
- Return false if no rule found and not in working memory

CLIPS

CLIPS: Expert System Shell

- Rule-based expert systems are often known as *production systems* (CLIPS actually stands for C Language Integrated Production System).

The CLIPS Programming Tool

- History of CLIPS
  - Influenced by OPS5 and ART
  - Implemented in C for efficiency and portability
  - Developed by NASA, distributed & supported by COSMIC
  - Runs on PC, Mac, also under UNIX and VAX VMS
- CLIPS provides mechanisms for expert systems
  - A top-level interpreter
  - Production rule interpreter
  - Object oriented programming language
  - LISP-like procedural language

Components of CLIPS

- Rule-Based Language
  - Can create a fact list
  - Can create a rule set
  - An inference engine matches facts against rules
- Object-Oriented Language
  - Can define classes
  - Can create different sets of instances
  - Special forms allow you to interface rules and objects

Defining Facts

- Facts can be asserted

  CLIPS> (assert (today is sunday))

  <Fact-0>

- Facts can be listed

  CLIPS> (facts)

  f-0 (today is sunday)

- Facts can be retracted

  CLIPS> (retract 0)

  CLIPS> (facts)

Managing Facts

- Clearing all facts

  CLIPS> (clear)

  CLIPS> (facts)

- Grouping facts - typically in a file (“today.clp”)

  (deffacts today ; can be cleared with (undeffacts today)

  (today is sunday)

  (weather is warm)

)

- After loading facts, assert with (reset)

Defining Rules

- Rules have the following structure

```
(defrule rule-name optional-comment
    optional-declaration
    condition
    …
    condition
    =>
    action
    …
    action
)
```

An Example CLIPS Rule

```
(defrule sunday “Things to do on Sunday”
    (salience 0)      ; salience in the interval [-10000, 10000] //defines priority
    (today is Sunday)
    (weather is sunny)
    =>
    (assert (chore wash car))
    (assert (chore chop wood))
)
```

- So, if fact list contains conditions, add assertions

Variables & Pattern Matching

- Variables make rules more applicable

```
(defrule pick-a-chore
    (today is ?day)
    (chore is ?job)
    =>
    (assert (do ?job on ?day))
)
```

- If conditions are matched, then bindings are used

Defining Functions in CLIPS

- Uses a LISP or Scheme-like syntax

```
(deffunction function-name (arg … arg)
    action … action)
```

```
(deffunction initialize ()
    (clear)
    (assert (today is sunday)))
```

Explanation Subsystem Example

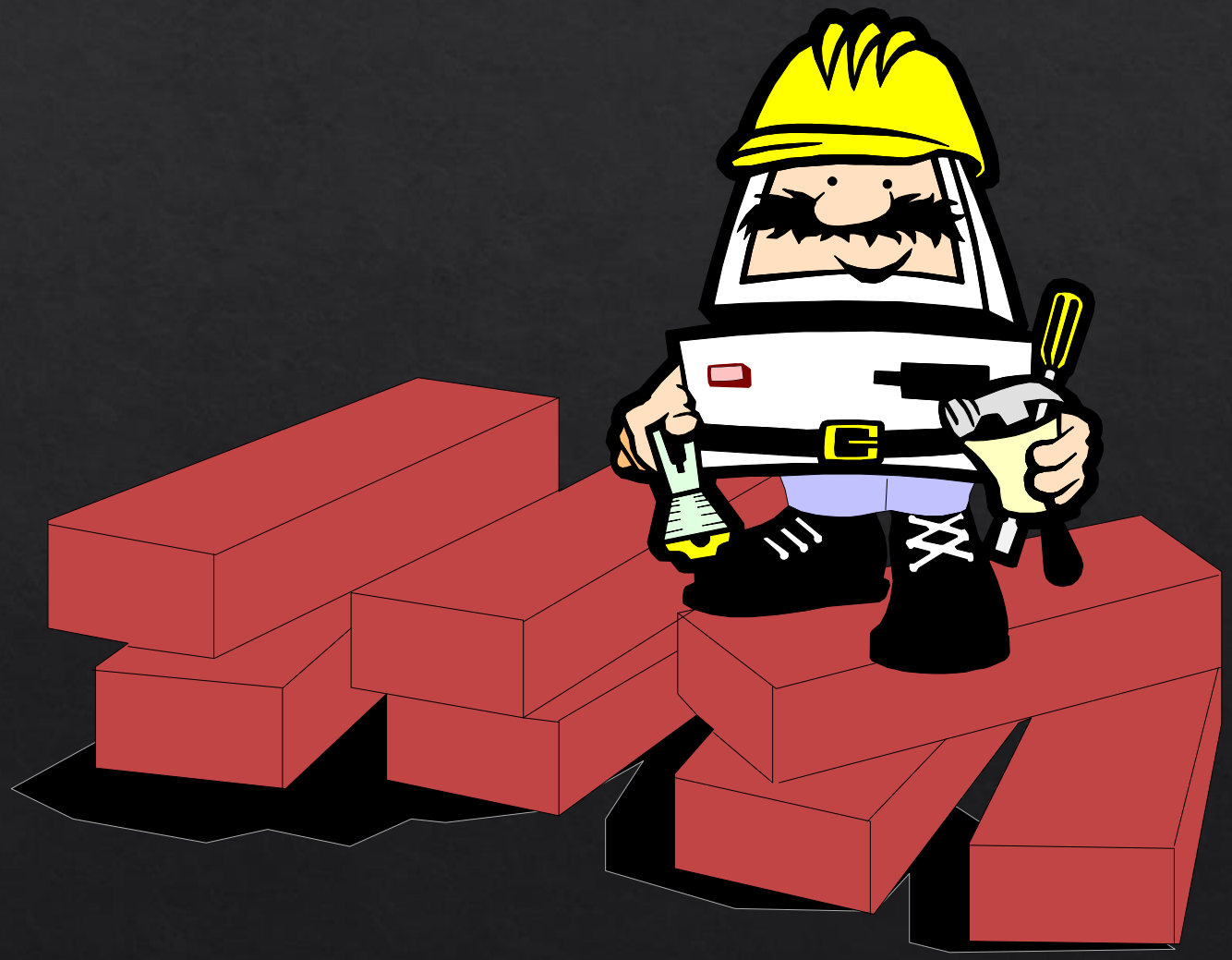
- This part of system allows us to provide an explanation of the reasoning that led to some conclusion.

- Let’s see an example:

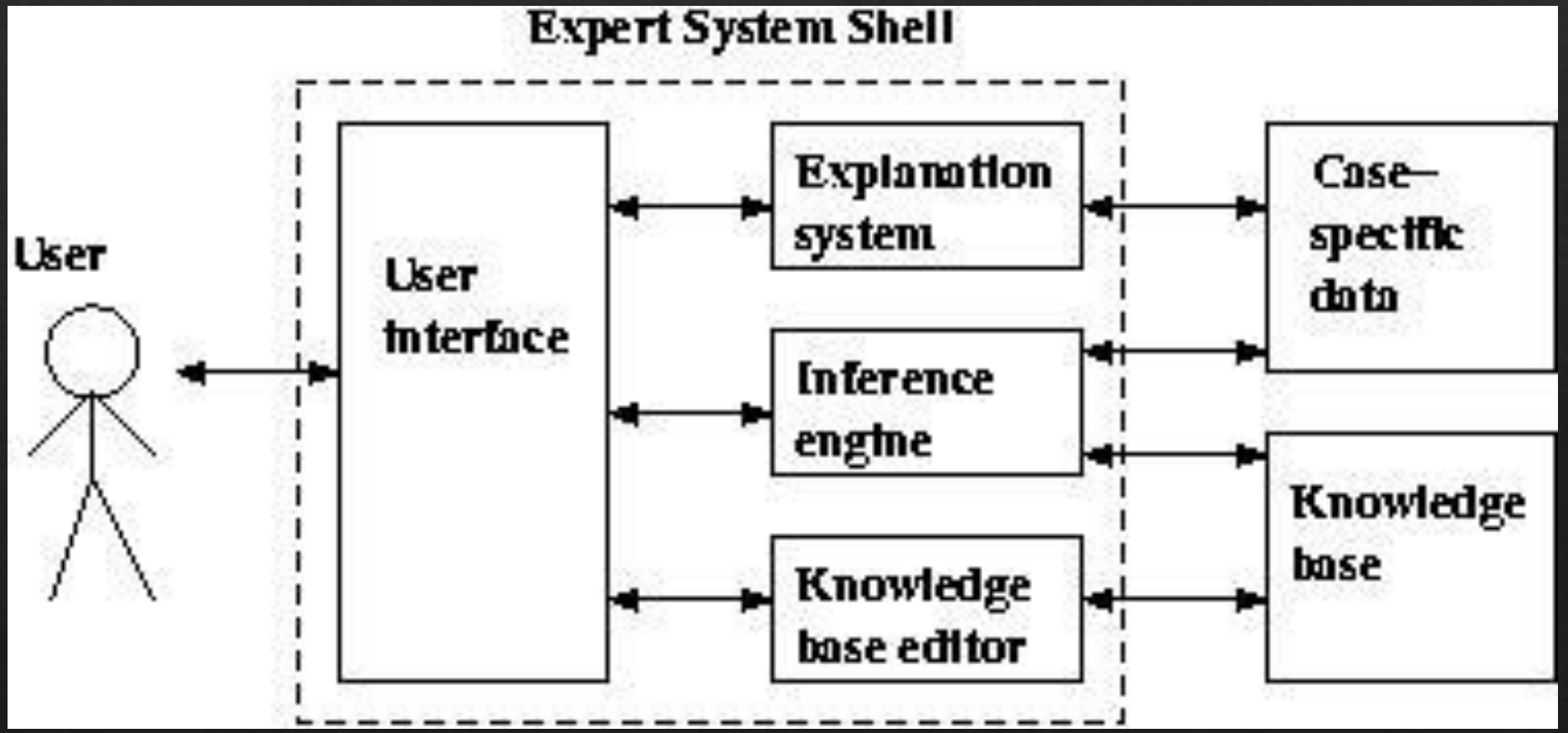
Explanation Subsystem Example

- System: Is there gas in the fuel tank?
  User: Yes.
  System: Is it true that the engine turns over?
  User: No.
  System Is it true that the lights come on?
  User: Why?
  System: I am trying to prove: problem with battery.
  It has been established that it is not the case that the engine turns over.
  Therefore if it is not the case that the lights come on then there is a problem with battery.
  Is it true that the lights come on?
  User: No.
  System: I conclude that there is a problem with the battery.
  User: How?
  System: This follows from Rule 2:
  IF NOT engine_turns_overAND ...

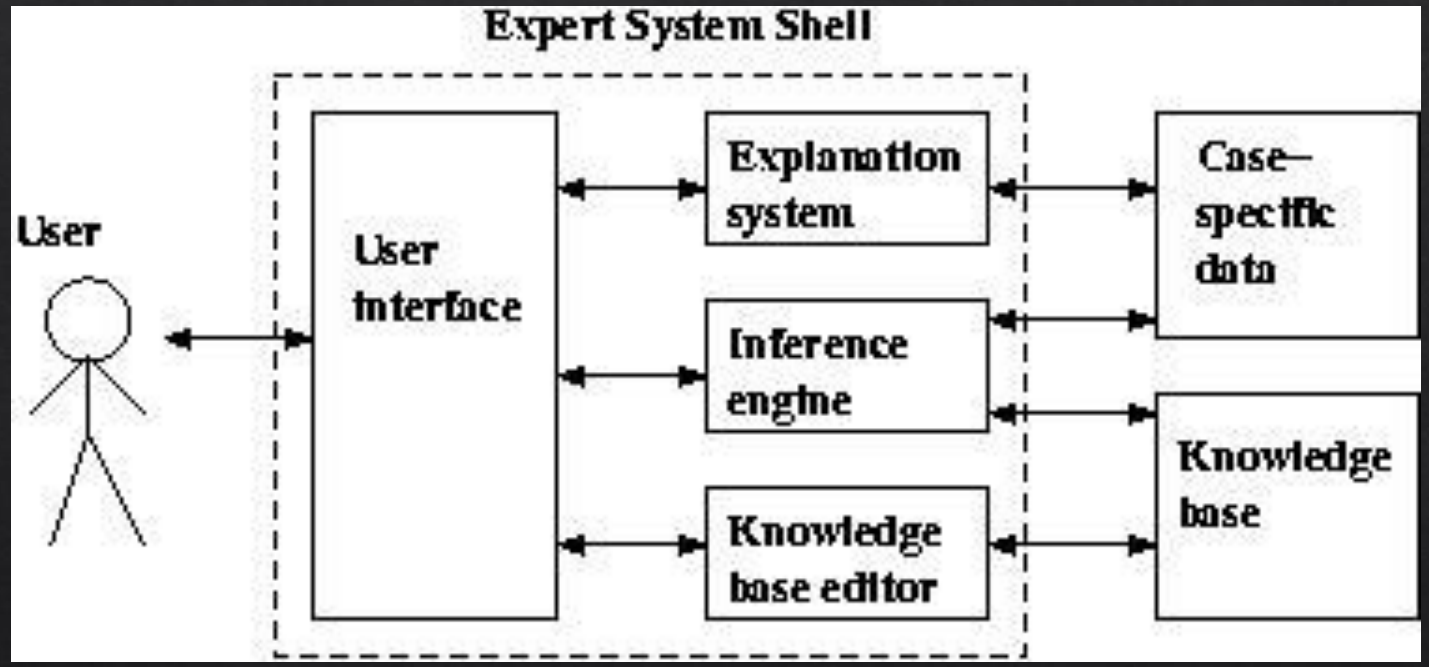
Building Blocks of Expert System



More General ES Architecture



More General ES Architecture



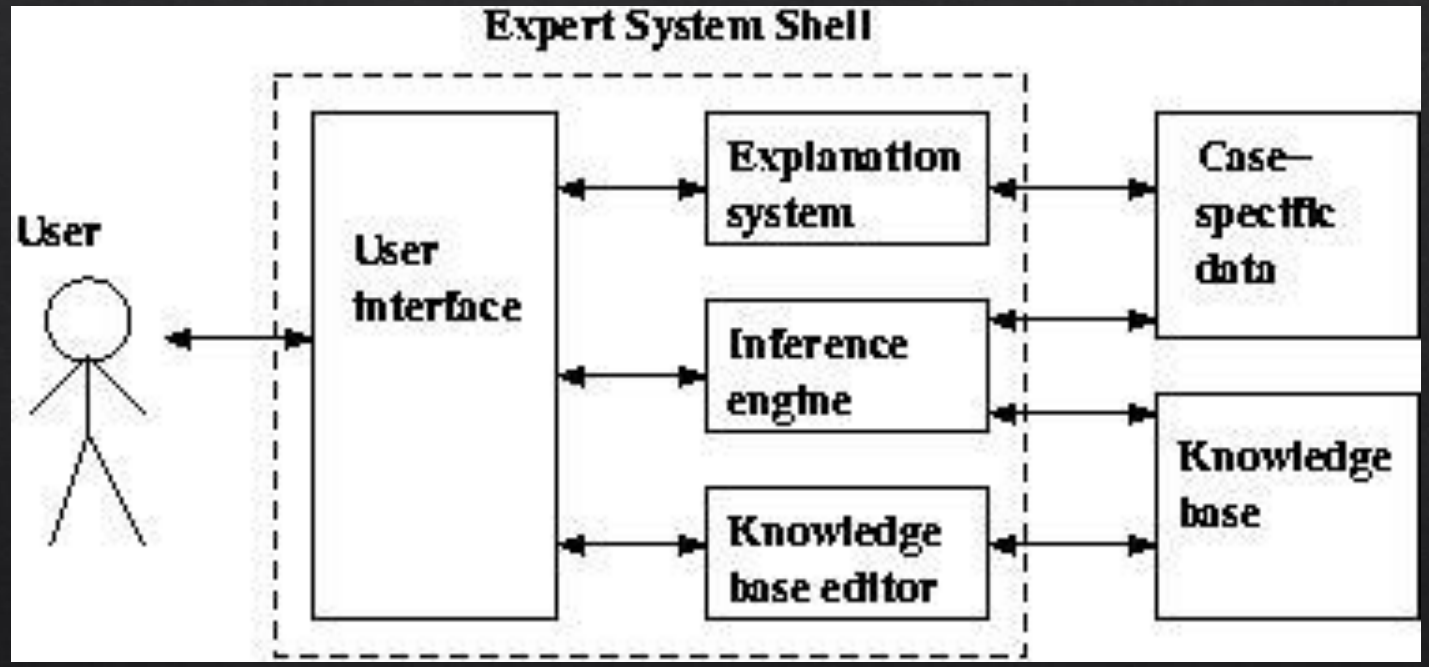
User Interface:
Menus, NLP, or any other interaction to interface with a human user

Explanation Subsystem:
Module that allows the program to explain its reasoning

Inference Engine:
Makes inferences about data given the knowledge base

Knowledge base editor:
Helps an expert or engineer tailor the knowledge base with a user-friendly tool

More General ES Architecture



Knowledge Base:
Knowledge as you know it. Inference Rules defined and general knowledge defined

Case-specific data:
Includes data provided by the user, data specific to one case, or partial conclusions constructed by inference rules.

Choosing A Problem

Creating an expert system can be VERY time consuming and expensive

Need domain experts

Need engineers

Need to build expert shell (like CLIPS) if don't already have it.

So when is this justified?

Choosing A Problem

- Cost-benefit analysis must make sense
- The “expert” is not already widely available
- Problem using symbolic reasoning
  - No dexterity or physical skill (but robotics getting better)
- Problem is well-structured and requires little common-sense knowledge
  - Rigorous technical knowledge easier to capture than fluffy common-sense knowledge

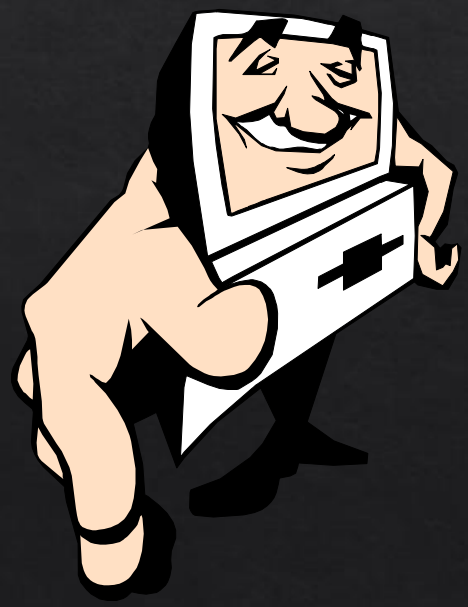
Choosing A Problem

- Problem doesn't already have a pure algorithmic solution
- Cooperative and articulate experts exist! Need to be able to find these domain experts
- Problem of proper size and scope
  - Why?
  - Requires expert knowledge, but only takes human an hour or so to finish

Who / What is involved?

User Interface

- **It enables the user to communicate with an expert system (Like CLIPS)**

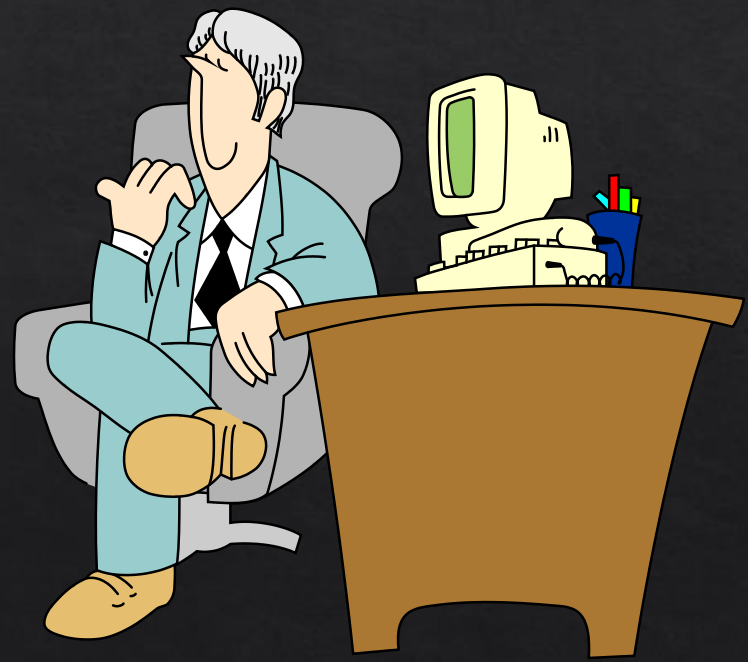


Knowledge Engineer

- A knowledge engineer is a computer scientist who knows how to design and implement programs that incorporate artificial intelligence techniques.

Domain Expert

- **A domain expert is an individual who has significant expertise in the domain of the expert system being developed.**



Knowledge Engineering

- **The art of designing and building the expert systems is known as KNOWLEDGE ENGINEERING knowledge engineers are its practitioners.**
- **Knowledge engineering relies heavily on the study of human experts in order to develop intelligent & skilled programs.**

Developing Expert Systems

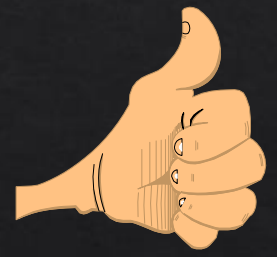
- **Determining the characteristics of the problem.**
- **Knowledge engineer and domain expert work together closely to describe the problem.**

- The engineer then translates the knowledge into a computer-usable language, and designs an inference engine, a reasoning structure, that uses the knowledge appropriately.

- He also determines how to integrate the use of uncertain knowledge in the reasoning process, and what kinds of explanation would be useful to the end user.

- When the expert system is implemented, it may be:
  - The inference engine is not just right
  - Form of representation of knowledge is awkward

- An expert system is judged to be entirely successful when it operates on the level of a human expert.



Human Expertise vs Artificial Expertise

1. **Perishable**
2. **Difficult to transfer**
3. **Difficult to document**
4. **Unpredictable**
5. **Expensive**

1. **Permanent**
2. **Easy to transfer**
3. **Easy to document**
4. **Consistent**
5. **Affordable**

Two Big Problems

- Knowledge takes a long time to build
- Knowledge is always changing
- How might you deal with this?

Automatic Knowledge-Acquisition

- Having domain experts create knowledge and update / maintain it is tedious
- Automatic Knowledge Acquisition is a set of techniques for automatically acquiring, validating, and incorporating new knowledge into an expert knowledge base

Automatic Knowledge-Acquisition

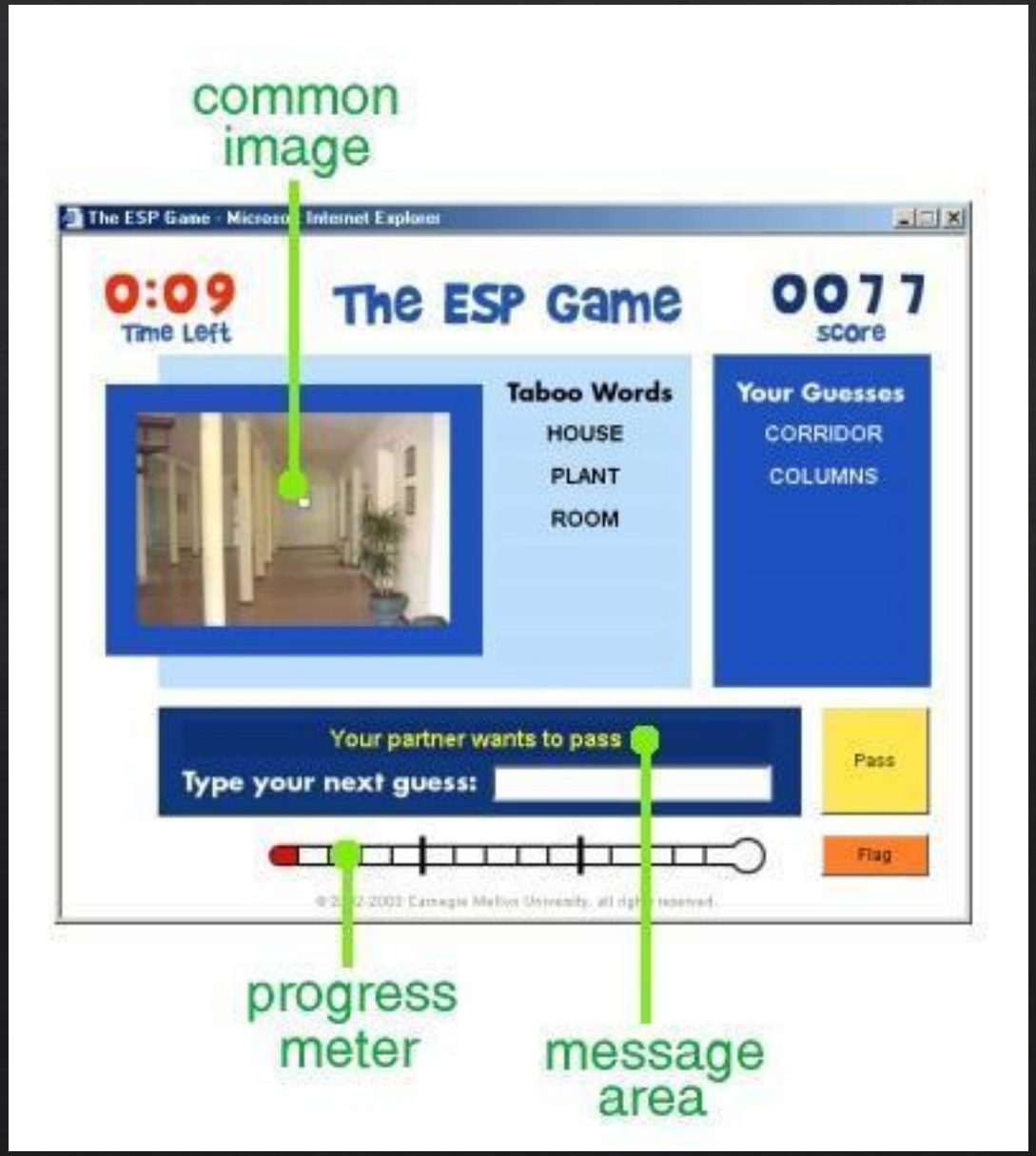
- Natural Language Processing:
  - Try to parse Wikipedia, technical documents, etc. to automatically build your knowledge base.
- Usually this would be just a first pass (as it doesn't work that well).
  - Then, some process for an expert to clean-up the knowledge

Crowdsourcing Knowledge

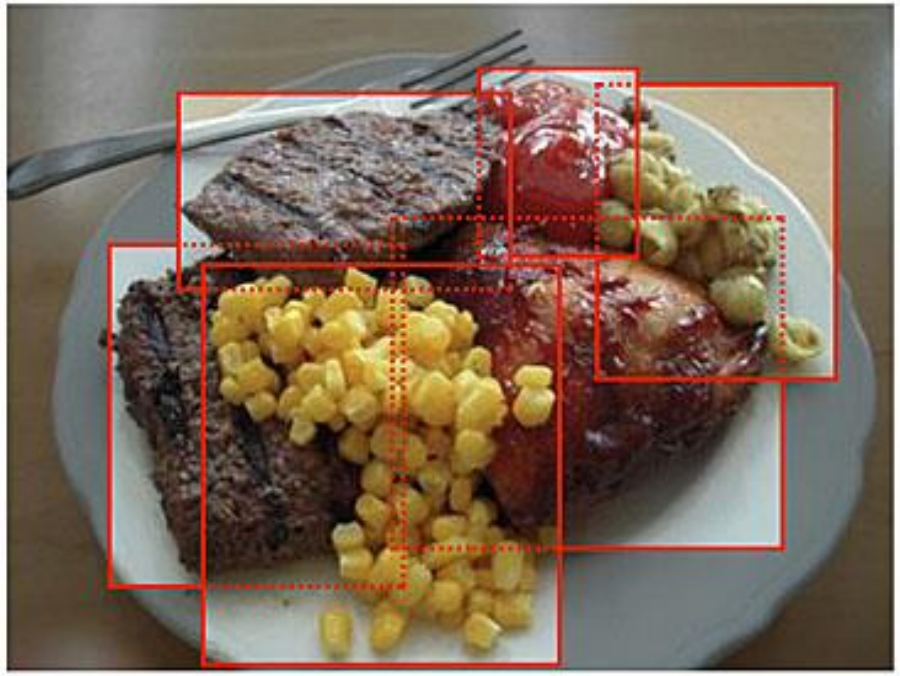
- Using the “crowd” to obtain the information you need.

- Amazon Mechanical Turk does this.
  - You need something translated
  - You can split each sentence or paragraph into a micro-task
  - You pay people to complete these tasks
  - “Turkers” in the crowd do the work for you

Crowdsourcing Knowledge



Crowdsourcing Knowledge



DINNER	1573.2	72.9	84	138.9
Yellow Corn (0.50 cup)	303	3.9	61.6	7.8
barbeque chicken breast				
Chicken Breast Meat and Skin (Broilers or Fryers) (1.00 breast, bone removed)	249	13.4	0	30.2
Barbeque Sauce (Low Sodium, Canned) (0.14 cup)	26.6	0.6	4.5	0.6
Beef Steak (0.92 medium steak (yield after cooking, bone removed))	471.3	28.1	0	51.0
Hominy (White, Canned) (0.44 cup)	52.8	0.6	10.4	1.1
Ketchup (2.00 tbsp)	30	0.1	7.5	0.5
Beef Steak (0.86 medium steak (yield after cooking, bone removed))	440.5	26.2	0	47.7

+ **Add Food**

Delete this photo

Crowdsourcing Knowledge

Crowdsourcing Knowledge

- Why would someone help you!!!!
- Turns out most crowdsourcing tools use one of three strategies:
  - Money (I'll pay you to help me with this)
  - Philanthropy (Your work will help soooo many people!)
  - Fun (It's a game! It's "fun"!)

Some Prominent Expert Systems

- **Dendral**
- **Dipmeter Advisor**
- **Mycin**
- **R1/Xcon**

MYCIN

- Developed in the 1970s at Stanford
- Job was to diagnose certain blood infections (pretty specific task)
- One of the first successful expert systems
  - Rule-Based, so very similar to example we saw
- Written in LISP (Wooooooooo…LISP!!!!)
- Goal-directed system (uses backward chaining primarily)

MYCIN

- Used backward chaining…but
  - As it is backward chaining, would ask the patient questions if it hit a variable that it needed an answer to.
    - Example: if chaining runs into “User feels dizzy” and doesn’t know, it would prompt user to answer
    - This would sometimes lead to user being asked MANY questions
  - Used heuristics to simplify the backward chaining search
    - Some paths ignored and not explored
  - Also would ask general questions to all patients to try to simplify the search process

MYCIN

- Example Rule:

- IF the infection is primary-bacteremia
  AND the site of the culture is one of the sterile sites
  AND the suspected portal of entry is the gastrointestinal tract
  THEN there is suggestive evidence (0.7) that infection is bacteroid.

- *Note the probabilistic result (0.7). These probabilities were hardcoded. More recent advances calculate these on the fly (we'll see this later)

SUMMARY: Expert Systems

- Good for developing knowledge of domain experts in a particular field.

- Advantages:
  - Can explain reasoning
  - One system shell can be used to develop system in many domains
  - Often very efficient
  - Can often deal with most “common” issues, freeing up human experts to deal with rare problems

- Disadvantages:
  - Can be expensive and time-consuming to build (need domain experts, engineers, etc.)
  - Can be hard to maintain if knowledge is always changing (knowledge acquisition techniques help)
  - Not super general. Most systems solve a VERY specific task.