Strong Method Problem Solving

- Expert System Technology
- Rule-Based Expert Systems
- Model-Based, Case-Based and Hybrid Systems
- Planning

Expert Systems

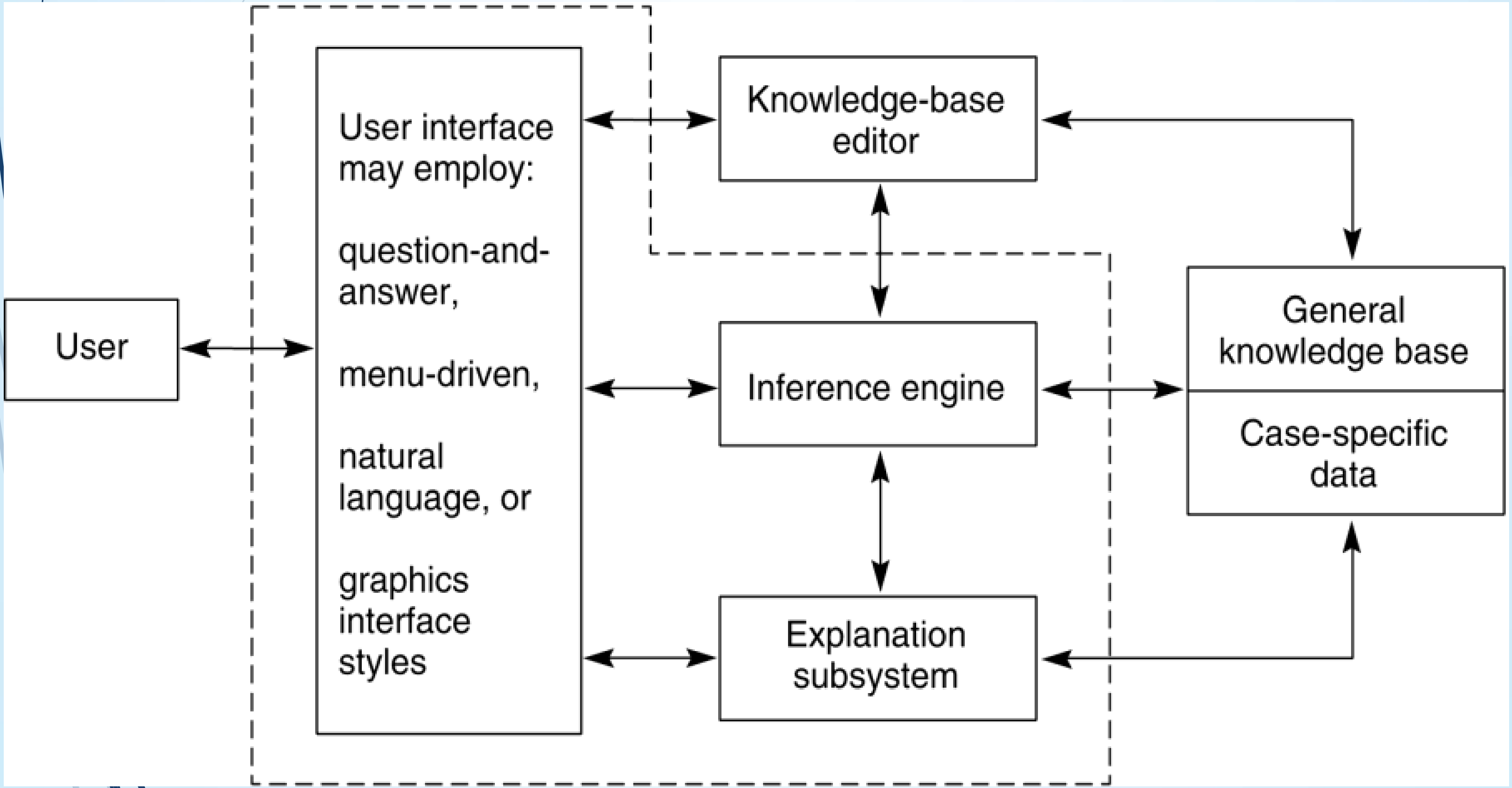
- Knowledge intensive method
- Expert systems take advantage of expertise of human domain experts
- Human expertise as heuristics
  - Support inspection of reasoning processes, presenting intermediate steps and answering questions about the solution process
  - Allow easy modification in adding and deleting skills from KB
  - Reason heuristically, using knowledge to get useful solutions

Expert System Features

- Rich KB and easy modification
- Open reasoning process
- Explanation of reasoning
- Exploratory prototyping
- Heuristic reasoning (problem-solving)
- Wide range applications
  - Interpretation – from raw data to conclusion
  - Prediction – weather
  - Diagnosis – medical, mechanical
  - Design – CAD, Civil
  - Planning – scheduling, Robot
  - Monitoring – Inspector
  - Instruction – CAE, online learning
  - Control – production line control

Architecture of Expert Systems

Architecture of a typical expert system for a particular problem domain.



Modules of Expert Systems

- User interface
  - simplify communication and hide much of the complexity, such as the internal structure of the knowledge base
- Knowledge base
  - Domain knowledge
  - General knowledge and case-specific information
  - Optional KB editor
- Inference engine
  - Reasoning mechanism to apply the knowledge to the solution of problems
  - E.g. recognize-act control cycle in production systems
- Explanation subsystem
  - Allow the program to explain its reasoning to the user, to justify the conclusion
  - Respond to why and how queries

Separation Knowledge Base from Inference Engine

- Represent knowledge in a more natural fashion
- Builders can focus on capturing and organizing domain knowledge
- Modularization, easy to maintain – modification, addition, removal, etc
- Build expert system shell that can be provided with different domain knowledge to form different ES

Selecting Problems for ES

Guidelines to determine whether a problem is appropriate for expert system solution:

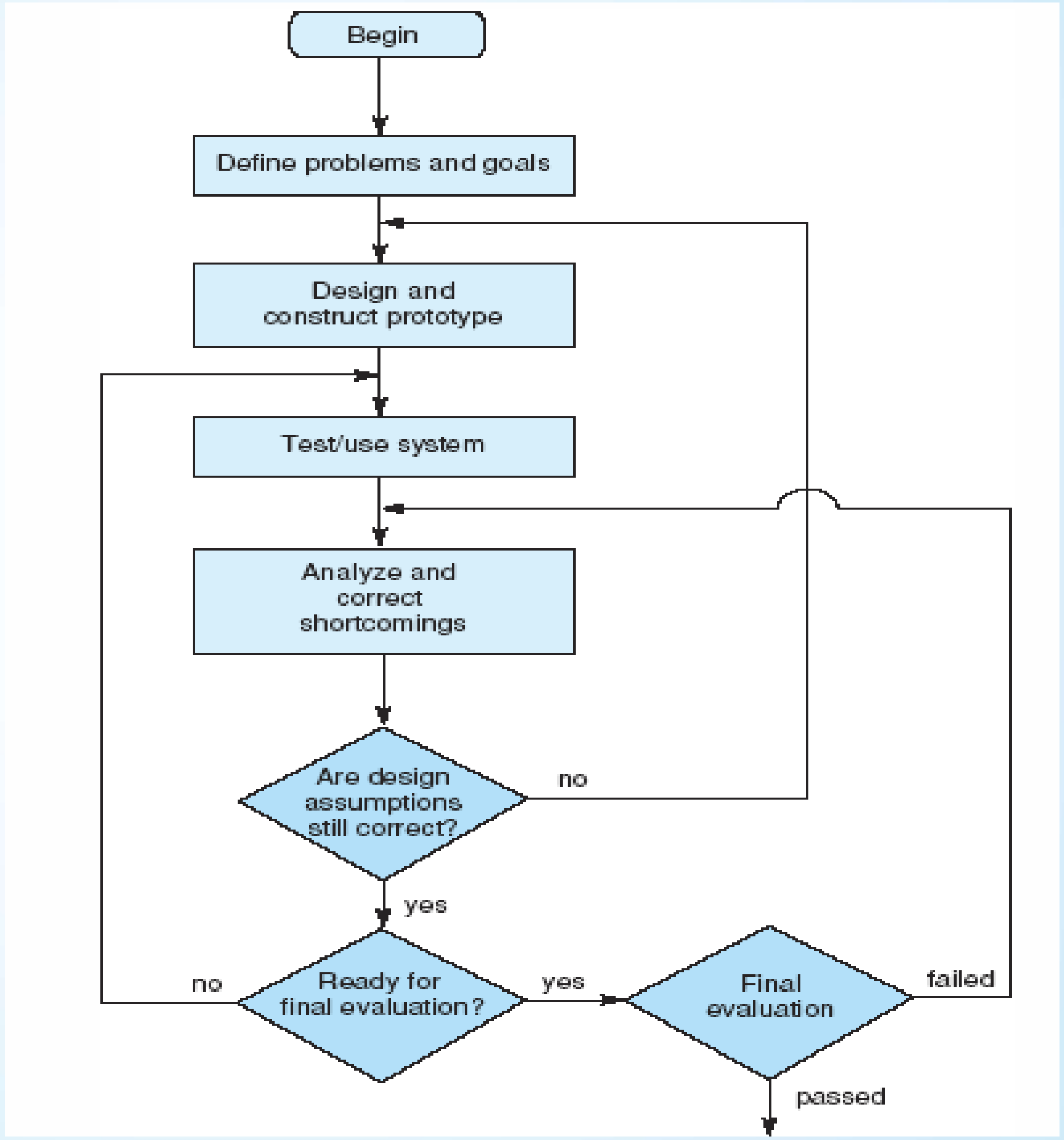
1. The need for the solution justifies the cost and effort of building an expert system.
2. Human expertise is not available in all situations where it is needed.
3. The problem may be solved using symbolic reasoning.
4. The problem domain is well structured and does not require commonsense reasoning.
5. The problem may not be solved using traditional computing methods.
6. Cooperative and articulate experts exist.
7. The problem is of proper size and scope.

Roles in Building ES

- Knowledge engineer
  - AI language and representation expert
  - Select software and hardware, acquire and organize knowledge in a specific form
- Domain expert
  - Professionals
  - Provide knowledge of the application area
- End user
  - Specify requirements and constraints
  - Test and verify the product

ES Exploratory Development Cycle

- ES Programming
  - Prototyping development
  - Unlimited maintenance

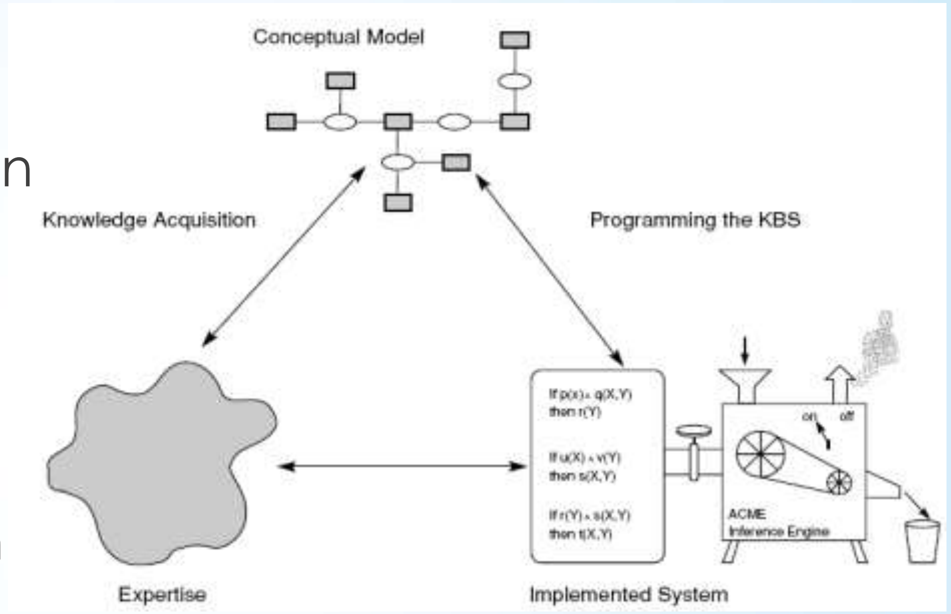

Begin
Define problems and goals
Design and construct prototype
Test/use system
Analyze and correct shortcomings
Are design assumptions still correct?
no
yes
Ready for final evaluation?
no
yes
Final evaluation
failed
passed

Knowledge Acquisition

- Acquire expertise (domain knowledge) from domain experts
- Characteristics of domain expertise
  - Inaccessible -- Perceptible but non-describable
  - Free format
  - Vague, imprecise, and bias
  - Dynamic (change)

Knowledge Engineering

- Engineering process
  - Knowledge acquisition
  - Knowledge representation
  - System implementation
  - System maintenance
- Conceptual Model
  - Intermediate representation of domain knowledge, like ER diagram in DB design



Rule-Based Expert Systems

- Features
  - Knowledge base organized as a set of ***if … then …*** rules
  - Lead to the ES architecture
  - Natural
  - Widely used
- Production system vs. Rule-based ES
  - Similarity: both use rules
  - Production system is a special case of rule-based systems: production *Condition → Action* can be considered as *if Condition then Action*
  - Differences: Production systems implement graph search with either goal-driven or data-driven strategy, while rule-based ES implements logical reasoning

A Production System

A small expert system for analysis of automotive problems.

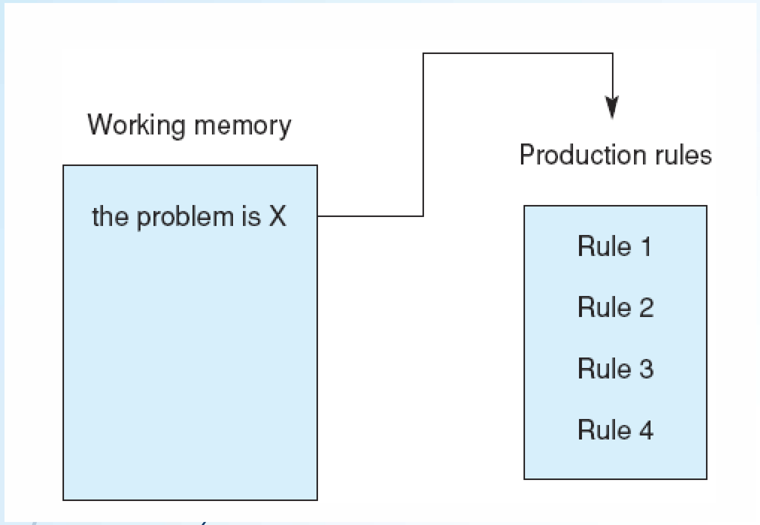
Rule 1:
if the engine is getting gas,
and
the engine will turn over,
then
the problem is spark plugs.

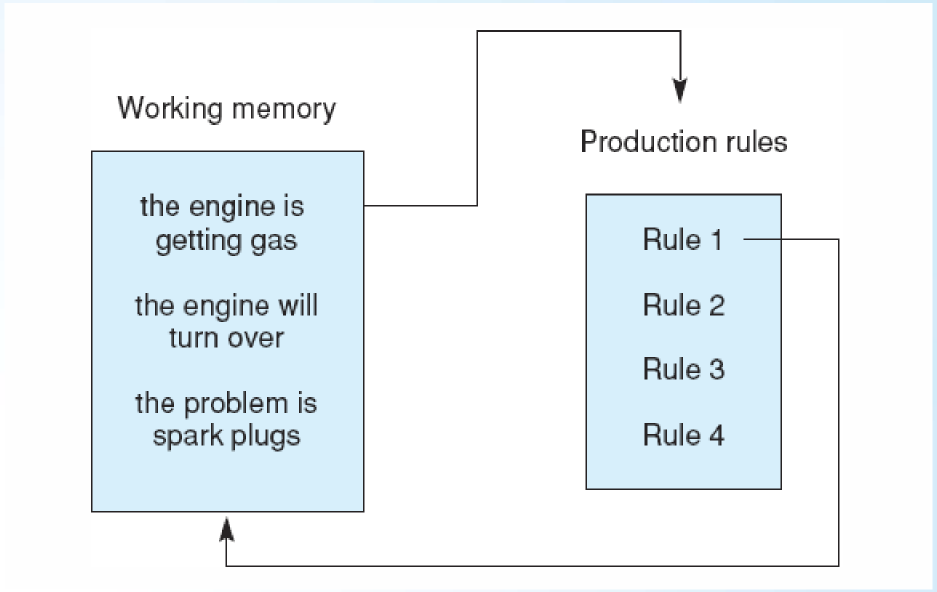
Rule 2:
If the engine does not turn over,
and
the lights do not come on
then
the problem is battery or cables.

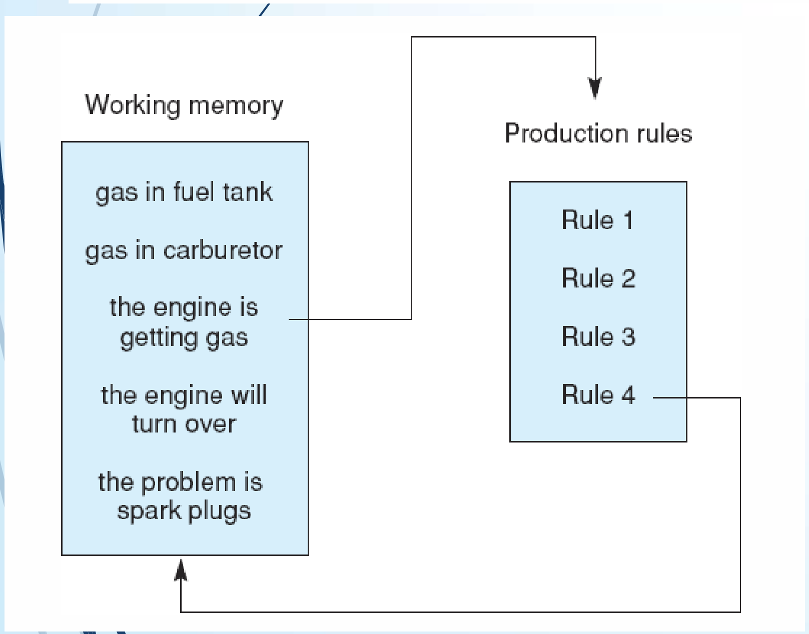
Rule 3:
If the engine does not turn over,
and
the lights do come on
then
the problem is the starter motor.

Rule 4:
If there is gas in the fuel tank,
and
there is gas in the carburettor
then
the engine is getting gas.

Goal-Driven Problem Solving



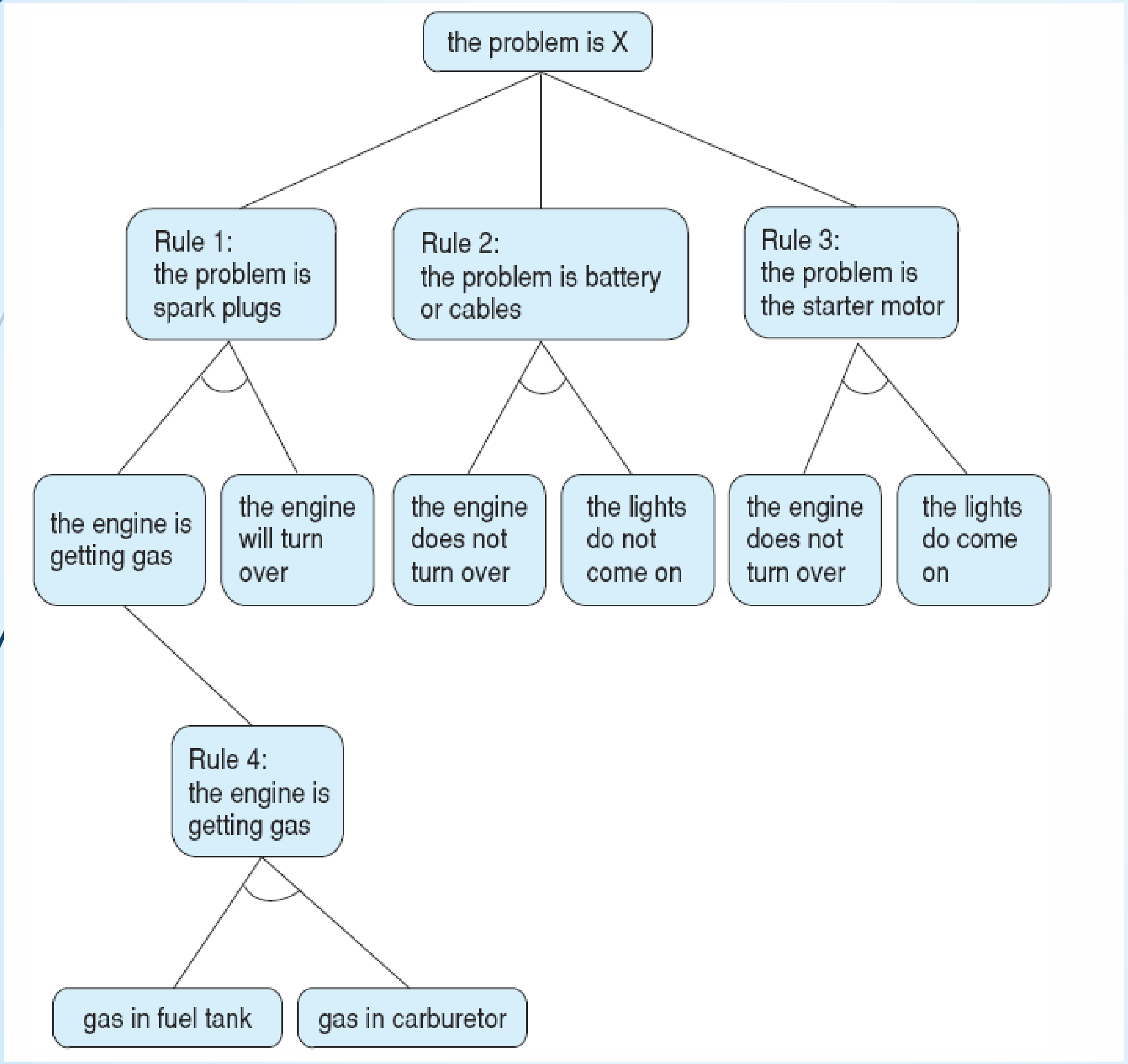

Working memory
the engine is getting gas
the engine will turn over
the problem is spark plugs
Production rules
Rule 1
Rule 2
Rule 3
Rule 4



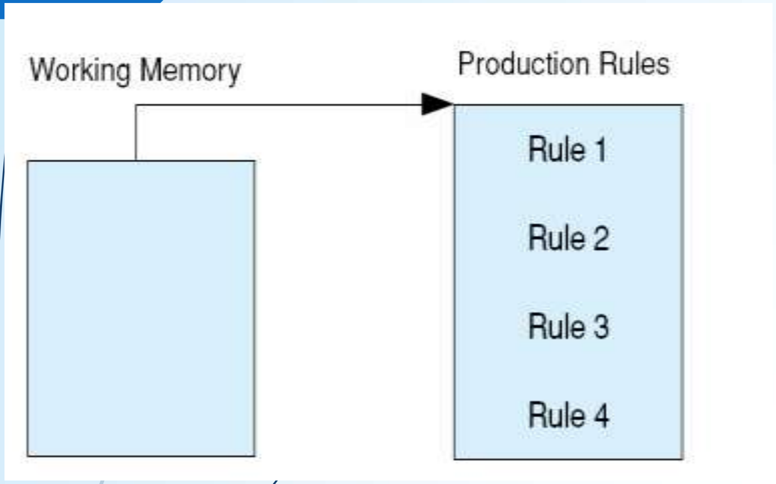
1. The production system at the start
2. The production system after Rule 1 has fired.
3. The system after Rule 4 has fired.

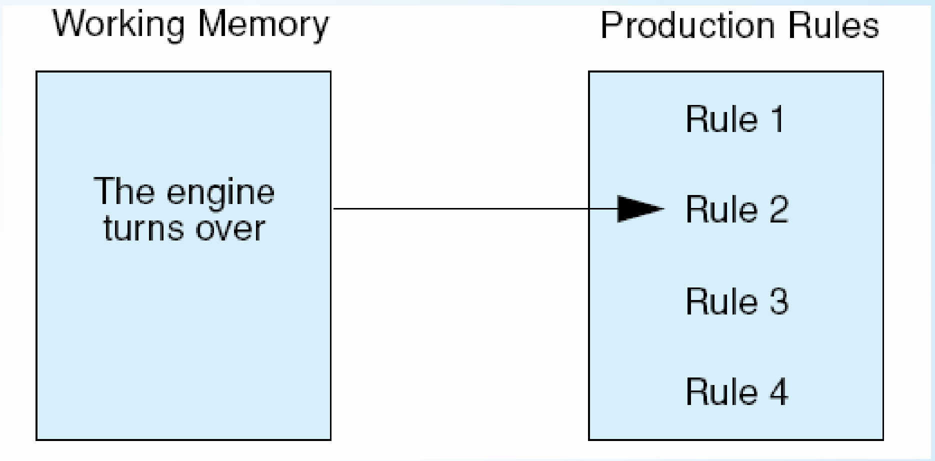
- ❖ Note the stack-based approach to goal reduction
- ❖ Depth-first search

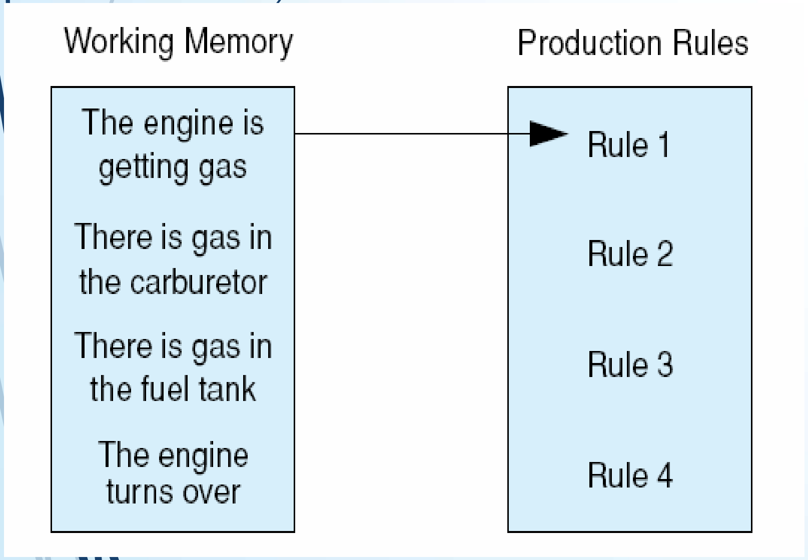
The and/or graph searched in the car diagnosis example, with the conclusion of Rule 4 matching the first premise of Rule 1.


the problem is X
Rule 1: the problem is spark plugs
Rule 2: the problem is battery or cables
Rule 3: the problem is the starter motor
the engine is getting gas
the engine will turn over
the engine does not turn over
the lights do not come on
the engine does not turn over
the lights do come on
Rule 4: the engine is getting gas
gas in fuel tank
gas in carburetor

Data-Driven Problem Solving






Working Memory
Production Rules
The engine is getting gas
There is gas in the carburetor
There is gas in the fuel tank
The engine turns over
Rule 1
Rule 2
Rule 3
Rule 4

1. The production system at the start.
2. The production system after evaluating the first premise of Rule 2, which then fails.
3. The data-driven production system after considering Rule 4, beginning its second pass through the rules.

❖ Breadth-first search

Explanations in ES

- Two general questions to be answered by rule-based ESs: Why and How
- Explanation in goal-driven reasoning
  - Why
    - When the system asks the user for information and the user responds "why to ask"
    - The answer is the current rule
  - How
    - When the system conclude a solution and the user asks "how to conclude"
    - The answer is the sequence of rules that have been used
- Explanation in data-driven reasoning
  - Less-focused on search
  - Limited explanation

Explanation in Goal-Driven Reasoning

The following dialogue begins with the computer asking the user about the goals present in working memory.

Gas in fuel tank?
Yes
Gas in carburetor?
Yes
Engine will turn over?
Why

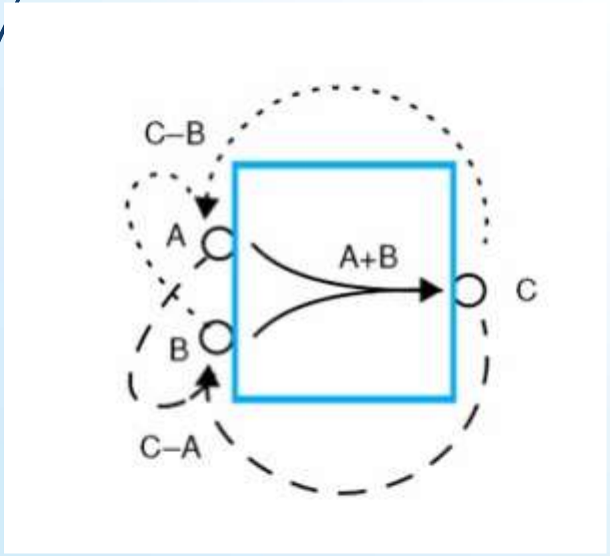
It has been established that:
1. The engine is getting gas,
2. The engine will turn over,
Then the problem is the spark plugs.

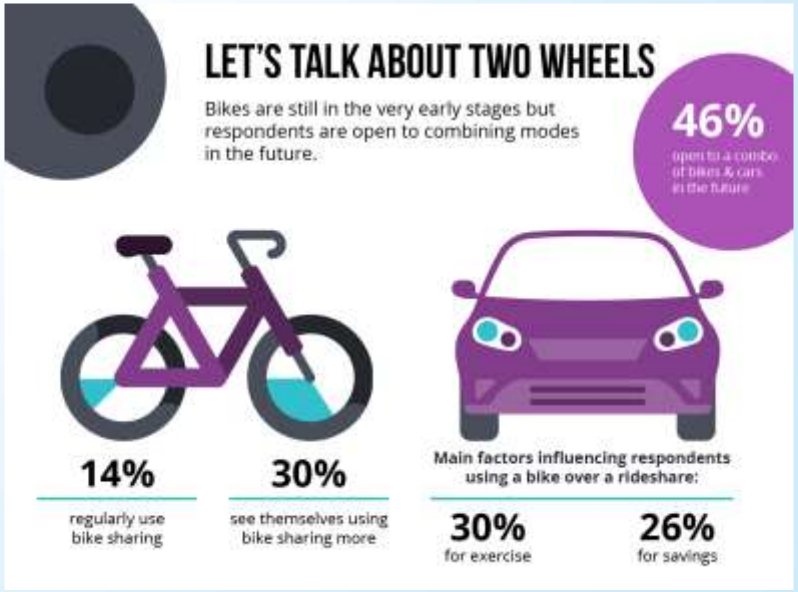
How the engine is getting gas

This follows from rule 4:
if
gas in fuel tank, and
gas in carburetor
then
engine is getting gas.
gas in fuel tank was given by the user
gas in carburetor was given by the user

Model Based Reasoning

- An inference method used in expert systems based on a model of the physical world.
- The main focus of application development is developing the model.
- Then at run time, an "engine" combines this model knowledge with observed data to derive conclusions such as a diagnosis or a prediction.
- In a model-based reasoning system knowledge can be represented using causal rules.




LET'S TALK ABOUT TWO WHEELS
Bikes are still in the very early stages but respondents are open to combining modes in the future.
46%
open to a combo of bikes & cars in the future
14%
regularly use bike sharing
30%
see themselves using bike sharing more
Main factors influencing respondents using a bike over a rideshare:
30%
for exercise
26%
for savings

Model-based reasoning

Development of a bicycle-sharing program for a city.

Model Creation: City transportation infrastructure, including existing bike lanes, road networks, public transportation routes, and population distribution.

Alternative Designs: Include different placement strategies for bike stations, variations in the number and types of bikes available, and pricing models for usage.

Simulation and Analysis: Simulate designs considering factors such as the demand for bikes at different times of day, the impact on traffic congestion, the accessibility of bike stations to different neighbourhoods, and the potential environmental benefits.

Evaluation and Decision-Making: Evaluate the strengths and weaknesses of each design option such as user convenience, cost-effectiveness, environmental sustainability, and social equity.

Implementation: Select most promising design for the bicycle-sharing program and work on its implementation involves securing funding, coordinating with city agencies and private partners, and conducting public outreach and education campaigns.

Case-Based Reasoning

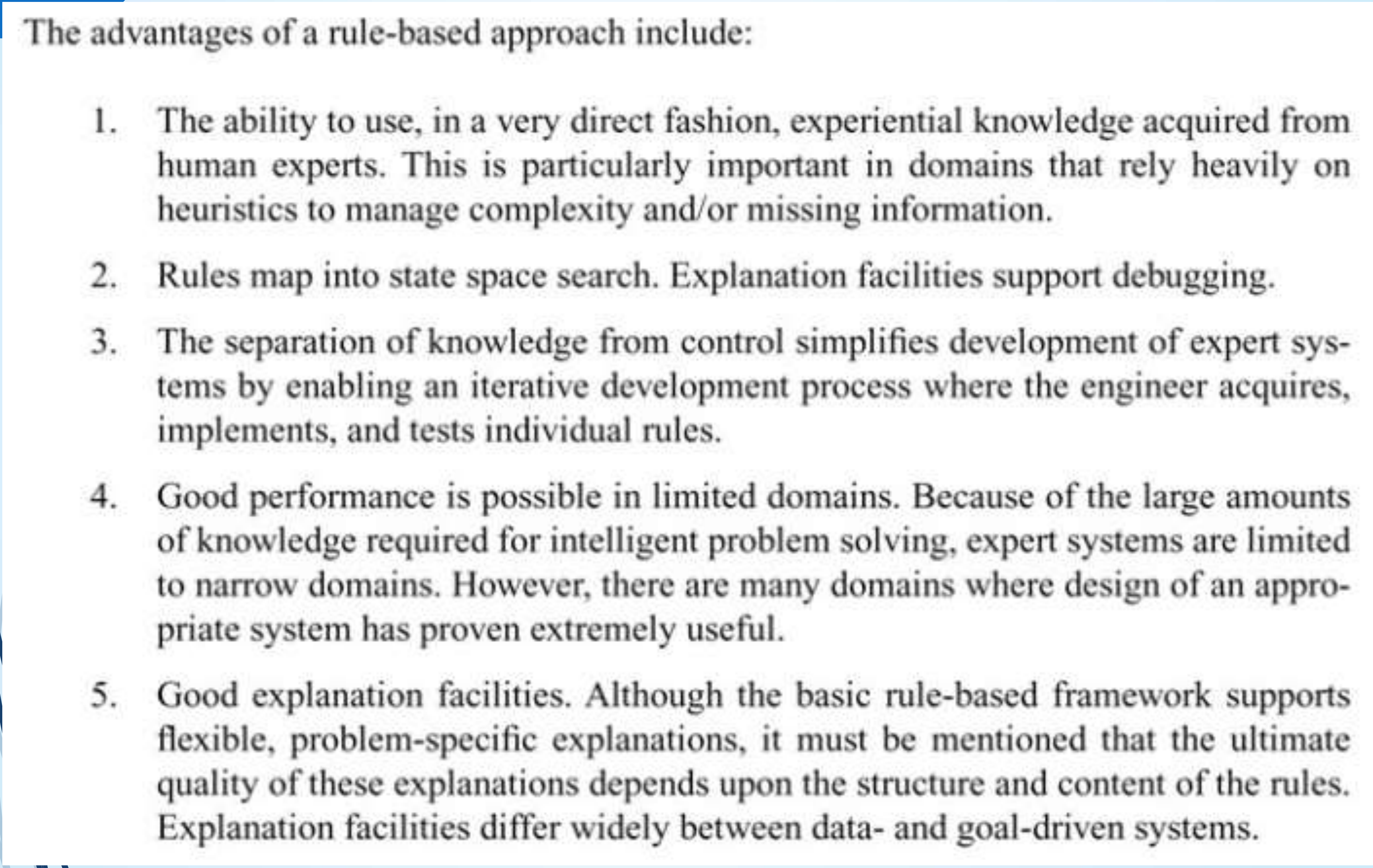
- CBR is the process of solving new problems based on the solutions of similar past problems.
- An auto mechanic who fixes an engine by recalling another car that exhibited similar symptoms is using case-based reasoning.
- A lawyer who advocates a particular outcome in a trial based on legal precedents or a judge who creates case law using case-based reasoning.
- Consider a movie recommendation system like Netflix.
  - When a user logs in, the system retrieves similar users' past viewing habits and recommends movies or TV shows based on those similarities.
  - If a user enjoys action movies and science fiction, the system might recommend a new release that falls into these categories.
  - As the user watches and rates movies, the system learns more about their preferences and can refine its recommendations over time.

Case-based reasoners share a common structure

- For each new problem they:
  - Retrieve appropriate cases from memory.
  - Modify a retrieved case so that it will apply to the current situation
  - Apply the transformed case
  - Save the solution, with a record of success or failure, for future use

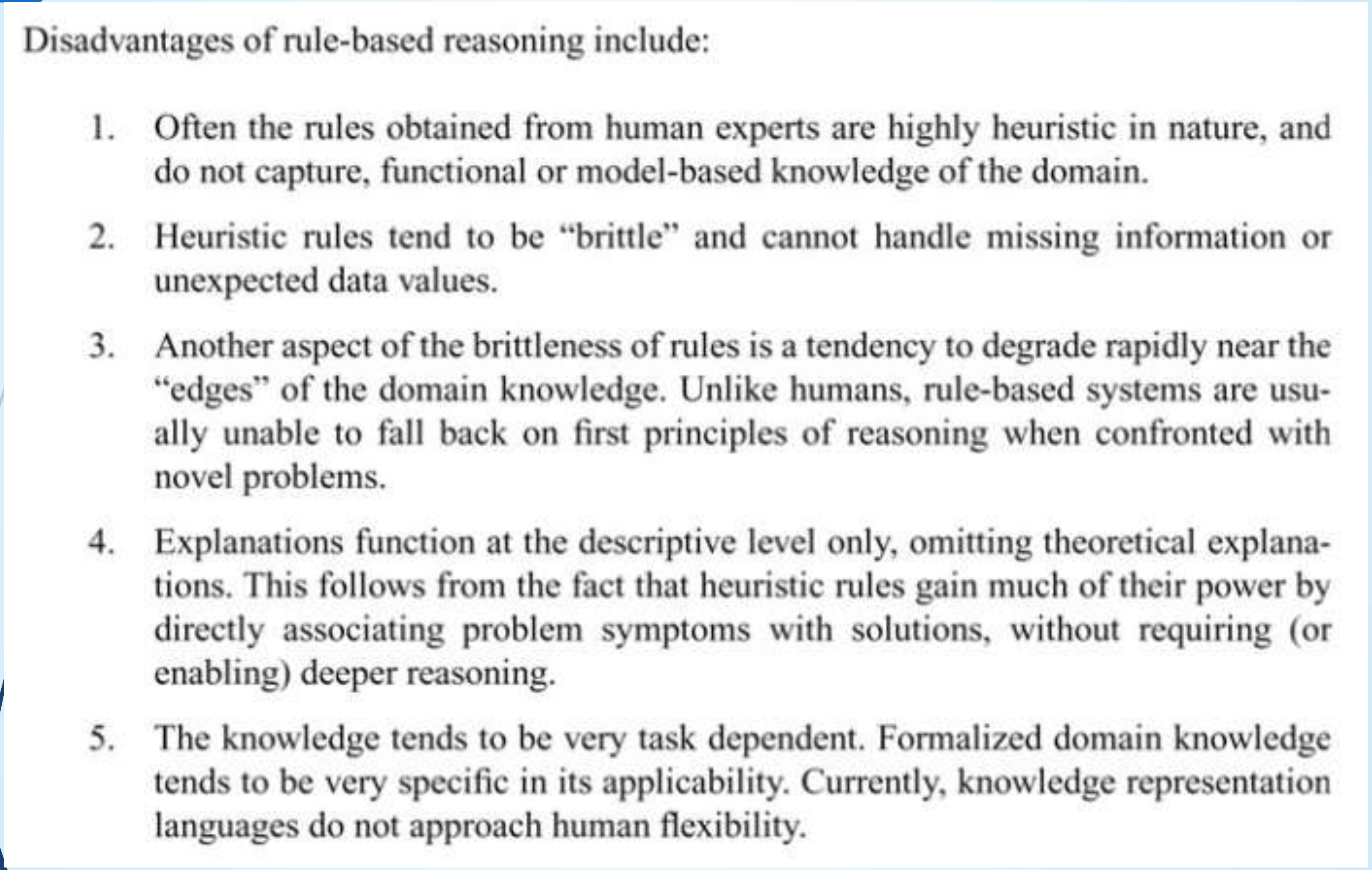
The storage and retrieval of cases.

- Kolodner (1993)
- **Goal-directed preference.** Organize cases, at least in part, by goal descriptions. Retrieve cases that have the same goal as the current situation
- **Salient-feature preference.** Prefer cases that match the most important features or those matching the largest number of important features
- **Specify preference.** Look for as exact as possible matches of features before considering more general matches
- **Frequency preference.** Check first the most frequently matched cases
- **Recency preference.** Prefer cases used most recently Ease of adaptation preference. Use first cases most easily adapted to the current situation.



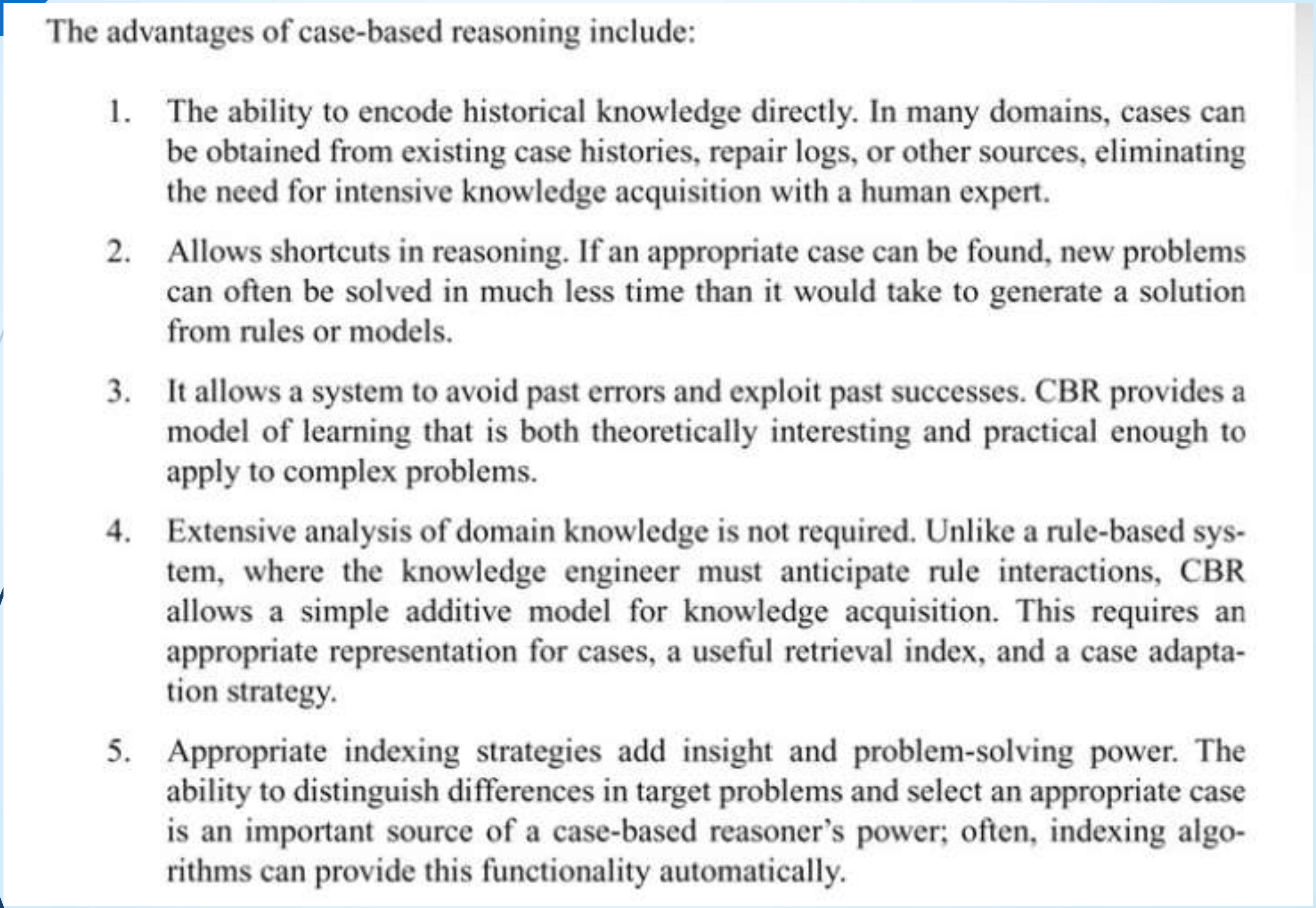
The advantages of a rule-based approach include:

1. The ability to use, in a very direct fashion, experiential knowledge acquired from human experts. This is particularly important in domains that rely heavily on heuristics to manage complexity and/or missing information.
2. Rules map into state space search. Explanation facilities support debugging.
3. The separation of knowledge from control simplifies development of expert systems by enabling an iterative development process where the engineer acquires, implements, and tests individual rules.
4. Good performance is possible in limited domains. Because of the large amounts of knowledge required for intelligent problem solving, expert systems are limited to narrow domains. However, there are many domains where design of an appropriate system has proven extremely useful.
5. Good explanation facilities. Although the basic rule-based framework supports flexible, problem-specific explanations, it must be mentioned that the ultimate quality of these explanations depends upon the structure and content of the rules. Explanation facilities differ widely between data- and goal-driven systems.



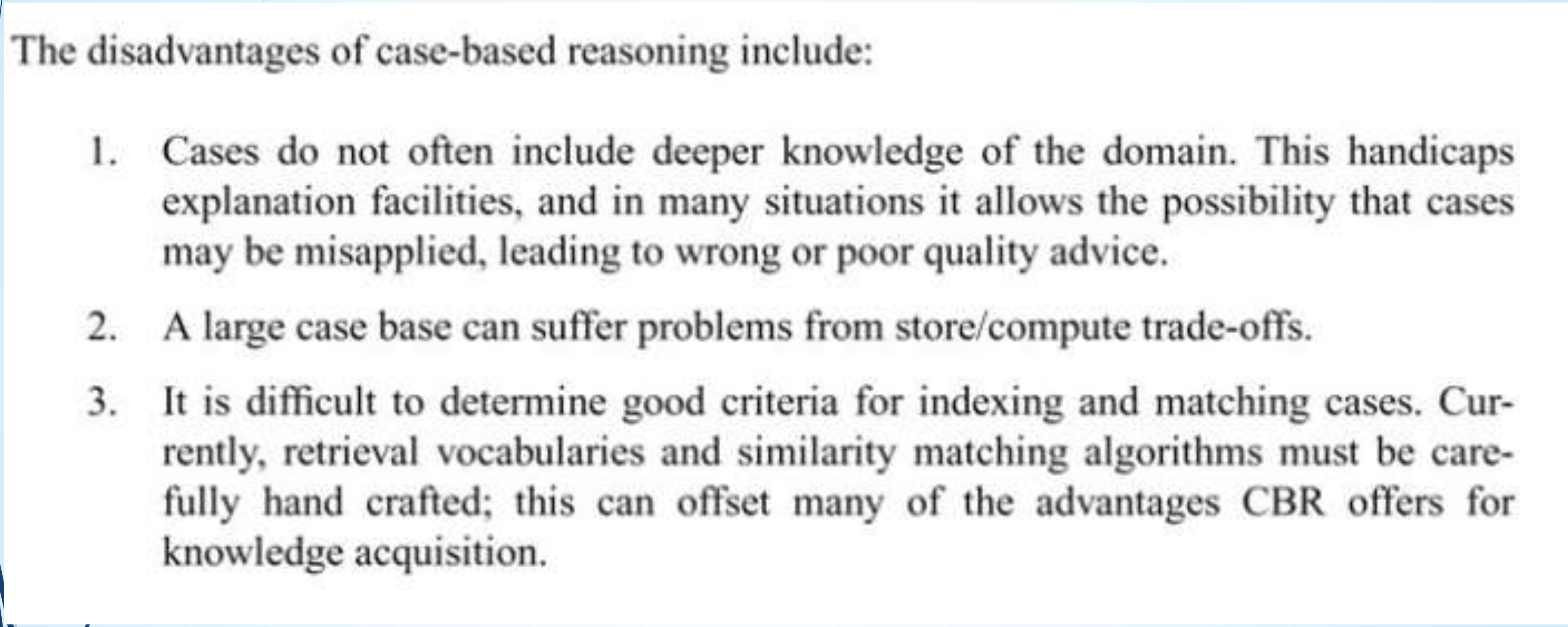
Disadvantages of rule-based reasoning include:

1. Often the rules obtained from human experts are highly heuristic in nature, and do not capture, functional or model-based knowledge of the domain.
2. Heuristic rules tend to be "brittle" and cannot handle missing information or unexpected data values.
3. Another aspect of the brittleness of rules is a tendency to degrade rapidly near the "edges" of the domain knowledge. Unlike humans, rule-based systems are usually unable to fall back on first principles of reasoning when confronted with novel problems.
4. Explanations function at the descriptive level only, omitting theoretical explanations. This follows from the fact that heuristic rules gain much of their power by directly associating problem symptoms with solutions, without requiring (or enabling) deeper reasoning.
5. The knowledge tends to be very task dependent. Formalized domain knowledge tends to be very specific in its applicability. Currently, knowledge representation languages do not approach human flexibility.



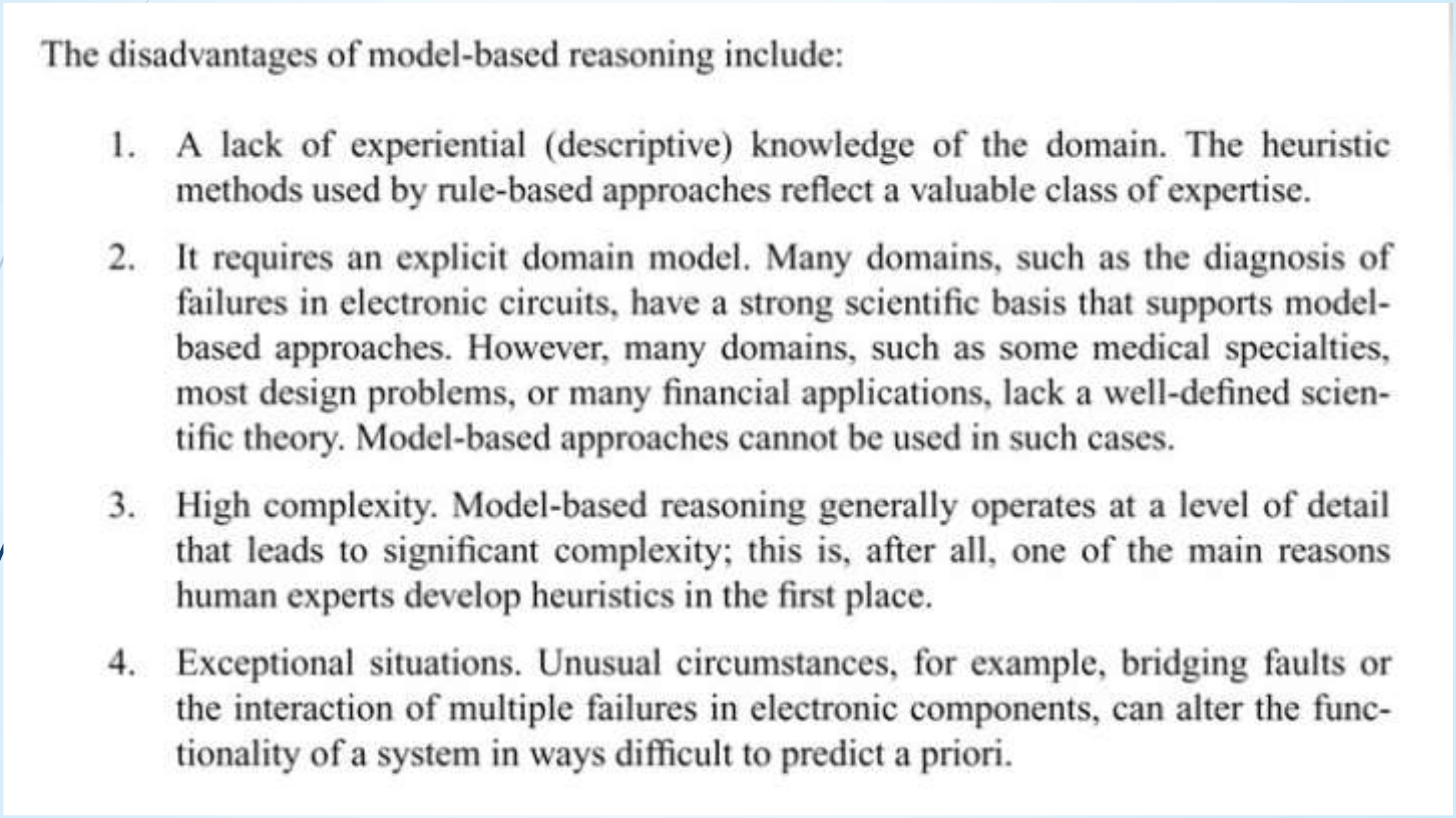
The advantages of case-based reasoning include:

1. The ability to encode historical knowledge directly. In many domains, cases can be obtained from existing case histories, repair logs, or other sources, eliminating the need for intensive knowledge acquisition with a human expert.
2. Allows shortcuts in reasoning. If an appropriate case can be found, new problems can often be solved in much less time than it would take to generate a solution from rules or models.
3. It allows a system to avoid past errors and exploit past successes. CBR provides a model of learning that is both theoretically interesting and practical enough to apply to complex problems.
4. Extensive analysis of domain knowledge is not required. Unlike a rule-based system, where the knowledge engineer must anticipate rule interactions, CBR allows a simple additive model for knowledge acquisition. This requires an appropriate representation for cases, a useful retrieval index, and a case adaptation strategy.
5. Appropriate indexing strategies add insight and problem-solving power. The ability to distinguish differences in target problems and select an appropriate case is an important source of a case-based reasoner's power; often, indexing algorithms can provide this functionality automatically.

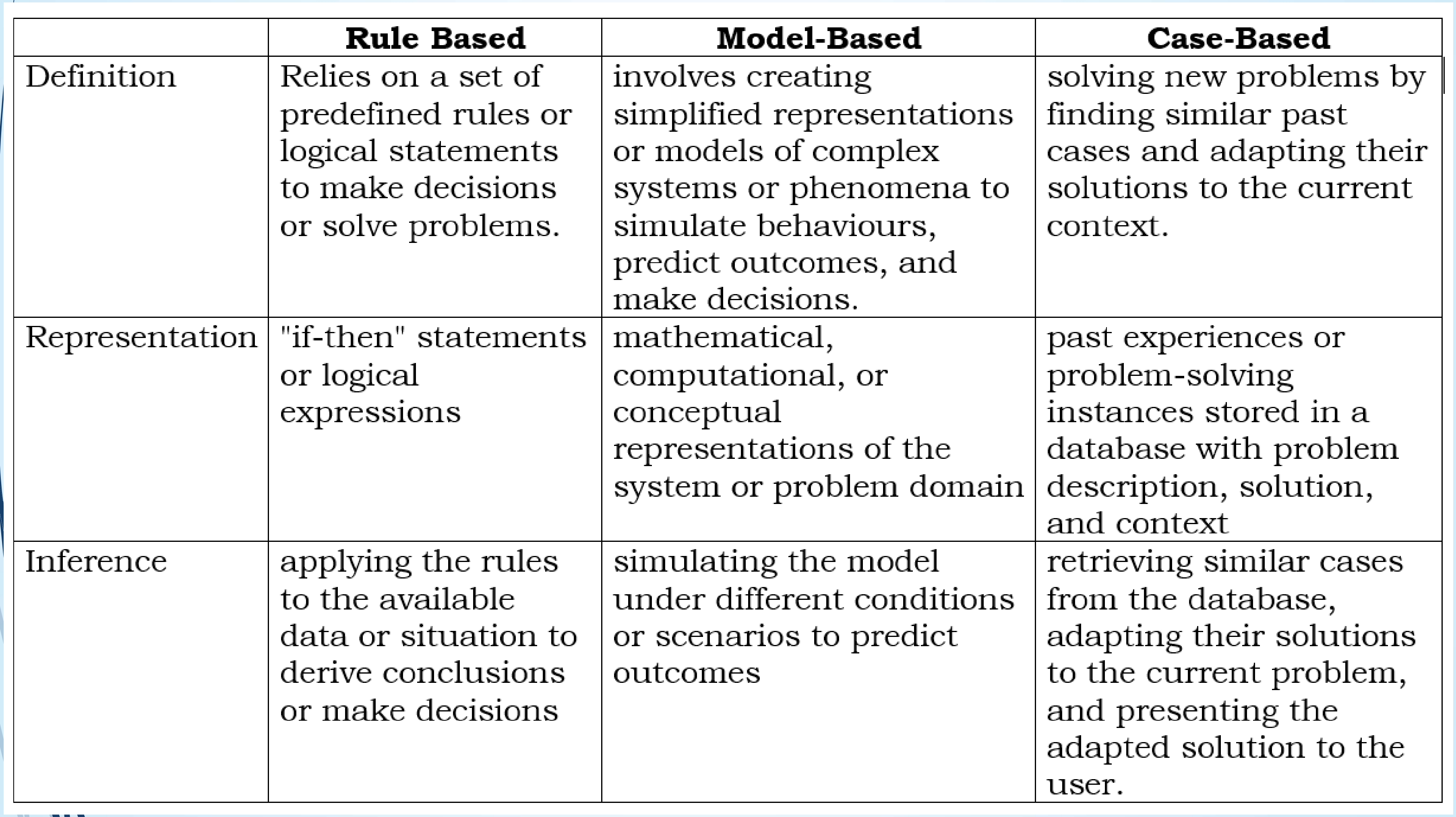


The disadvantages of case-based reasoning include:

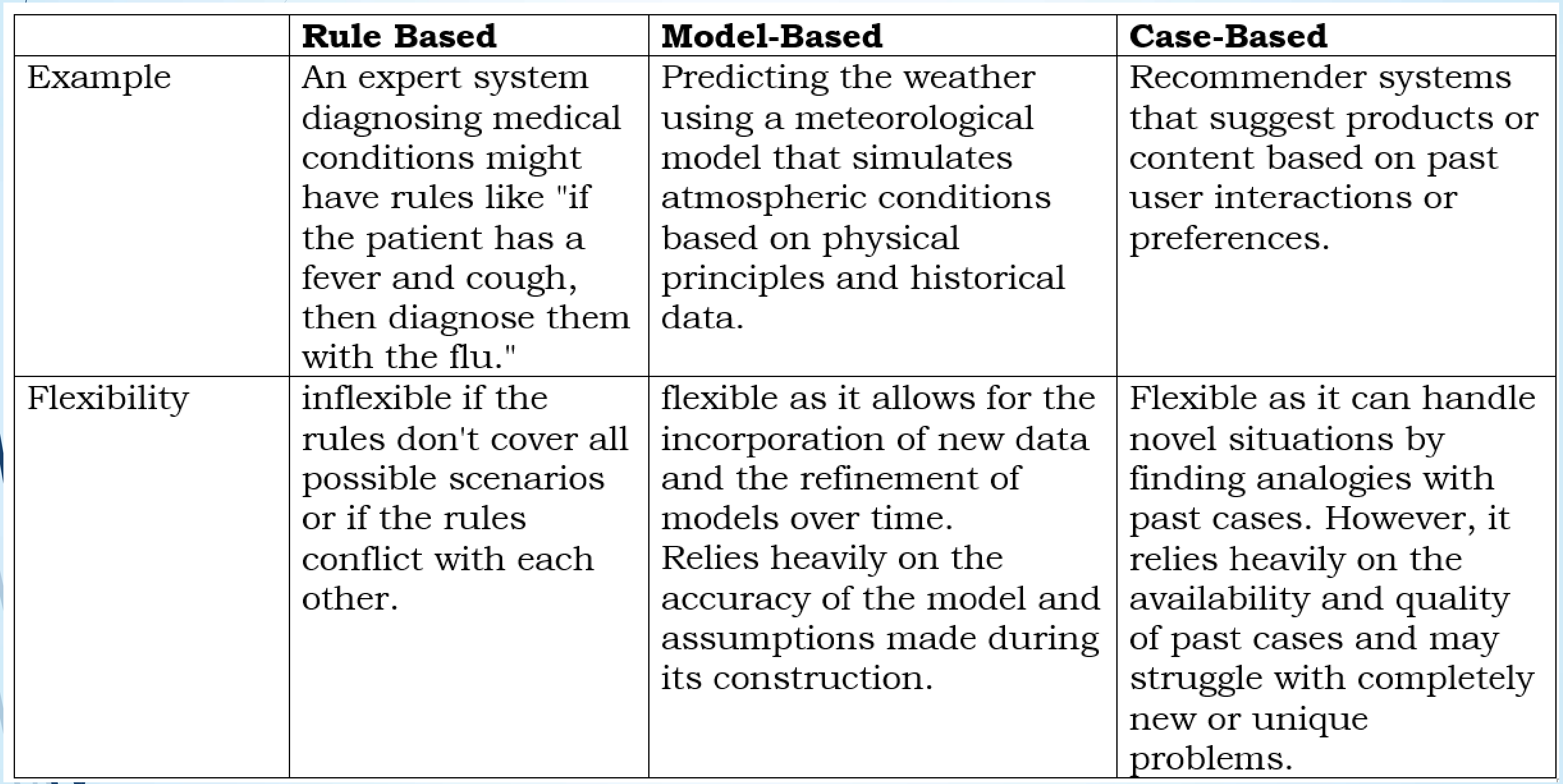
1. Cases do not often include deeper knowledge of the domain. This handicaps explanation facilities, and in many situations it allows the possibility that cases may be misapplied, leading to wrong or poor quality advice.
2. A large case base can suffer problems from store/compute trade-offs.
3. It is difficult to determine good criteria for indexing and matching cases. Currently, retrieval vocabularies and similarity matching algorithms must be carefully hand crafted; this can offset many of the advantages CBR offers for knowledge acquisition.


The disadvantages of model-based reasoning include:

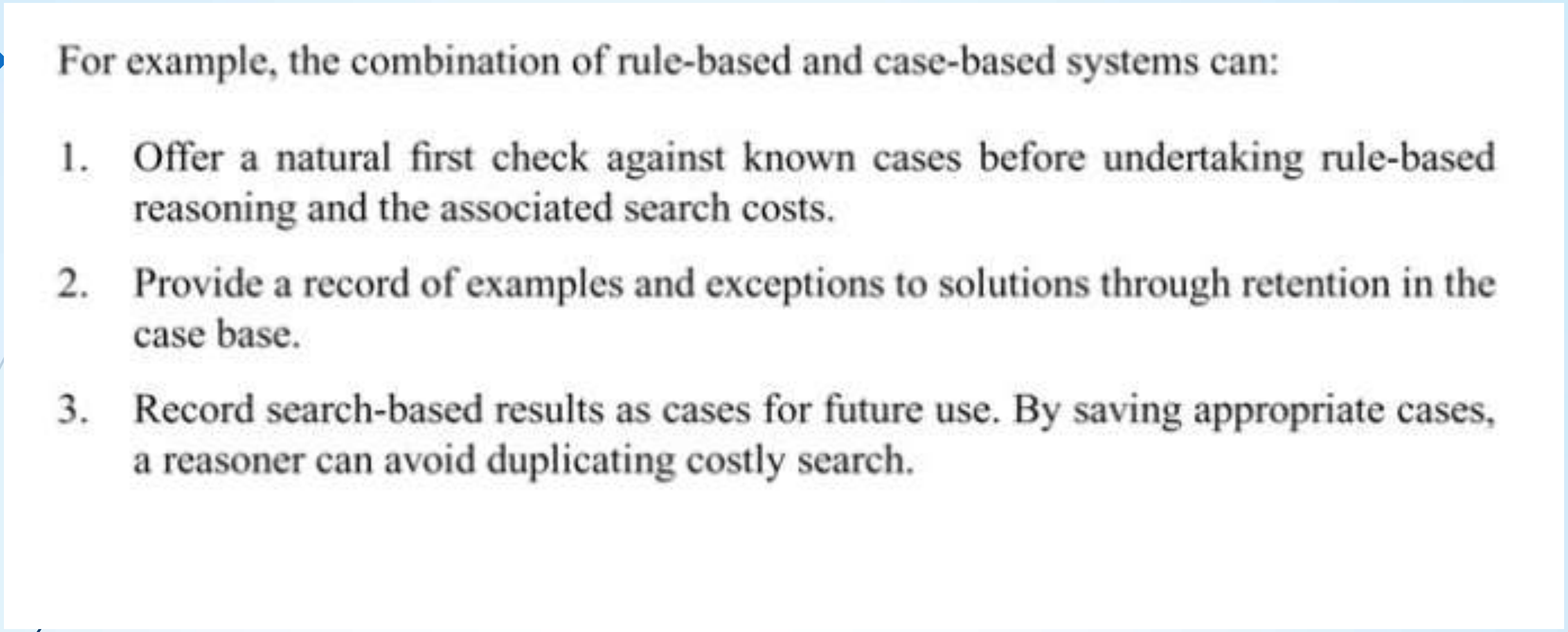
1. A lack of experiential (descriptive) knowledge of the domain. The heuristic methods used by rule-based approaches reflect a valuable class of expertise.
2. It requires an explicit domain model. Many domains, such as the diagnosis of failures in electronic circuits, have a strong scientific basis that supports model-based approaches. However, many domains, such as some medical specialties, most design problems, or many financial applications, lack a well-defined scientific theory. Model-based approaches cannot be used in such cases.
3. High complexity. Model-based reasoning generally operates at a level of detail that leads to significant complexity; this is, after all, one of the main reasons human experts develop heuristics in the first place.
4. Exceptional situations. Unusual circumstances, for example, bridging faults or the interaction of multiple failures in electronic components, can alter the functionality of a system in ways difficult to predict a priori.



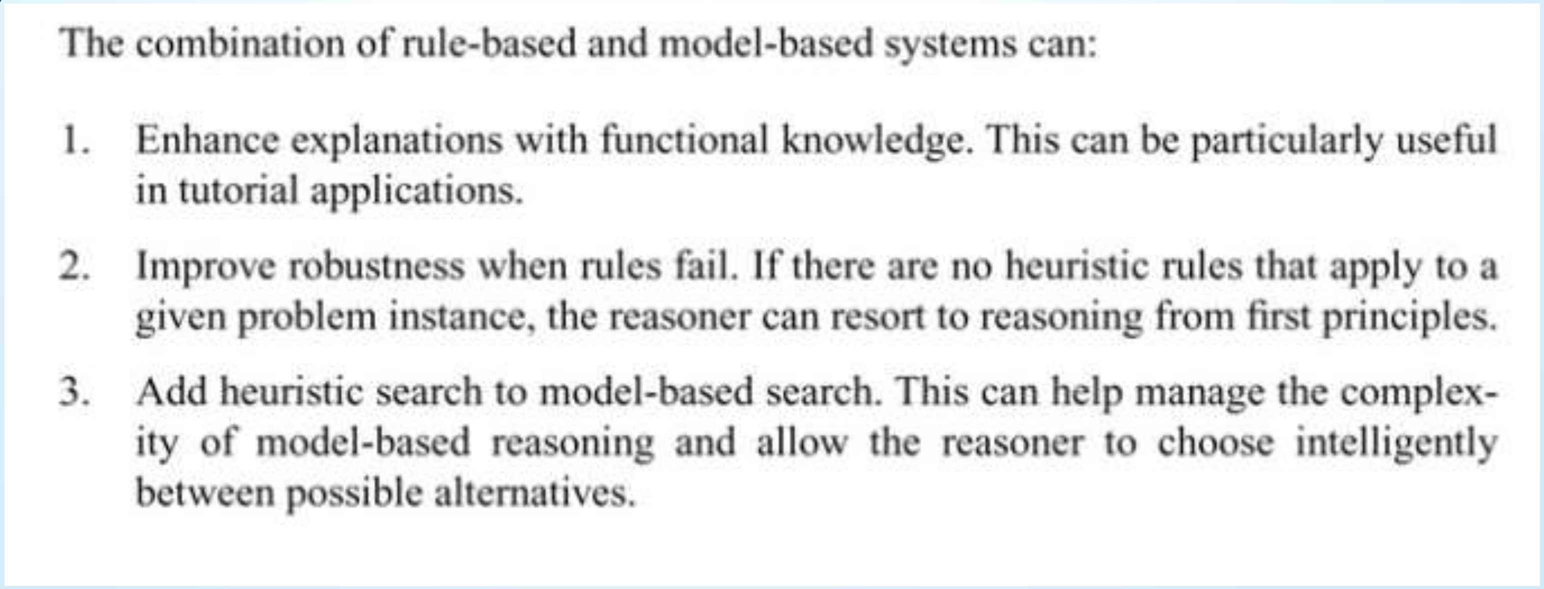
	Rule Based	Model-Based	Case-Based
Definition	Relies on a set of predefined rules or logical statements to make decisions or solve problems.	involves creating simplified representations or models of complex systems or phenomena to simulate behaviours, predict outcomes, and make decisions.	solving new problems by finding similar past cases and adapting their solutions to the current context.
Representation	"if-then" statements or logical expressions	mathematical, computational, or conceptual representations of the system or problem domain	past experiences or problem-solving instances stored in a database with problem description, solution, and context
Inference	applying the rules to the available data or situation to derive conclusions or make decisions	simulating the model under different conditions or scenarios to predict outcomes	retrieving similar cases from the database, adapting their solutions to the current problem, and presenting the adapted solution to the user.



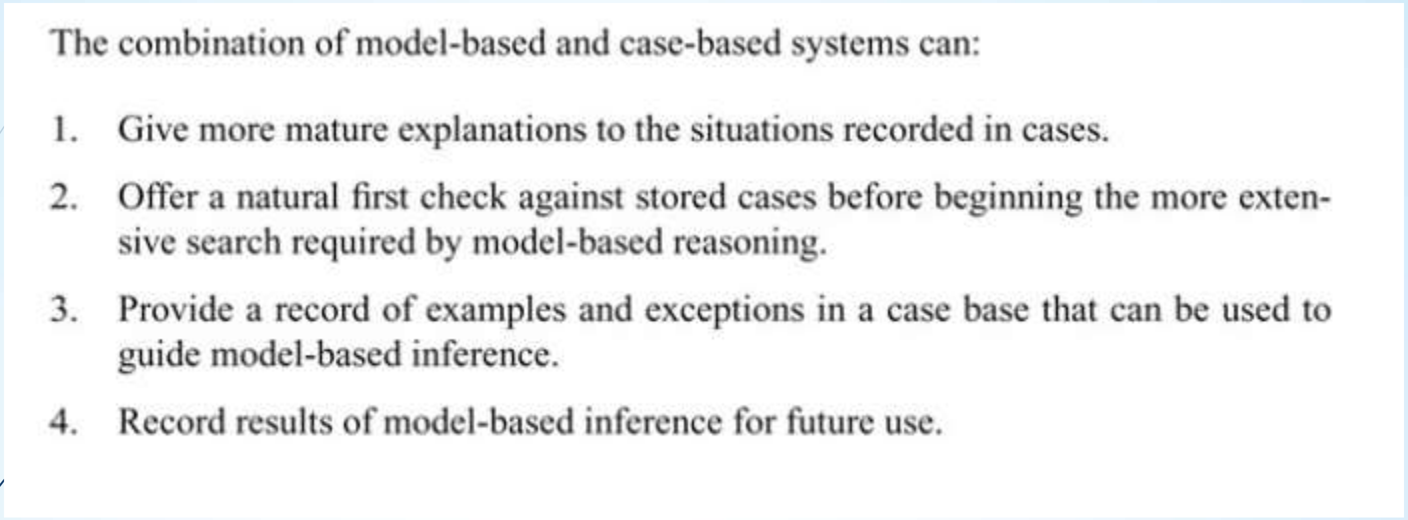
	Rule Based	Model-Based	Case-Based
Example	An expert system diagnosing medical conditions might have rules like "if the patient has a fever and cough, then diagnose them with the flu."	Predicting the weather using a meteorological model that simulates atmospheric conditions based on physical principles and historical data.	Recommender systems that suggest products or content based on past user interactions or preferences.
Flexibility	inflexible if the rules don't cover all possible scenarios or if the rules conflict with each other.	flexible as it allows for the incorporation of new data and the refinement of models over time. Relies heavily on the accuracy of the model and assumptions made during its construction.	Flexible as it can handle novel situations by finding analogies with past cases. However, it relies heavily on the availability and quality of past cases and may struggle with completely new or unique problems.


For example, the combination of rule-based and case-based systems can:

1. Offer a natural first check against known cases before undertaking rule-based reasoning and the associated search costs.
2. Provide a record of examples and exceptions to solutions through retention in the case base.
3. Record search-based results as cases for future use. By saving appropriate cases, a reasoner can avoid duplicating costly search.


The combination of rule-based and model-based systems can:

1. Enhance explanations with functional knowledge. This can be particularly useful in tutorial applications.
2. Improve robustness when rules fail. If there are no heuristic rules that apply to a given problem instance, the reasoner can resort to reasoning from first principles.
3. Add heuristic search to model-based search. This can help manage the complexity of model-based reasoning and allow the reasoner to choose intelligently between possible alternatives.


The combination of model-based and case-based systems can:

1. Give more mature explanations to the situations recorded in cases.
2. Offer a natural first check against stored cases before beginning the more extensive search required by model-based reasoning.
3. Provide a record of examples and exceptions in a case base that can be used to guide model-based inference.
4. Record results of model-based inference for future use.

Planning

- Find a sequence of actions to accomplish some specific task
- Knowledge intensive
  - Organize pieces of knowledge and partial plans into a solution procedure
- Applications
  - Robotics
  - Expert systems in reasoning about events occurring over time
  - Process control, monitoring
  - Natural language understanding where discussing plans, goals, and intentions

Robotics

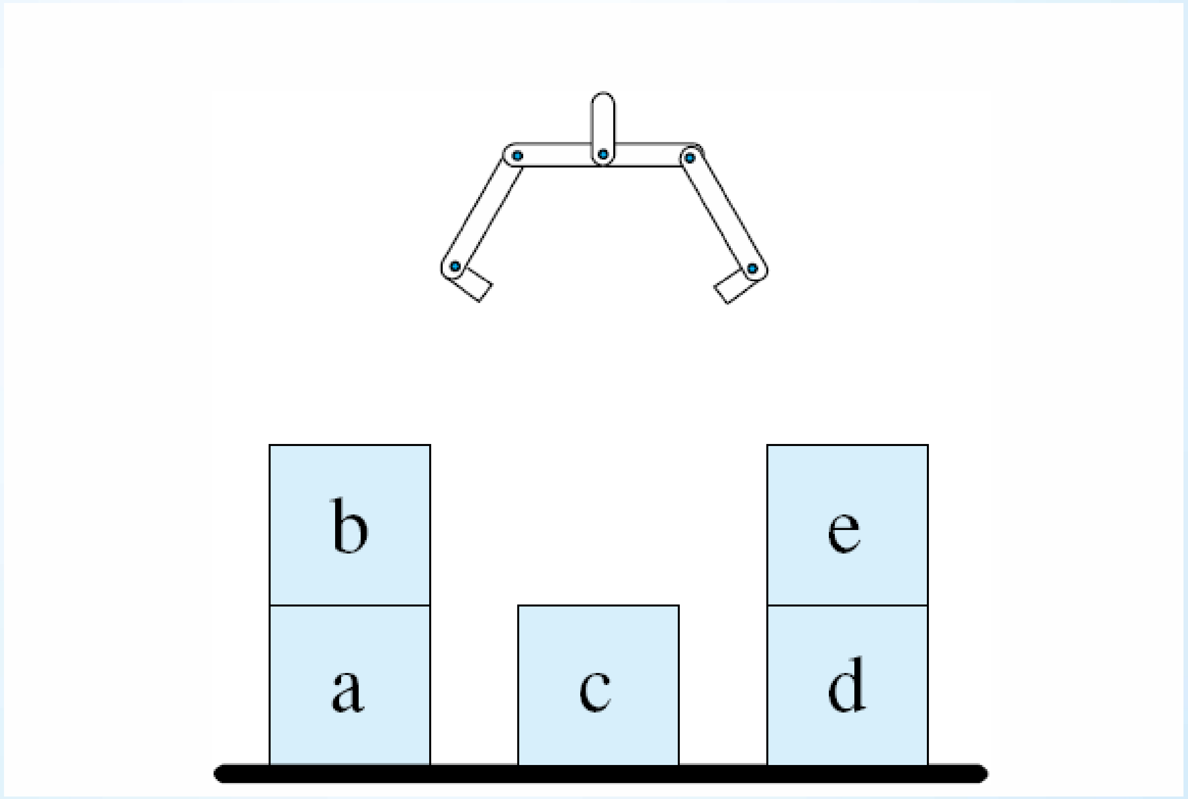
- A plan is a set of atomic actions that are domain-dependent
- Block world robot
  - atomic actions may be
    - Pick up an object a
    - Go to location x
  - Task: go get block a from room b
  - A plan:
    - Put down whatever is now held
    - Go to room b
    - Go over to block a
    - Pick up block a
    - Leave room b
    - Return to the original location
- Planning is to search through a space of possible actions to find the sequence necessary to accomplish the task

Issues of Planning

- State description of the world
  - Possible states
  - Atomic actions
  - Effect of actions on the world
- State transition
  - Which part changed
  - Which part unchanged
  - Frame problem: specification of exactly what is changed by performing an action on the world
- Generating, saving, optimizing plans
- Generalizing plan
- Recovering from unexpected plan failure
- Maintaining consistency between the world and the system internal model of the world

The Blocks World

The blocks world consists of 5 blocks, 1 table, and 1 gripper (hand).



Atomic Actions

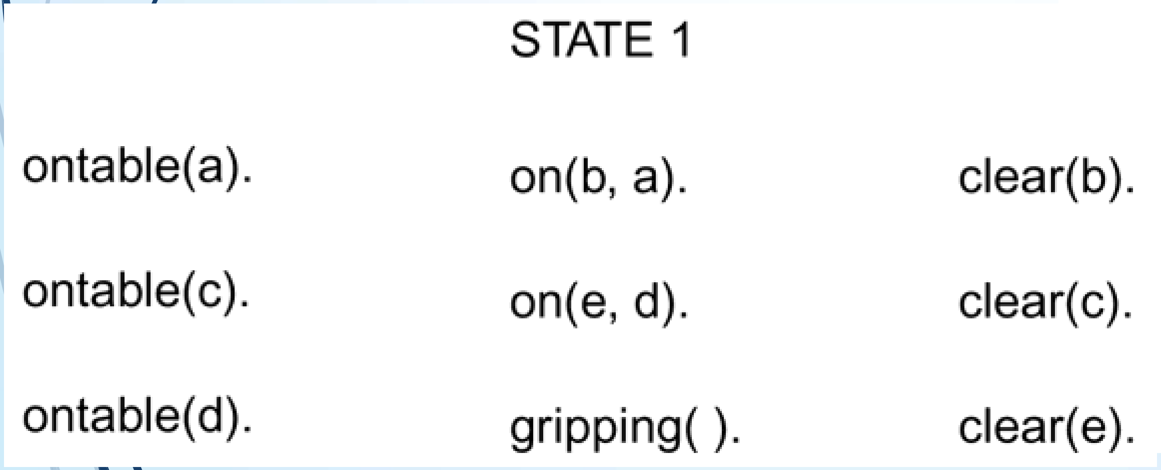
- Goto(X, Y, Z)
  - go to location (x, y, z)
  - This location might be implicit in pickup(W) where block W has location (X, Y, Z)
- Pickup(W)
  - pickup and hold block W from the current location
  - The block is clear on top, the gripper is empty, and the location of block W is known
- Putdown(W)
  - place W at the current location on the table
  - W must be held
  - Record the new location for W.
- Stack(U, V)
  - place U on top of V
  - The gripper must be holding U, and V is clear on top
- Unstack(U, V)
  - remove U from the top of V
  - U must be clear of other blocks, V must have U on top of it, and the gripper must be empty

State Representation

- A set of predicates and predicate relationships

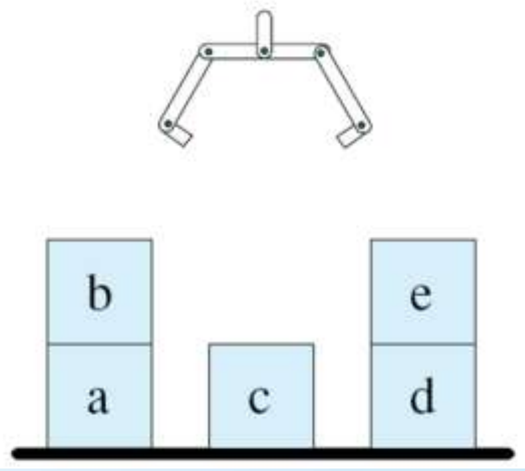
location(W, X, Y, Z):	block W is at (X, Y, Z)
on(X, Y):	block X is immediately on top of block Y
clear(X):	block X has nothing on top of it
gripping(X):	the robot arm is holding block X
gripping():	the gripper is empty
ontable(W):	block W is on the table

- Initial state

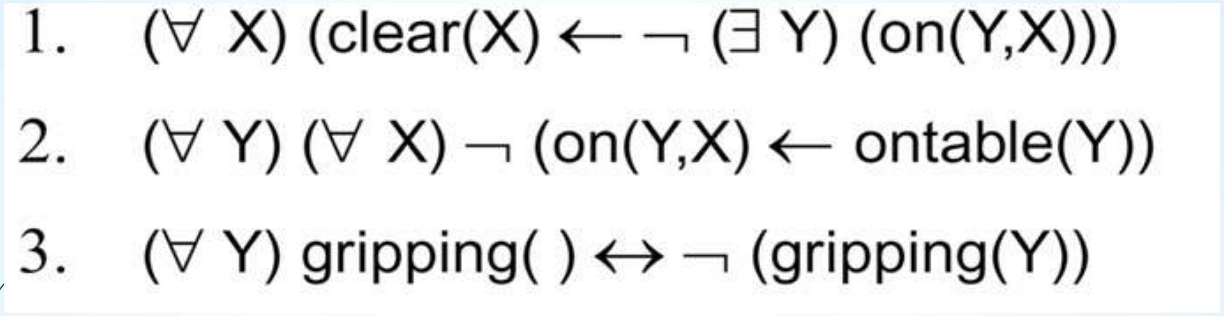


STATE 1

ontable(a).	on(b, a).	clear(b).
ontable(c).	on(e, d).	clear(c).
ontable(d).	gripping( ).	clear(e).



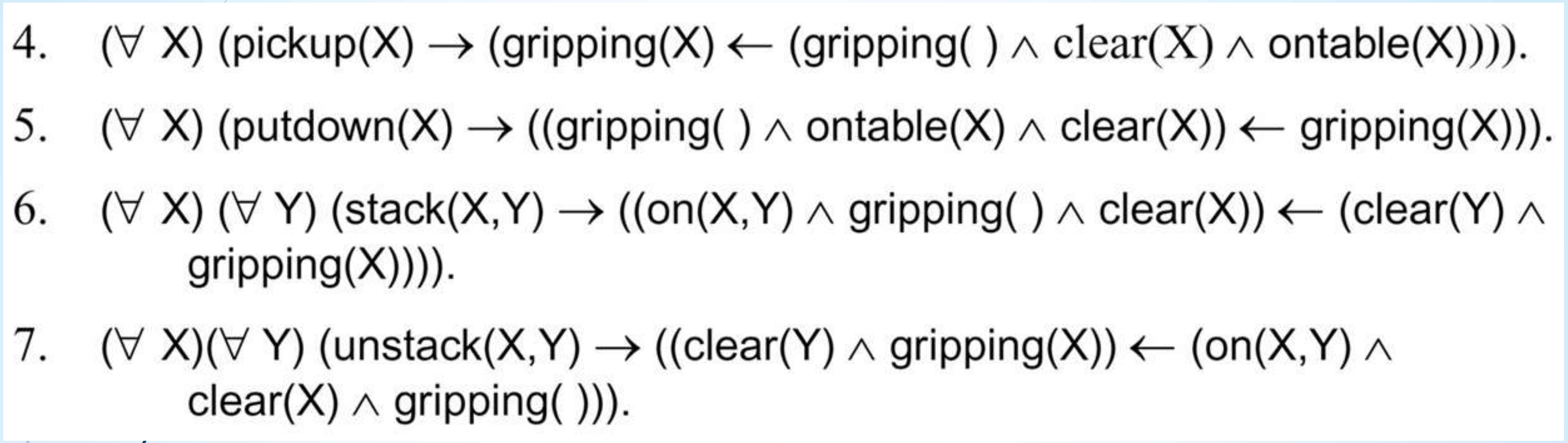
A number of truth relations or rules for performance are created for the clear (X), ontable (X), and gripping( ).



1. $(\forall X)\ (\text{clear}(X) \leftarrow \neg\ (\exists Y)\ (\text{on}(Y,X)))$
2. $(\forall Y)\ (\forall X) \neg\ (\text{on}(Y,X) \leftarrow \text{ontable}(Y))$
3. $(\forall Y)\ \text{gripping}(\ ) \leftrightarrow \neg\ (\text{gripping}(Y))$

- Rules can be interpreted either logically or procedurally. Consider the first rule.
  - Logic interpretation: If block X is clear, there does not exist any block Y such that Y is on top of X. Initial state
  - Procedural interpretation: to clear X, go and remove any state Y that might be on top of X

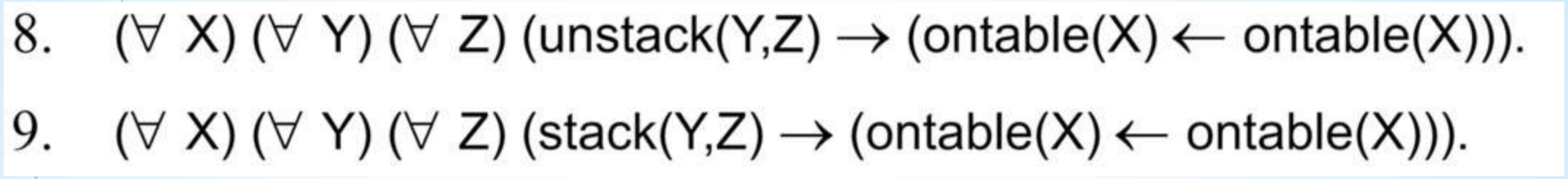
- Rules to operate on states and produce new states:



4. $(\forall X)\ (\mathrm{pickup}(X) \rightarrow (\mathrm{gripping}(X) \leftarrow (\mathrm{gripping}(\ ) \wedge \mathrm{clear}(X) \wedge \mathrm{ontable}(X))))$.
5. $(\forall X)\ (\mathrm{putdown}(X) \rightarrow ((\mathrm{gripping}(\ ) \wedge \mathrm{ontable}(X) \wedge \mathrm{clear}(X)) \leftarrow \mathrm{gripping}(X)))$.
6. $(\forall X)\ (\forall Y)\ (\mathrm{stack}(X,Y) \rightarrow ((\mathrm{on}(X,Y) \wedge \mathrm{gripping}(\ ) \wedge \mathrm{clear}(X)) \leftarrow (\mathrm{clear}(Y) \wedge \mathrm{gripping}(X))))$.
7. $(\forall X)(\forall Y)\ (\mathrm{unstack}(X,Y) \rightarrow ((\mathrm{clear}(Y) \wedge \mathrm{gripping}(X)) \leftarrow (\mathrm{on}(X,Y) \wedge \mathrm{clear}(X) \wedge \mathrm{gripping}(\ ))))$.

- A→ (B← C) means that A produces B when C is true
- Consider Rule 4:
  - For all blocks X, *pickup(X)* means *gripping(X)* if the hand is empty (gripping nothing) and X is clear.

Frame rules (axioms) to describe what predicates are not changed by rule applications and thus carried over the new states



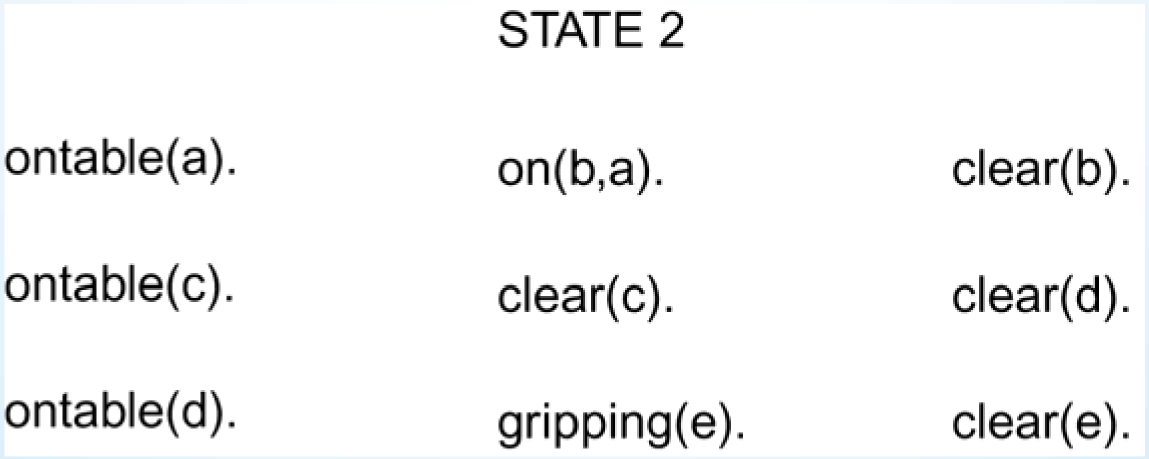
8. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{unstack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X))).$

9. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{stack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X))).$

- These two rules say:
  - *ontable* is not affected by the *stack* and *unstack* operators
- May have other frame axioms such as:
  - *on* and *clear* are affected by *stack* and *unstack* operators only when that particular *on* relation is *unstacked* or when a *clear* relation is stacked
  - Thus, *on(b, a)* is not affected by *unstacked(c, d)*

New State

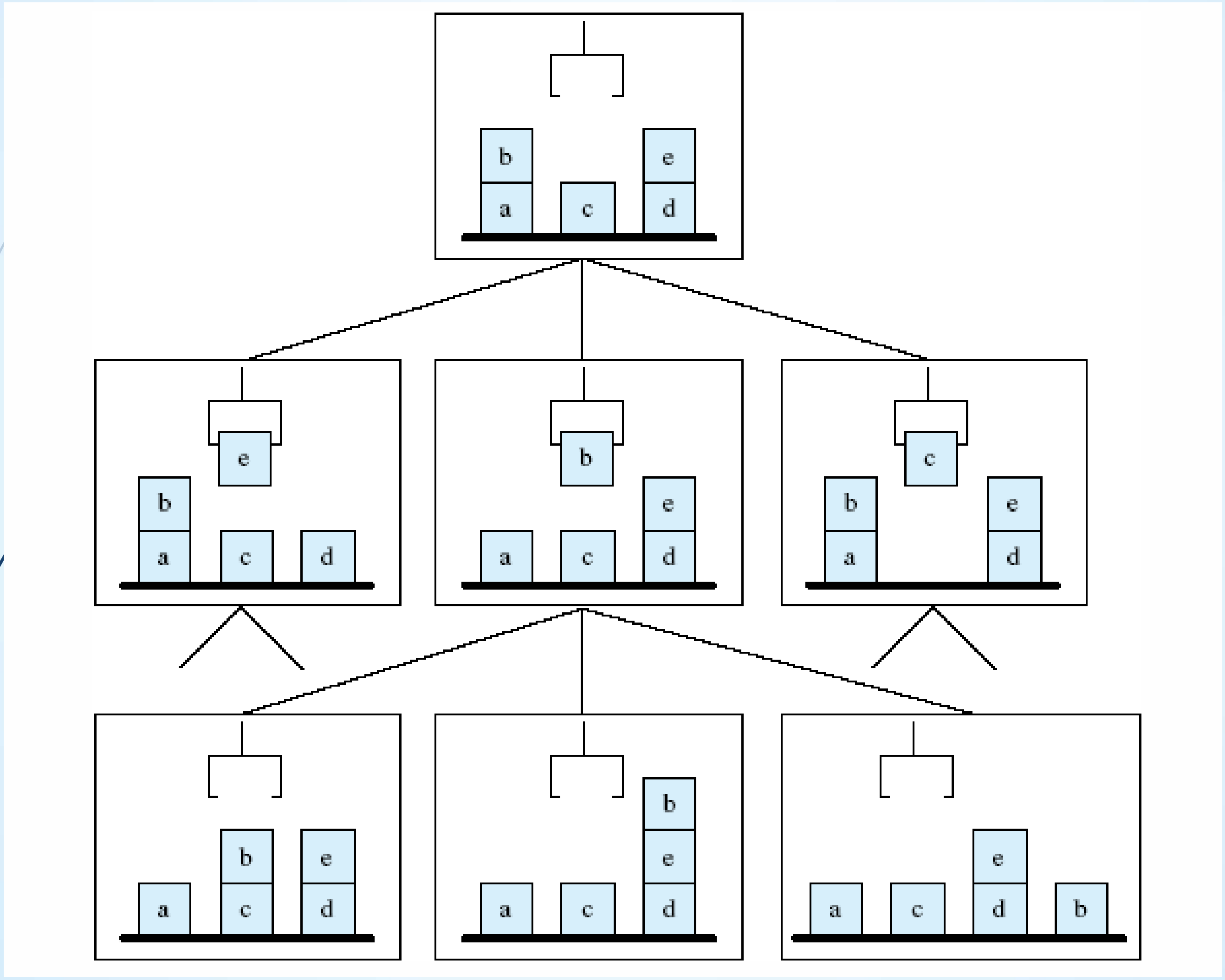
- Operators and frame axioms define a state space
- New state by applying the *unstack* operator and the frame axioms to the nine predicates of the initial state: STATE 1



STATE 2

ontable(a).	on(b,a).	clear(b).
ontable(c).	clear(c).	clear(d).
ontable(d).	gripping(e).	clear(e).

Portion of the state space for blocks world



STRIPS

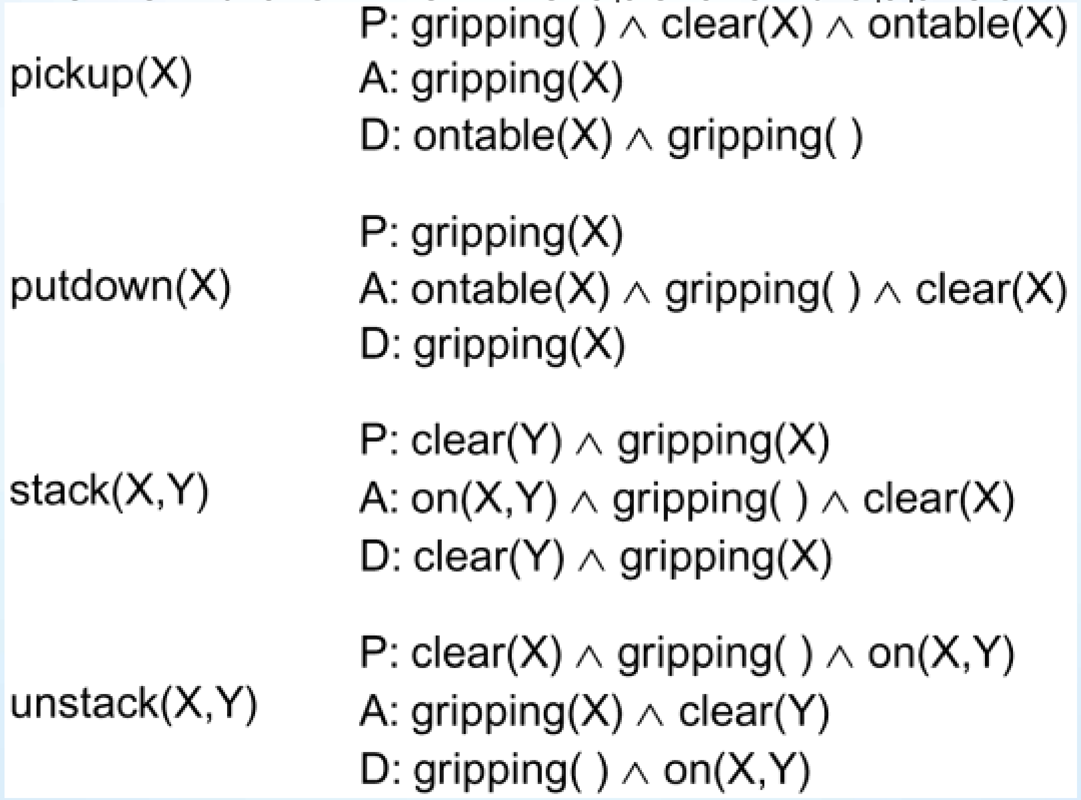
- STRIPS – Stanford Research Institute Planning System (now SRI International)
- Drove the SHAKEY robot (1970s)
- Addresses:
  - Efficiently represent and implement the operations of a planner
  - Resolve conflicting subgoals
  - Provide a learning model: save and generalize successful plans as macro operators for the future use
- Data structure: triangle tables for organizing and store macro operations

Triangle Tables

- Data structure for organizing sequence of actions, including potentially incompatible subgoals
- Relates the pre-conditions to post-conditions (combined add and delete lists)
- Used to determine when a macro operator could be used
  - A macro operator is a sequence of primitive operations that perform a subgoal
  - the problem of conflicting subgoals within macro actions by representing the global interaction of sequences of operations
- Save macro operators and reuse them in the future plane
  - Reuse macro operators to solve the problem of conflicting subgoals

STRIPS on Blocks World

- Four operators *pickup, putdown*, *stack*, and *unstack*
- Precodition-Add-Delete approach
  - P – Preconditions that must be met for the operator
  - A – Add list to add states that are the result of the operator
  - D – Delete list to delete items that are removed from a state or create the new state when the operator is applied

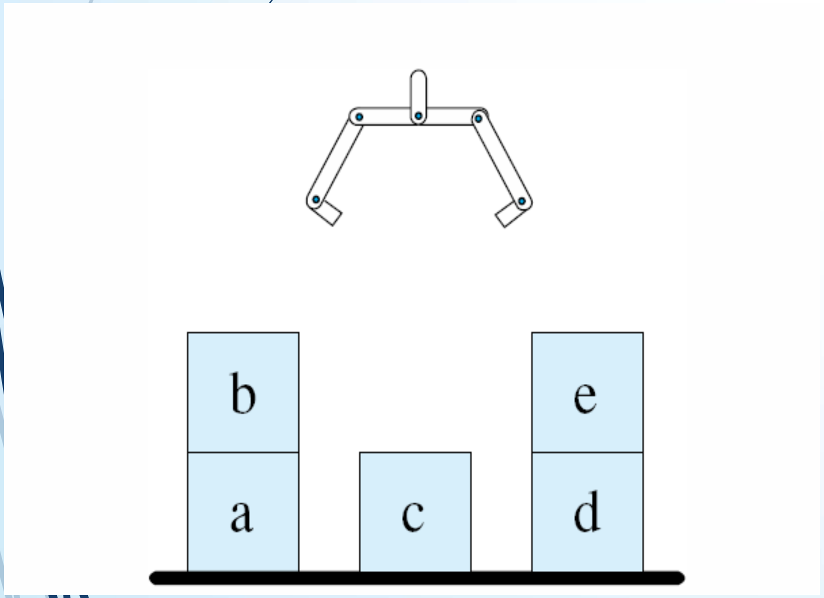


pickup(X)	P: gripping( ) $\wedge$ clear(X) $\wedge$ ontable(X) A: gripping(X) D: ontable(X) $\wedge$ gripping( )
putdown(X)	P: gripping(X) A: ontable(X) $\wedge$ gripping( ) $\wedge$ clear(X) D: gripping(X)
stack(X,Y)	P: clear(Y) $\wedge$ gripping(X) A: on(X,Y) $\wedge$ gripping( ) $\wedge$ clear(X) D: clear(Y) $\wedge$ gripping(X)
unstack(X,Y)	P: clear(X) $\wedge$ gripping( ) $\wedge$ on(X,Y) A: gripping(X) $\wedge$ clear(Y) D: gripping( ) $\wedge$ on(X,Y)

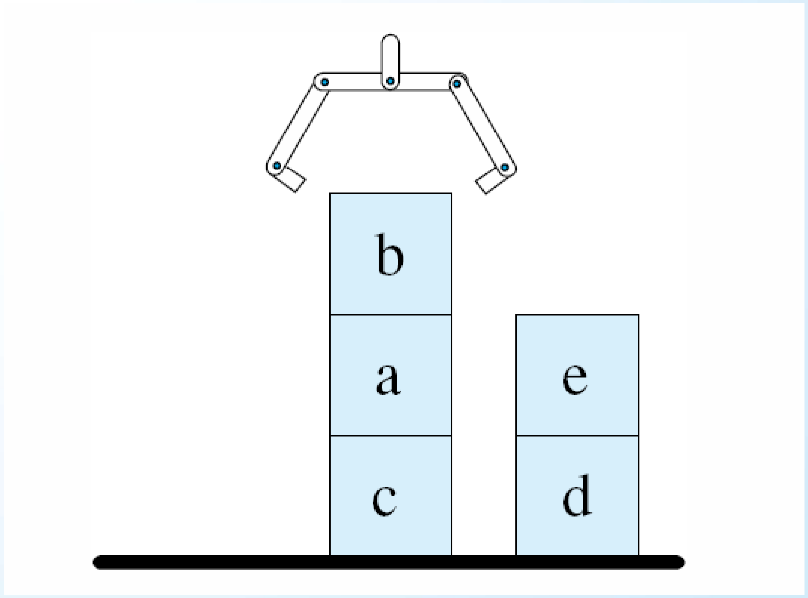
- Consider the following goal. on(b,a)∧on(a,c) is part of the goal. On(b, a) is true in both states, but must be undone.

- If the planner has developed a plan for the subgoal of the form stack(X, Y)∧stack(Y, Z), it does not need to break the goal into subgoals and avoids the complications

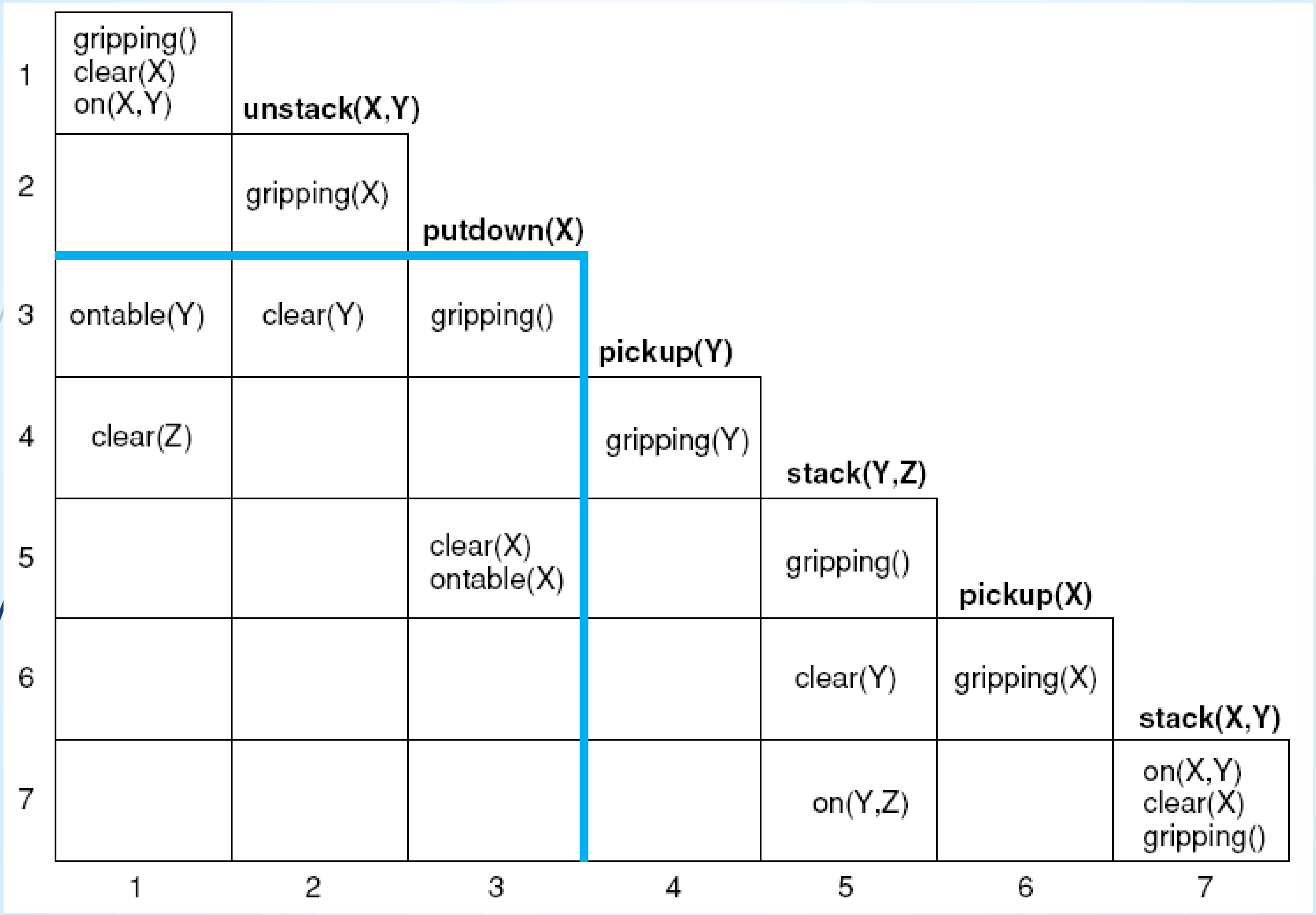
Initial state



Goal state



A triangle table of the blocks world



1	gripping()
	clear(X)
	on(X,Y)
		unstack(X,Y)					

2		gripping(X)	**putdown(X)**				
3	ontable(Y)	clear(Y)	gripping()	**pickup(Y)**			
4	clear(Z)			gripping(Y)	**stack(Y,Z)**		

5			clear(X)
			ontable(X)
					gripping()	**pickup(X)**	

6					clear(Y)	gripping(X)	**stack(X,Y)**

7					on(Y,Z)		on(X,Y)
							clear(X)
							gripping()

	1	2	3	4	5	6	7

Summarization on Planning

- Planning may be seen as a state space search
- New states are produced by general operators such as stack and unstack plus frame rules
- The techniques of graph search may be applied to find a path from the start state to the goal state. The operators on this path constitute a plan