

The Design of Spatial Information Systems Part 1: Formal Systems

Andrew U. Frank
Max J. Egenhofer
Douglas L. Hudson

©1997

This version does not contain
any references to pertinent literature.

Contents

1	Introduction	1
2	Information Systems	3
2.1	Reality	3
2.2	Systems	4
2.3	Perception and Mental Models	6
2.4	Language for Communication of Mental Models	8
2.5	Indirect Communication	9
2.6	Signs	11
2.7	Information Systems	12
2.8	Data and Information	13
2.9	Correct and Consistent Data	15
3	Formal Systems	17
3.1	Formal Languages	17
3.1.1	Backus-Naur Notation	20
3.1.2	First-Order Languages	21
3.2	Formal Theories	23
3.2.1	Truth Values	24
3.2.2	Boolean Operators	24
3.2.3	Axioms and Theorems	29
3.2.4	Examples	30
3.2.5	Clausal Forms	34

4	Models	37
4.1	Homomorphisms	38
4.2	Computers as Machines for Symbol Manipulation	40
4.3	An Information System as a Formal Model	43
4.4	Databases as Formal Systems	44

Chapter 1

Introduction

This is Part 1 of a series of texts covering the treatment of spatial information and spatial information systems, i.e., maps, land information systems, and geographic information systems. Part 1 concentrates on formal languages and formal systems. This part provides a quite generic treatment, suitable for the discussion of any complex information system that deals with a significant part of reality. Subsequent parts will be concerned with data modeling and knowledge representation for spatial data; various aspects of building software systems for spatial information systems; and a discussion of the architectural principles of designing spatial information systems.

Often enough a spatial information system is discussed as if it were only a computerized mapping system. The view taken here is considerably different and attempts to include a broader range of topics. It is based on a more comprehensive vision of how a spatial information system should perform.

Computer cartography is the subject of a number of courses and a few books have recently appeared on the topic. They discuss how maps can be drawn using a computer and show results that are achieved using typical software packages. Their focus is on the graphical process of map creation and to a lesser degree on map design; very little is said about the source and organization of knowledge about the world which is necessary to draw the map. Moreover, there is little apparent concern with the user's understanding of the map.

This series of texts on spatial information systems takes a radically different approach, trying to encompass the problem of constructing systems that will collect,

maintain, and disseminate spatial information. It will be shown that it is clearly beneficial to discuss these problems in context and to understand the interaction among their different components. Using this view, a map is a spatial information system and can be analyzed in these terms, from data collection to map usage.

This treatment strives for theoretical correctness and for the formal analysis and specification of a spatial information system. It is based on the observation that many of the problems with present day systems start with shortcuts and seemingly reasonable abbreviations, which later turn out not to be correct and which demand extensive remedial countermeasures. We start with the assumption that “a good theory is the most practical tool” and try to find the principles human cartographers intuitively apply. We try to cast them into a formal language which we can then use to program computerized information systems.

We will not discuss automatic cartography, but only computer assisted solutions; the cartographic process is not yet understood well enough to envisage a completely automatic solution. We have to be content with the assistance that specialists can receive from computer systems to increase their productivity or with the support from the computer system which lets them do things they could otherwise not do. The computer is a tool for the use of the spatial information specialist, not for his or her replacement.

The discussion about the design and organization of spatial information systems will start with what we understand by information systems. We will then extensively discuss how an information system models reality and how we can formally understand concepts like consistency and how to assess data quality. This should help us in understanding the interaction of the spatial information system with its environment, its users, and their requirements. A number of basic philosophical questions will emerge, i.e., the concept of objective information. We will not attempt new solutions to these problems, but having an awareness of them can help us avoid their most damaging consequences during the implementation of specific systems and keep us from trying the impossible.

Chapter 2

Information Systems

This chapter introduces the basic concept of an information system and as such it is a prerequisite for most subsequent discussions. It includes some philosophical questions regarding the assumption of the existence of an exterior reality. It then discusses humans' perception of reality which leads to mental models. Natural language is the means humans use to communicate their mental models and relies on the significance of signs which not only have their own physical existence, but stand for something else as well. Finally, we briefly discuss devices that can facilitate the communication process among humans and extend the reach beyond the direct, face-to-face communication.

2.1 Reality

We assume that reality exists outside of our brains and that we can observe it. This is a position that cannot be proven and not all philosophers agree with it, but it does not make sense for us to discuss an information system without assuming an exterior reality independent of the observer.

2.2 Systems

General systems theory considers a system as a delimited collection of interacting parts. The system has an outer limit and exchanges inputs and outputs with its environment. Within such a system parts interact among themselves, but also with the environment across the system's boundaries.

A system is comprised of interacting components differentiated from their environment (Figure 2.1).

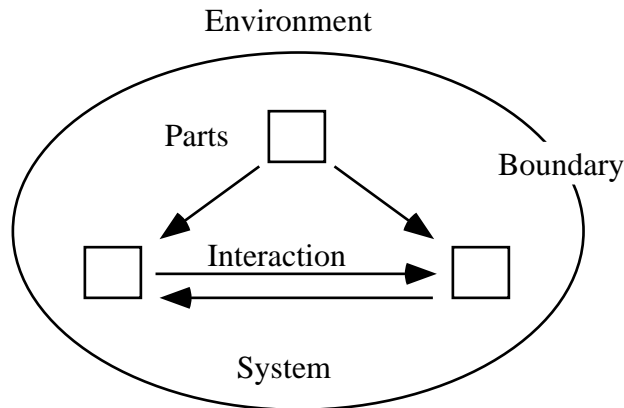


Figure 2.1: The schematic of a system.

A modeled system is a vehicle for concentrating the attention on certain, somehow related, aspects of reality and on abstractions of many other interactions as well. These systems are not part of that reality, but the result of human perception and assessment of it for specific purposes. The part of reality included in a system—the parts of interest—is often called a *mini-world*. This notion indicates that a system models some part of a world, i.e., it is in some sense complete and self-contained and considered without regards to other parts of reality, but also pointing to the fact that it is only a small, restricted part to which we draw our attention. Reality is far

too complex for comprehension and it is necessary to restrict one's attention to a few aspects in order not to be overwhelmed.

Selecting what is to be included in a system depends on the task. Generally, we want to include just enough elements and interactions to account for the effects of interest. Including too few elements makes appropriate use of the model impossible, while including too many results in an unnecessarily difficult job. An important aspect of professional education is learning what aspects are important so that they are included, and which are not. Again, this depends on the task; the significance of the color of your grandmother's hair may not be important for a discussion of a real estate deed, but may be for a investigation of descendance.

A system may exchange with its environment energy, material, information, etc. Central to the systems concept is the idea of feedback loops: the system is influenced by its environment which results in a system output that changes that environment. The system senses the new environment through its inputs and effects some further changes in system behavior.

A simple example of such a system is the heating system in a house with a furnace which heats the house and a thermostat which senses the temperature in the house and switches the furnace on or off (Figure 2.2). In this example information flows between thermostat and furnace, and energy flows between furnace and house (and also to a lesser degree to the thermostat).

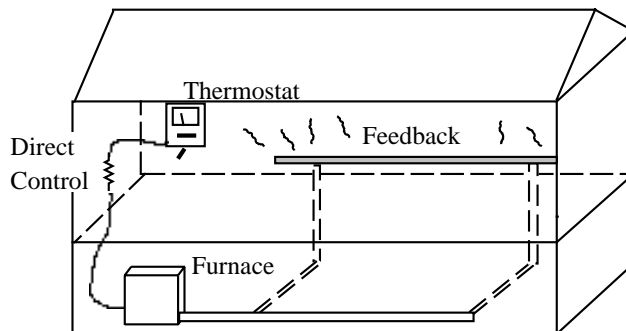


Figure 2.2: A house heating system.

Systems with feedback loops may be designed to be self-stabilizing, i.e., the heating system is designed to keep the house at a more or less constant, predetermined temperature. This can be achieved by a flow of information to cause corrective action if the room temperature is too low—“Switch furnace on”—or too high—“Switch furnace off and wait for natural heat loss through the system’s boundary.” Feedback can be destabilizing, however. Such a situation is often found in political, social or biological systems, e.g., Parkinson’s disease, but is rarely engineered.

To consider something a system, it is necessary to describe its boundary and its interaction with the environment. In a more detailed look its parts are identified and their interactions described. The interactions can be material or informational. The primary focus of this book is the latter. Systems are often analyzed in a hierarchical fashion. Starting with a coarse decomposition, as in the example above, it is possible to further decompose and study each part, e.g., the thermostat itself may be considered another system with interacting parts.

2.3 Perception and Mental Models

Humans may perceive real objects using their sensors—primarily eyes and ears. From these perceptions humans form *mental models*, cognitive structures, which represent in their brains their appreciation of the outside world. Certainly, not everyone arrives at the same model of a given reality; such a process is highly subjective and influenced by the observer’s expectations, needs, past experience, the task at hand, etc. The building of the mental model is governed by the observer’s *rules of modelization* and it is an important task during education—including professional education—to acquire typical and standardized sets of modelization rules. Such rules are often very specific for a profession and a situation.

RULES OF MODELIZATION = Rules by which mental models are extracted from perceived signals.

A surveyor and an artist looking at the same house lot will “see” very different things, since their rules for modelization have them concentrate on different aspects (Figure 2.3).

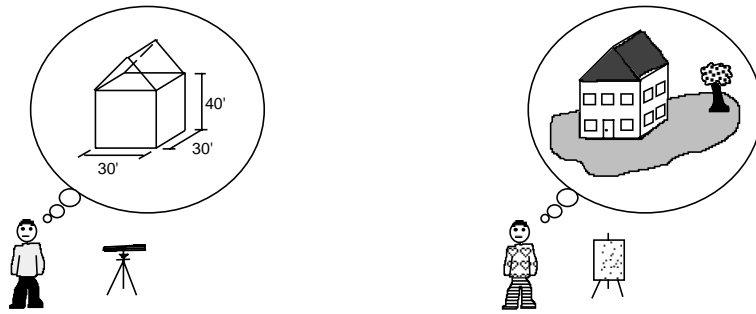


Figure 2.3: Perception is subjective.

Building a conceptual model is comparable to the building of a system: a few relevant aspects of reality are included depending on the situation and purpose (Figure 2.4). Generally, humans are not good at dealing with a lot of small details all at once; therefore, they learn to abstract and generalize and thus eliminate irrelevant detail so that they can intellectually cope with the situation at hand.

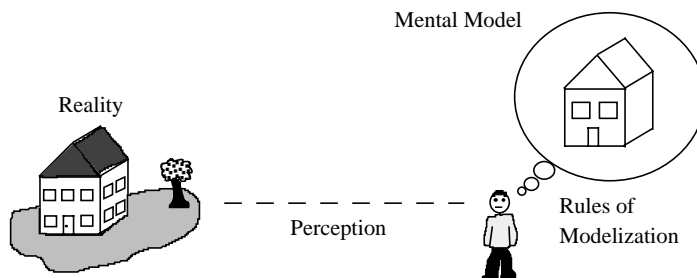


Figure 2.4: Perception and Rules of Modelization.

2.4 Language for Communication of Mental Models

Mental models cannot be communicated directly between individuals; the cognitive structures are strictly internal to the brain and no direct perception of them by another individual is possible, at least not today. In order to communicate a mental model humans use a variety of external physical phenomena—acoustic waves, graphical signs, etc.—which can be perceived by another individual. Those perceivable physical patterns are different from other observable phenomena, since they not only stand for themselves, but also convey information about something else. For example, the written word *building* is not only an interesting pattern of black and white on paper, but has some meaning attached, namely the concept of a *building* with all of its connotative baggage of possible constructions, the existence of doors, windows, etc.

All situations in which meaning is attached to physical signs will be referred to as a *language*, e.g., natural languages (English, German), computer languages (Pascal, PROLOG), and graphical languages (maps).

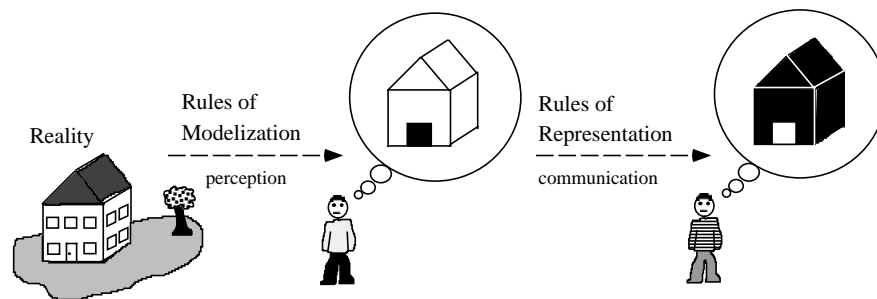


Figure 2.5: Communication.

Conventions establish the meaning of signs in a language, called *words*, and different languages can attribute different meanings to the same sign. For example, the significance of colors differs among cultures—white is, for instance, the mourning color in Japan, but not in the USA. We call those conventions *rules of representation*. Humans receiving physical representations of signs interpret them according

to their individual rules of representation and thus form a mental model (Figure 2.5). These rules are dependent on various things, like culture or profession, and we must be aware of differences in interpretation of signs in different contexts. It is hoped that this model is similar to the one in the sender's mind.

RULES OF REPRESENTATION = Rules by which meanings are assigned to or extracted from signs.

It is a well-known fact that certain professions have established their own languages which attribute meaning to certain words that are different from what would be found in an everyday situation. Most obvious is the legal and medical professions, but surveyors have also their own terms. Furthermore, it is not simply that every sign (word, token) has a specific meaning, but there are complex rules (syntax, grammar) that put one sign in relation to another and the complete idea is only expressed in the compound meaning.

The practical goal of communication, is to reproduce in the recipient the same mental model of the sender—or more ambitiously: producing a true picture of the reality originally perceived by the sender. Success depends on how well-matched the sender and recipient are in terms of:

- rules of modelization
- rules of representation

For instance, do the two parties agree on what is important to know concerning some aspect of reality, e.g., to describe a house by color or size (modelization rules), and how will these aspects be communicated, e.g., by using terms like just “blue house” instead of “sky blue siding with slate grey roof and black trim” for color or by using measurements in square feet of floor space instead of a count of rooms (representation model)? If they do agree the chances of success are high. If either party is aware that the other's rules are different, some corrective actions may be taken.

2.5 Indirect Communication

Communication between humans is most often thought of as a face-to-face situation: two persons talking with each other. This is not necessarily the case, e.g., when

using written language: potential utterances of one person can be written down and read later by another person, or even distributed and read by different persons at different times. Speech can also be recorded (on tape, for example) and transmitted to a receiver at a different time and in a different location.

Storing data in a computer system can also lead to the data being received by another person distant in time or location and thus resulting in an indirect communication (Figure 2.6).

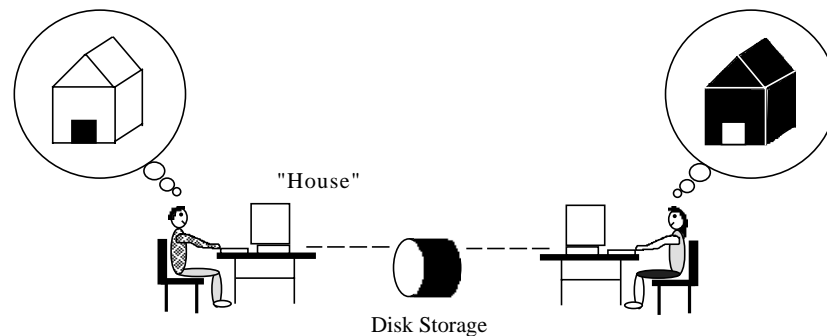


Figure 2.6: Indirect Communication.

All of what we have said about communication applies here. Success still depends on the match of modelization and representation rules between sender and receiver. If the parties are distant in space or time communication becomes more difficult, because these rules are not fixed, but are influenced by circumstances, technically called the *context* of the message. Two examples may underline the importance of context.

- Reading a novel that is hundreds of years old could be difficult, because the context may not be fully understood, e.g., different social structures existed then or the language may have changed.
- Even communication by telephone is more difficult than talking face-to-face because neither participant is as fully aware of the other's context. For instance, it should be easy to appreciate that the subtle nuances of a newly dis-

covered romantic poem would often be less than successfully transmitted if the intended recipients were standing in the rain watching their unattended car roll backwards down a hill into a busy intersection.

2.6 Signs

The physical phenomena that are used to express ideas can be treated by physical processes. Typical examples are the duplication of graphical signs in copy machines and the use of the telephone to transfer acoustic waves over distances. All those traditional means of the communication industry treat the phenomena without any regard for their meaning, without even noticing that they do not stand for themselves, but with the assumption that they mean something.

Computers can store and duplicate written or spoken text that is put into their storage systems. If that is all they are asked to do, they are not significantly different from copying machines. Many of the computer mapping systems on the market today rely on digitizing existing maps, which are subsequently output to faithfully reproduce the map input. In other instances, however, computers act “intelligently,” as if they understand the significance of data. They can solve problems in algebra and calculus, do accounting, and handle fancy word processing.

Even if computers seem to act intelligently, they really only treat the simple signs in a formal way and do not understand their meaning. Computers are not conscious about the world. They are general purpose machines and can assume, given the right program, a great many different performance roles. This may help to blur the issue that sometimes computers treat signs as if they understand their meanings and at other times they just faithfully reproduce their input. All the intermediate possibilities can be seen, too!

Consider the following examples:

- A word processing program reproduces text. Does not in any way understand what the text means. In fact, some word processors do not even understand the relatively simple concepts of “word.”
- A program like MacPaint acts like an electronic sketch pad—it manipulates black and white areas on a sheet. Compare it with MacDraw into which at least

some simple graphical concepts are built. MacDraw “understands” centering of text, etc., but does still not interpret your drawing with respect to some real world significance.

The meaning of signs is assigned by conventions which are not very stable and may vary among groups. Meanings are dependent on context, not only with other signs, but also with factual circumstances. Meaning is most often incomplete and relies on human intelligence and experience to fill in the missing details. Likewise, communication between humans is usually incomplete and only explicitly transfers the most important facts, relying on the partner in the communication to fill in the missing details, assumed to be known from previous shared or common experience, context, etc.

2.7 Information Systems

“If the operations, carried out by the elements of a system, consist of the collection, treatment, and communication of information, the system is called an *information system*.”

There are a great number of information systems, some known under this name, some with other labels such as telephone directory assistance, library, or accounting system. The above definition is very broad and the term is often used in a more restricted sense.

INFORMATION SYSTEM = A system with a primary goal to provide information about something.

Discussing an information system falls into two parts: (1) the objects about which information is collected and (2) the collection of information together with its methods for adding and retrieving information. Information systems can only describe limited aspects of the world and must exclude others, e.g., a telephone book only stores names, addresses, and phone numbers; it does not mention, for instance, whether or not the people who are listed have pets.

Concerns of Information Systems:

- *Domain*: What kind of data is contained?
- *Access method*: What kind of questions can be asked?

Information systems are only useful because they answer questions. If they only passively store data they would be pretty useless; therefore, the methods by which questions are posed and responses formulated are fundamentally important concerns in the discussion of an information system. It is generally not practical to query a phone book for the numbers of all the people in Middletown, USA, with the first name of Jack. There is no mechanism for retrieving that information. It is pointless to expect information from a system which it is not designed to deal with.

2.8 Data and Information

An information system is typically related to reality in that it supposes to inform about reality. As reality changes the information system, therefore, must be updated. Sometimes an information system is used to represent a plan for how reality should look. After the plan is executed a user must carefully check that the result matches the plan. If not, the user must modify the system such that it better represents what actually can be built.

Building an information system is a time-consuming and expensive process, however, once completed it becomes a valuable resource for the organization controlling it. To allow for the best use of this resource it should be possible that multiple users have access at the same time. Technical safeguards must prevent one user from interfering with another and to prevent some user from unauthorized access or from destroying, purposefully or not, all or part of the system. Later in the text we will discuss solutions to problems like these.

The term *information* will be reserved for contributions to the users' mental models. Only signs that can be perceived and interpreted by humans should be called information. In fact, it could be argued that information is only that which humans actually perceive and add to their mental models, and is not the raw material, i.e., *data*, from which they get this information. Information is "an answer to a human's question."

This definition excludes a number of things which are often considered information:

- A telephone directory is not, by itself, information, as humans do not ordinarily read and comprehend it. We rather use it as an information system for extracting the information we need when we want to call someone.
- Newspapers and television are not (necessarily) information, as they provide entertainment, but do usually not inform in the sense of providing answers to questions—who really wants to know that a dog with two heads was born today in Wichita, Kansas?

For information we retain the following condition: that humans (1) perceive it and (2) add it to their mental models. This includes both the question that initiated the request for the information and the perception of the answer.

The word *data*, on the other hand, will be used for symbols with a known interpretation. Most often data are in a form accessible to computer hardware, e.g., encoded and stored on media that are accessible to computers; this is not a necessary condition. In general, it can be taken for granted that a computer program can read, write, and otherwise manipulate data.

The word *document* denotes information recorded in a natural language without special formatting or encoding, but not data, because they are not formal symbols with a fixed interpretation. Typical examples are the registry of deeds, libraries, and maps. This is not information unless it is used by a human user, but it is also not data, as it cannot be interpreted within a formal model.

DATA	=	Signs in a formal language.
INFORMATION	=	Material for constructing mental models.
DOCUMENT	=	Signs in a natural language that need human interpretation.

Computers manipulate symbols according to given formal rules, called *programs*. In order to understand computerized information systems, it is necessary to gain some insight into formal systems and how they relate to reality. This leads to a concise definition of data—as contrasted to information.

2.9 Correct and Consistent Data

Data stored in an information system database generally must be *correct* to be useful. A computerized database management system cannot, by itself, guarantee factual correctness; it has no way of actually going out and checking that the grass is indeed green, that the moon is not really made of cheese, or that the house at 26 Grove Street has fourteen windows.

It should, however, include the weaker notion of *consistency*, which means the database must be free of internal contradiction. For instance, the database should never contain information that the house at 26 Grove Street has ten windows at the same time that it has information that there are fourteen. Such contradictions can easily happen if there are two storage areas for data on houses, only one of which was updated after a new room to the house was built. It can happen more insidiously if the information product is calculated or derived in some way and the result depends on some arbitrary arrangement of parameters.

CONSISTENT DATA = No contradictions with itself.

CORRECT DATA = No contradictions with reality.

Because consistency is a condition internal to a system, it can often be automatically checked by the system. Checking correctness, however, requires operations external to the information system (Figure 2.7).

Users will not employ an information system which occasionally “changes its mind.” You will place little trust in a companion who reports that he owns a Mercedes one day and a red Thunderbird the next. There are obvious exceptions to this—your companion may be a Saudi prince! If something is not correct, but is consistent, the observer does not seem to be as disturbed.

In many cases the context of the information system used may effectively replace “real” reality with its own version. This may or may not be beneficial. A social security agency’s information system, for instance, may not have a printer with foreign characters, but it can still disburse a money order using a phonetic interpretation of some ethnic names, e.g., Boris and Mohammed. In contrast, however, that agency may insist that your monthly check be stopped, because you have been reported as dead, irrespective of your personal arguments to the contrary.

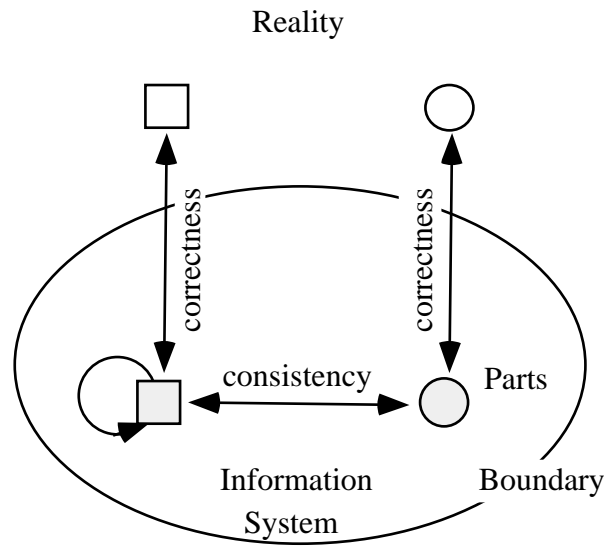


Figure 2.7: Consistency vs. Correctness.

Chapter 3

Formal Systems

Mathematics deals with formal systems, i.e., systems of symbols which are manipulated by formal rules. Mathematics does not deal with the real world! Formal systems consist of a set of symbols and rules on how these can be combined to form valid expressions in a language. The symbols do not have significance within the formal system—the symbols do not mean anything. A formal theory is such a language together with additional rules on how to attribute truth values to the expressions in the language, on when expressions are equivalent, etc.

FORMAL LANGUAGE = A set of symbols + rules for their combination.

FORMAL THEORY = A language + rules concerning valid relationships within the language.

3.1 Formal Languages

A set of symbols, together with a set of rules for their combination, form a *language*. The rule set can be called the *syntax* of the language. A symbol or valid combination of symbols constructed by some appropriate application of the rule will be called a *sentence* or a *well-formed formula*, often abbreviated by *wff*. A wff exists only if it can be shown to have been constructed from the symbols by means of the rules. Any other construction is not a wff, i.e., it is not a sentence in the language. Is not

just a “false” statement; a construction that is not well-formed cannot be considered in any meaningful way.

This chapter will explain a formal method to describe formal languages by the specification of symbols and rules (called *production rules*), e.g., we present a language to use to discuss other languages. The language to be described is called the object language, to separate it from the language we use for its description. A more extensive treatment of formal languages and their definition can be found in most texts on compiler design.

There are two kinds of symbols in the description language, *terminal* and *non-terminal* symbols. The terminal symbols are the only symbols that may appear in well-formed formulae of the language. Most often terminal symbols are characters or numbers, but they may be sequences of characters, e.g., BEGIN and END in Pascal. The non-terminal symbols are only used in rules which lead to intermediate steps in the production of a language.

The production rules describe the syntax of an object language by showing how all valid wff's of this language can be constructed. The rules can also be used to decide if a given formula is well-formed or not.

Let us use the (somewhat familiar) formal system for writing integers in decimal:

- The *terminal symbols*, i.e., the only symbols that appear in valid integers, are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, and -.
- To those we add the *non-terminal symbols*, which are intermediate symbols only used within the production rules to describe how to form a valid expression. They help us construct a wff of terminal symbols. Non-terminals are not part of the language to be described, but part of the language used to describe the object language.

The non-terminal symbols in this system are:

start, signum, digit, and string-of-digits.

The *production rules* state how a non-terminal symbol can be expanded into a sequence of terminal and/or non-terminal symbols. The idea is to use the rules to get from the start symbol to a string of terminal symbols. The production rules are:

```
start ::= signum string-of-digits
```

This rule states that the (non-terminal) `start` symbol can be expanded into the sequence of two symbols (again non-terminal): `signum` and `string-of-digits`, with `signum` appearing first. The `::=` sign is not part of the object language, but part of the language used to write the production rules.

```
signum ::= [ "+" | "-" ]
```

This rule says that the non-terminal symbol `signum` can be expanded to either nothing or a `"+"` or a `"-"` terminal symbol. The symbol `" | "` means “or” and is part of the language to write production rules, as are the brackets `" [" and "] "` which enclose parts which are optional, i.e., which may be used zero or one times.

```
string-of-digits ::= digit [ string-of-digits ]
```

This rule explains that a string of digits is a digit followed by either a `string-of-digits` or nothing. This rule can be applied repeatedly to expand `string-of-digits` to an arbitrarily length.

```
digit ::= "0" | "1" | "2" | "3" | "4" | "5" |
         "6" | "7" | "8" | "9"
```

This rule converts the non-terminal `digit` to an actual terminal symbol: `0...9`.

These rules can be applied to produce all valid (in the sense of these rules) integers. For example:

```
start  ->  signum string-of-digits
        ->  +      string-of-digits
        ->  +      digit string-of-digits
        ->  +      digit digit string-of-digits
        ->  +      1      digit digit (nothing)
        ->  +      1      3      digit
        ->  +      1      3      6
```

produces the number `+136`.

Using the previous definition, which of the following sequences are valid? 3, +0, -0, +559, 55, -000340, + ?

Saying that a Pascal program is syntactically correct is equivalent to saying that it is a well-formed formula in the formal system of the Pascal language. Pascal (and most other modern programming languages) are formally defined using production rules like those shown above.

3.1.1 Backus-Naur Notation

The notation scheme used is usually called the Backus-Naur form (or Backus-Normal form, each of which is abbreviated BNF). The Backus-Naur form is equivalent to the familiar syntax diagrams found in many computer language texts. Because BNF is a formal language itself, we can describe it using the above principles. We can even describe BNF using BNF—which is something like the famous Baron Munchhausen pulling himself out of a bog by his own hair. It is not all that difficult.

BNF uses the following terminal symbols:

- `::=` is replaced by, or, produces
- `|` or
- `[]` optional (zero or one times)
- `{ }` any number (zero, one, several times)
- `()` parentheses can be used for grouping
- `" "` quotes enclose terminal symbols

We include the interpretation of these formal symbols afterwards only to assist understanding. They are not part of the symbolism.

The production rules of BNF are:

```

syntax      ::= { statement }
statement   ::= identifier ::= expression
expression  ::= term { "|" factor }
term        ::= factor { factor }
factor      ::= identifier | "(" expression ")" | "[" expression "]" |
               "{" expression "}"
identifier  ::= string
string      ::= character { character }
character   ::= "A" | "B" | ... | "a" | "b" | ...

```

It should be easy to see that the previous description for the syntax of integers is indeed a collection of BNF production rules.

3.1.2 First-Order Languages

A first-order language is another very important formal language we will use. Its terminal symbols are:

“(”, “)”, “not”, “and”, “or”, “implies”, “=”, “for-all”, “there-exists”, and something to represent constants, variables, and predicates¹ such as “Dog”, “3”, “348”, “x”, “y”, “z”, “father”, “Blue”.

In this text, we will select identifiers starting with lower case letters to denote constants (peter, elizabeth, etc.) and starting with capital letters to denote variables (House, Person, etc.). Predicates will also start in lower case, but the difference between them and variables will be clear from the context. The parentheses play the same role as in mathematics in that they are used for grouping other symbols. Constants and variables have their usual meaning in that they identify some quantity, quality, or relation. “And”, “or”, “implies”, “=”, and “not” are special symbols called Boolean operators. Those operators that just require one operand are called *unary operators* and those that relate two operands are called *binary operators* (or connectives). Their functions are described below.

The non-terminal symbols are: `literal`, `wff`, `variable`, `constant`, `operator`, `predicate`, `term`, and `atomic formula` (usually shortened to just `atom`). The language (in BNF):

```
wff      ::= literal | (wff "or" wff) | (wff "and" wff)
          | (wff "implies" wff) | (wff "=" wff)
literal  ::= ["not"] atom
atom     ::= predicate "(" term { "," term } ")"
term     ::= constant | variable
constant ::= "a" | "b" | "123" | "fido" | ...
variable ::= "Char" | "Number" | "Dog" | ...
predicate ::= "mo" | "fa" | "ma" | ...
```

¹In general a predicate is similar to a mathematical function in that it describes some relationship among its symbol arguments. A predicate in a first-order language is similar to a Boolean function in Pascal in that it returns a logical value. A predicate with no arguments is called a “proposition.”

The language just describes the appearance of a wff. When a first-order language is extended by including operational definitions and values for its components, the result is a first-order logic (abbreviated FOL), otherwise known as the *first-order predicate calculus*. These operations will be defined in Section 3.2.

The following are some examples for first-order languages with the production rules applied from bottom up:

1. Choose a predicate ma and a constant a .

Then the construction $ma(a)$ is a wff, because:

- a literal is a wff:
wff \rightarrow literal
- an atom is a literal, therefore, an atom is a wff:
wff \rightarrow atom
- predicate (term) is an atom, therefore predicate (term) is a wff:
wff \rightarrow predicate "(" term ")"
- a constant is a term, therefore, predicate (constant) is a wff:
term \rightarrow constant
wff \rightarrow predicate "(" constant ")"
- ma is a predicate and a is a constant.

2. Choose the constants a and s and the predicate fa . Then $\text{not } fa(a, s)$ is a wff, because:

- a literal is a wff.
- $\text{not } \text{atom}$ is a literal, therefore $\text{not } \text{atom}$ is a wff.
- predicate (term, term) is an atom, therefore, NOT predicate (term, term) is a wff.
- constant is a term, therefore, NOT predicate (constant, constant) is a wff.
- a and s are constants, and fa is a predicate.

3. Choose variables X and Y and predicates fa , pa , and ma . Then $fa(X, Y)$ implies $(pa(X, Y) \text{ and } ma(X))$ is a wff, because:
- $wff \text{ implies } wff$ is a wff.
 - $wff \text{ and } wff$ is a wff, therefore, $wff \text{ implies } wff \text{ and } wff$ is a wff.
 - a literal is a wff, therefore, $literal \text{ implies } (literal \text{ and } literal)$.
 - an atom is a literal, therefore, $atom \text{ implies } (atom \text{ and } atom)$.
 - $predicate(term)$ is an atom.
 - $predicate(term, term)$ is an atom, too, therefore, $(predicate(term, term)) \text{ implies } (predicate(term, term) \text{ and } predicate(term))$.
 - a variable is a term, therefore, $(predicate(variable, variable)) \text{ implies } (predicate(variable, variable) \text{ and } predicate(variable))$.
 - fa , mo , and pa are predicates and X and Y are variables.

Although we could have chosen names like *father* in lieu of fa or *andrew* instead of a , we refrained, because we are not yet ready to discuss the application of meaning to predicate and constant names. The mnemonic content of some symbols may tend to distract us from the importance of appreciating the formal aspects of what we are trying to do here. Remember that a formal language is only a set of symbols and combination rules. It does not attach any meaning to well-formed formulae in the language.

3.2 Formal Theories

A formal theory is a mechanism whereby some rules are employed to associate an initial set of well-formed formulae with all others. If the appropriate associations can be made, the other wff's are said to be true in, or proven in, or deduced from, the theory.

3.2.1 Truth Values

In a formal logic system, atomic formulae (predicates) are assigned values (called *truth values*) depending upon what they represent, i.e., the relationship they describe. Usually the range of the values is restricted to either TRUE or FALSE, although multi-valued logics have frequently been employed, e.g., with values: TRUE, FALSE, MAYBE; or real number values ranging between 0 and 1. There are many reasons for using two-valued logics, not the least of which is that two-valued (or Boolean) logics have been historically most productive and are well-understood.

An initial assignment of truth values must be made by some agent external to the logic system; there is nothing intrinsically valuable about a particular formula. Mathematicians call an assignment of values an interpretation (in the sense of Tarski). A wff (possibly a combination of predicates) can be true or false depending on the values assigned to the individual predicates and on the operations used (see Boolean operators below). It is somewhat equivalent to say that a wff p (or $\text{not } p$) can be proven as it is to say p is true (or, respectively, false) in a particular theory. A wff is either true (provable) or not; it can either be associated with the original set using the rules or it cannot.

It is only meaningful to discuss whether or not a well-formed formula is true or can be proven; formulae that are not well-formed have no meaningful truth value nor can they be proven or disproven. The system just doesn't work for anything but well-formed formulae.

3.2.2 Boolean Operators

Some fundamental associations provided by the logic system are those concerned with the “and”, “or”, “not”, “=”, and “implies” Boolean operators mentioned above in the discussion of first-order languages. The analysis of their effect is most easily presented in truth or decision tables, which simply list the possible outcomes of the associations.

Given predicates P and Q with the following tables of values:

P	not P
true	false
false	true

The table above simply states that if P has a value true assigned, not P is false and vice versa.

P	Q	P and Q
true	true	true
true	false	false
false	true	false
false	false	false

“P and Q” is true if and only if both are true.

P	Q	P or Q
true	true	true
true	false	true
false	true	true
false	false	false

“P or Q” is false if and only if both are false.

P	Q	P = Q
true	true	true
true	false	false
false	true	false
false	false	true

“P = Q” is true if and only if both are true or both are false.

P	Q	P implies Q	Q if P
true	true	true	true
true	false	false	false
false	true	true	true
false	false	true	true

is equivalent to

The previous table defines “P implies Q” as false if and only if P is true and Q is false. P is called the antecedent, Q the consequent. If P is true the results depend on the value of Q (which seems “logical”); however, if P is false, it doesn’t matter what

Q is. The result is true. This last result may not be easy to appreciate sometimes, as it does not seem to correspond to our natural language ideas of what implies means. What it is saying, however, is something like: “If you start with a false premise, anything is possible.”

The symbol `implies` has the same meaning as `if`. You often find in the literature in lieu of `a implies b` the equivalent expression `if a then b`, or `b if a`. The last version is most often used in this text.

There are other operators defined, some of which are:

- `nand`: This is the complement of `and`. `P nand Q` is the same as `not (P and Q)`.
- `nor`: This is the complement of `or`. `P nor Q` is the same as `not (P or Q)`.
- `xor` (“exclusive or”): the complement of equivalence. `P xor Q` is the same as `not (P = Q)`.

All operators can be written in terms of the others. Minimal sets of operators can be described from which (using the proper combinations) all others can be derived, e.g., `{and, not}`, `{or, not}`, `{nor}`. Some of the equivalent combinations are extremely tedious to employ, however, and the shorter forms are usually preferred, e.g.,

- `P = Q` is the same as `(P and Q) or (not P and not Q)`
- `P implies Q` is the same as `(not P) or Q`

There are special Boolean modifiers called *quantifiers* which impose conditions on Boolean expressions:

- **FOR-ALL** (\forall): This quantifier requires that associated predicate(s) are true for all values that may possibly be assigned to a designated argument variable. For instance, $\forall x(p(x))$ means that the predicate $p(x)$ is true no matter what the value of x .

- **THERE-EXISTS (\exists):** This requires that associated predicate(s) are true for at least one value that may possibly be assigned to a designated argument variable. For instance, $\exists(x)(p(x))$ means that the predicate $p(x)$ is never always false.

A text on mathematical or philosophical logic or discrete structures should be consulted for a more substantial discussion of these topics. There are, however, some other interesting associations (rules of logic) worth mentioning here:

Given the predicates P, Q, and R and the truth values T and F (for true and false, correspondingly).

- **Idempotent laws:**

$$P \text{ and } P = P$$

$$P \text{ or } P = P$$

- **Identity laws:**

$$P \text{ and } F = F$$

$$P \text{ or } F = P$$

$$P \text{ and } T = P$$

$$P \text{ or } T = T$$

- **Complement laws:**

$$\text{not } F = T$$

$$\text{not } T = F$$

$$P \text{ and not } P = F$$

$$P \text{ or not } P = T$$

$$\text{not not } P = P$$

- **Commutative laws:**

$$P \text{ and } Q = Q \text{ and } P$$

$$P \text{ or } Q = Q \text{ or } P$$

- Associative laws:

$$P \text{ and } (Q \text{ and } R) = (P \text{ and } Q) \text{ and } R$$

$$P \text{ or } (Q \text{ or } R) = (P \text{ or } Q) \text{ or } R$$

- Distributive laws:

$$P \text{ and } (Q \text{ or } R) = (P \text{ and } Q) \text{ or } (P \text{ and } R)$$

$$P \text{ or } (Q \text{ and } R) = (P \text{ or } Q) \text{ and } (P \text{ or } R)$$

- Absorption laws

$$P \text{ and } (P \text{ or } Q) = P$$

$$P \text{ or } (P \text{ and } Q) = P$$

- DeMorgan's Rules:

$$\text{not } (P \text{ or } Q) = \text{not } P \text{ and not } Q$$

$$\text{not } (P \text{ and } Q) = \text{not } P \text{ or not } Q$$

- Modus ponens:

$$((P \text{ implies } Q) \text{ and } P) \text{ implies } Q$$

- Modus tollens:

$$((P \text{ implies } Q) \text{ and not } Q) \text{ implies not } P$$

- Modus barbara:

$$((P \text{ implies } Q) \text{ and } (Q \text{ implies } R)) \text{ implies } (P \text{ implies } R)$$

DeMorgan's rules can be used to simplify complex expressions. For instance, the following Pascal conditional is difficult to decipher, but it has an equivalent form:

```
IF NOT ((name <> "Bob") OR (count <= 72)) THEN
```

can be transformed into the more tractable expression:

```
IF (name = "Bob") AND (count > 72) THEN
```

Modus ponens is most often used for logical conclusions, such as

```
IF all humans are mortal
   AND Socrates is human
   THEN Socrates is mortal.
```

Modus tollens is used extensively in theorem proving by computer programs—more on this later.

We close this chapter with some useful terminology: Given that P implies Q :

- the *converse* is: Q implies P .
- the *inverse* is: not P implies not Q .
- the *contrapositive* is: not Q implies not P .

3.2.3 Axioms and Theorems

An *axiom* is a statement (a wff) in a theory that does not need to be proved; it is taken for granted. Any non-trivial theory must have some axiom(s) just to get started. Sometimes the rules which explain how to prove some (non-axiom) wff are called logical axioms of the theory. Of course all the logical axioms must be formal, i.e., rules that explain the manipulation of symbols independent of any attached meaning. Any other axiom is called non-logical. The non-logical axioms

of the theory which do not contain any variables are called ground axioms, ground rules, or simply *facts*.

If all the axioms are given, what good is a theory? Useful results are obtained when a proposed statement, called a *theorem*, is tested in the theory to determine its truth value. If the proposed wff can be derived from the facts using the logical axioms it is then (and only then) regarded as a true statement in that theory. It may then be added to the theory as an new axiom generating a new theory.

AXIOM	=	A fundamental statement in a theory—it needs no proof.
LOGICAL AXIOM	=	An association rule.
NON-LOGICAL AXIOM	=	All other axioms.
GROUND AXIOM	=	A non-logical axiom that contains all constant values.
THEOREM	=	A statement you wish to prove.

Typically, practical theories are built up from a number of other more basic theories, until we are not strictly aware of the formal theory/axiomatic aspects any more. In the past we have not been too attentive to this because of the amount of detail necessary to surmount for even small problems of practical interest; however, with computers being able to assist in the manipulation of symbols, formal methods become more accessible and, therefore, more important.

Very often the logical axioms in the formal theory explain when two wff's are equivalent and we can thus replace the one with the other (remember, that a minimal operator set can replace redundant constructs). This is often used, for instance, in an engineering formula where the more complex wff describes different influences on an element in a system and the equivalent simplified form is the resulting value.

If a theory is appropriately restricted we can decide the truth value for all wff's in that theory. It is an important practical problem to design a formal system with these restrictions so that, for all wff's of interest, the truth value can be decided.

3.2.4 Examples

An example of of a formal theory is the system of written numbers. The symbols are the digits 0 . . . 9 and the rules explain how to manipulate them for translation into integers for multiplication, addition, etc.

Ground axioms:

$$\begin{aligned}
 0 + 0 &= 0 \\
 1 + 0 &= 1 \\
 2 + 0 &= 2 \\
 &\vdots \\
 9 + 0 &= 9 \\
 0 + 1 &= 1 \\
 1 + 1 &= 2 \\
 &\vdots \\
 9 + 9 &= 18
 \end{aligned}$$

and logical axioms:

$$\begin{aligned}
 a * (b + c) &= a * b + a * c \\
 xyz &= x * H + y * T + z * O
 \end{aligned}$$

where $O = 1$, $T = 10 * O$, and $H = 10 * T$.

For example:

$$355 = 3 * H + 5 * T + 5 * O$$

or

$$\begin{aligned}
 23 + 15 &= (2 * T + 3 * O) + (1 * T + 5 * O) \\
 &= (2 + 1) * T + (3 + 5) * O \\
 &= 38
 \end{aligned}$$

The important thing to note is that the rules presented are absolutely formal and are based on a table for addition of 0 . . . 9. No understanding of the concept of number is necessary and there is no reality associated with, say, the symbol “3.”

Other formal systems deal with adjustments of triangulation networks, loads and stresses in statics, etc. In all these cases a system of rules, normally mathematical

equations, is given, which indicates how symbols must be related and how we can compute the quantities we are interested in from the given ones. To apply the formal methods, no understanding of the application area is necessary. This can be seen emphatically by noting that the formal system for adjustment of surveying networks and for calculating static frames is very closely related.

Finally, a simple, formal theory, not from mathematics. The ground axioms, which are all assigned a value of true:

$$\begin{aligned} & \text{fa}(a, s) \\ & \text{fa}(h, a) \\ & \text{fa}(g, h) \end{aligned}$$

We further add the logical axiom:

$$\text{fa}(X, Y) \text{ and } \text{fa}(Y, Z) \text{ implies } \text{gfa}(X, Z)$$

From these we can conclude (using only formal symbol manipulation without reference to any meaning of the symbols involved) that the following statements are true as well:

$$\begin{aligned} & \text{gfa}(h, s) \\ & \text{gfa}(g, a) \end{aligned}$$

The logical axiom we used for the above conclusions is *modus ponens*, and states:

$$((a \text{ implies } b) \text{ and } a) \text{ implies } b$$

or in English:

if b follows from a and a is true then b is true.

As applied to the above theory:

- $\text{fa}(X, Y)$ and $\text{fa}(Y, Z)$ is the a part of modus ponens.

- $\text{gfa}(X, Z)$ is the b part.

We select constants for the variables as follows:

- $h = X$,
- $a = Y$,
- $s = Z$.

Substitute them into the logical axiom to get:

- $\text{fa}(h, a)$ and $\text{fa}(a, s)$ implies $\text{gfa}(h, s)$.

From the ground axioms:

- $\text{fa}(h, a)$ is true.
- $\text{fa}(a, s)$ is true.

Using the rules for “and”:

- $(\text{fa}(h, a) \text{ and } \text{fa}(a, s))$ is true in combination.

Now we apply modus ponens on

- $(\text{fa}(h, a) \text{ and } \text{fa}(a, s))$ implies $\text{gfa}(h, s)$
- and
- $(\text{fa}(h, a) \text{ and } \text{fa}(a, s))$

and we can imply

- $\text{gfa}(h, s)$.

If we had substituted $g = X$, $h = Y$, $a = Z$, we could deduce $\text{gfa}(g, a)$. No other substitutions will work. No other wff can be deduced.

3.2.5 Clausal Forms

Since any wff can be algebraically rewritten in a different form using equivalent combinations of connectives (Boolean operators), it is often convenient to write them in a standardized form. In logic, well-formed formulae are sometimes called *clauses* and there is an extremely useful arrangement called *clausal form* where in general every clause expresses that a number (possibly zero) of joint conditions implies a number (possibly zero) of alternative conclusions.

We can rewrite any wff as an implication, the general form of which is:

$$A_1 \text{ and } A_2 \text{ and } A_3 \text{ and } \dots \text{ and } A_n \text{ implies } B_1 \text{ or } B_2 \text{ or } B_3 \dots \text{ or } B_m$$

A_i and B_j are predicates, and $n, m \geq 0$.

Since A implies B is equivalent to B if A , we often write clauses in the following alternative clausal form:

$$\begin{aligned} B_1 \text{ or } B_2 \text{ or } B_3 \text{ or } \dots \text{ or } B_m & \text{ if } A_1 \text{ and } A_2 \text{ and } A_3 \dots \text{ and } A_n \\ \text{gfa}(h, s) \text{ or gma}(h, s) & \text{ if } \text{pa}(h, X) \text{ and pa}(X, s) \end{aligned}$$

We can classify clauses by the number of predicate terms in their consequent sides as: definite (if there are zero or one term) or indefinite (if there are two or more terms). The definite clauses are generally called Horn clauses.

Horn clauses

In the case where $m = 1$, and $n = 0$, we have a definite clause that represents a *fact* or *ground axiom*.

$$\text{fa}(a, s) \text{ if}$$

Since no antecedent is required for the consequent, whatever you enter is true. Usually the empty “if” is discarded in this situation.

Definite clauses where $m = 1$ and $n > 0$ are called *rule clauses*. (They represent a form of logical axiom).

$$\begin{aligned} B_1 & \text{ if } A_1 \text{ and } A_2 \text{ and } A_3 \dots \text{ and } A_n \\ \text{gfa}(X, Z) & \text{ if } \text{fa}(X, Y) \text{ and fa}(Y, Z) \end{aligned}$$

With a definite clause which has no consequent (i.e., $m = 0, n > 0$), the antecedents are considered to be *negative facts*, i.e., facts which are known to be false.

$$\text{if } \text{fa}(a, i)$$

When both $m, n = 0$, it is the *empty clause*, which is always false by definition. It is a definite clause.

Indefinite clauses

Whenever $m > 1$ the clause has indefinite possibilities. In this case, at least one predicate is true, but we do not know which.

Using clausal forms

We will extensively use the clausal form from here on and virtually restrict our presentations to Horn clause forms corresponding to facts and rules. The use of the other clause possibilities in a database environment (the ultimate direction of this text) is either forbidden (e.g., empty clause) or imperfectly understood and still subject to intensive research (e.g., the handling of indefinite data).

Moreover, we will drop the use of quantifiers (“for-all” and “there-exists”) by insisting that all variables be universally quantified. Moreover, we will replace the appearance of “and” in the antecedent side with commas. Since there are no “or” possibilities using only Horn clauses, the only operator still employed is the “if” (some additional punctuation is added in actual programming implementations). The result of all this is a somewhat friendlier, streamlined version of first order logic.

$$\text{fa}(a, s)$$

$$\text{fa}(h, a)$$

$$\text{gfa}(X, Z) \quad \text{if} \quad \text{fa}(X, Y), \text{fa}(Y, Z)$$

Chapter 4

Models

The term *model* is a very general one, commonly used, but often without a clear understanding what it means. A model represents a system. Models are used for the study and prediction of the behavior of a system without exercising the original; they are extremely important whenever operating the system is impossible, hazardous, or very expensive. Scientists and engineers build formal models of systems in which they are interested and work with the model instead of the real thing. We do not want to build bridges with a trial and error method, nor do we want to actually test the effects of major accidents in nuclear power plants! The appropriateness of the model is determined by the sufficiency of information it provides about the system. There are many cost/benefit trade-offs to consider in choosing a model.

We can make them more complete by adding more detail; however, this may not necessarily result in more accurate predictions of the operation of the original. Often the inclusion of more detail makes the model more difficult to handle and use. It is a very hard problem to decide when a model is appropriate for a specific usage; often many technical questions must be considered. We discuss here the techniques of model building only in the abstract, leaving construction concerns of specific models for later.

Formal models have three concerns:

1. a real system to be modeled
2. a formal system with symbols and rules to be used as the model

3. a consistent mapping from properties of the real system to symbols in the formal model, often called the interpretation.

(The use of the terms “model” and “interpretation” presented here is not exactly the same as that found in logic.)

4.1 Homomorphisms

Obviously the mapping from a real system onto a formal system is what makes the model. Mathematically we can see a situation similar to a homomorphism, which essentially is a mapping that preserves structure.

$$f(x + y) = f(x) + f(y)$$

is a homomorphism if

$$f(a) = 2a$$

because

$$\begin{aligned} f(x + y) &= 2(x + y) \\ &= 2x + 2y \\ &= f(x) + f(y) \end{aligned}$$

Note that the mapping is not a homomorphism if $f(a)$ was the following function:

$$f(a) = a^2$$

A *homomorphism* is a mapping from an algebraic system (consisting of objects and operations on them) to another algebraic system, such that mapping the result of an operation on objects in the domain onto the range or mapping the object onto the range and then performing the (mapped) operation on them produces the same outcome.

In (Figure 4.1) the function $g()$ is such a mapping.

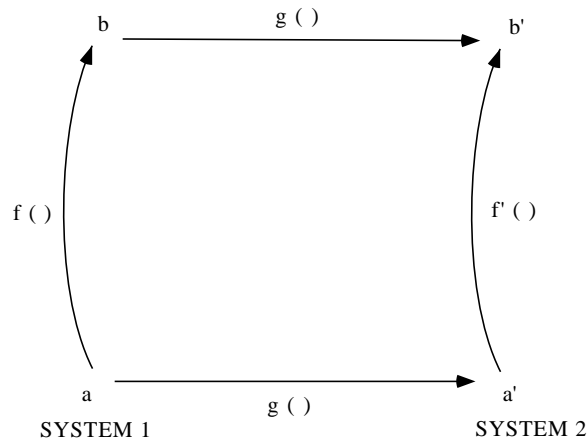


Figure 4.1: Homomorphism.

In (Figure 4.2), the real house was painted green.

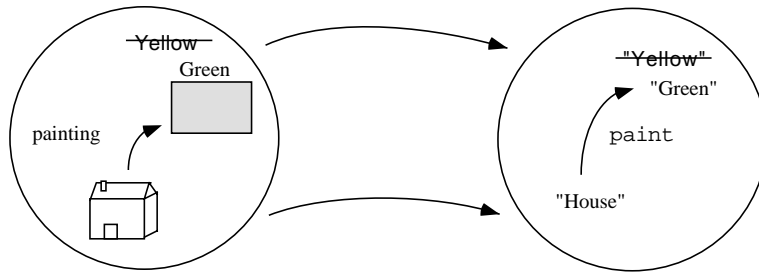


Figure 4.2: Two ways of “observing” the house color.

The model can reflect this change in two ways (i.e., there are two paths to get from real house to model HouseColor). One way is by observing that an operation on the real house (painting) changed the value of real house color and that the new

real house color can be mapped directly onto the model's color; the other way is to map the real house onto the model house and apply the model's operation which is equivalent to painting (perhaps: `FUNCTION paint ("green")`). Either way the model house's color is now green.

This is the idea in a model: in lieu of performing our experiment we perform the corresponding operation on the model and interpret the result in terms of the real system.

Now we can see how theories can be used. By referring to a previous example, we map:

1. the predicate symbol $f a$ onto the relation *father of*
2. the predicate symbol $g f a$ onto the relation *grandfather of*
3. the constants s, a, h, g onto the individuals *stella, andrew, henri, and george*, respectively.

It may not be obvious, but this is a valid model of a part of a particular family, the Frank family. That henri is the grandfather of stella is something we can deduce from the model or discover by direct observation of the Franks.

Recall that symbols in a formal system have no meaning in themselves; they only interpreted when the formal system is used as a model for some real system.

4.2 Computers as Machines for Symbol Manipulation

Computers manipulate symbols, and all their operations can be understood as symbol manipulation. Even if human users often tend to see it differently (e.g., as a numerical computation, or as a more complex operation like booking an airline passage), the internal operation of a computer is never anything more than a manipulation of symbols according to formal rules laid down in programs.

Computers represent symbols internally in bit patterns. Hardware and software operations are built in to manipulate those patterns in a way consistent with our understanding of arithmetic or logical operations. Other operations can as easily be seen as symbol manipulation and can be formally described. For example, there exists a logical formal description of the meaning of Pascal programs.

Computer programs are thus considered as incorporating special theories that then operate, together with the rules embodied in the program, to deduce logical conclusions from given inputs. A payroll program, for example, contains the equivalent of an axiom stating that the total payroll is the sum of all individual paychecks. This may not be the prevalent view of how computers and programs operate, but it is beneficial as it makes a time proven system of formal reasoning available.

It should be no surprise, therefore, that it is possible to write a computer program that operates like a first-order logical theory (with some restrictions). Such a program can be considered as already containing some fundamental logical axioms and it can be operated with application domain specific axioms provided by the user. Such programs are completely general purpose and can be adapted to be useful in any domain, in ways similar to general purpose programming languages like COBOL, FORTRAN or Pascal. PROLOG is such a system.

It is important to realize that it is this ability of a general purpose computer system to model symbolically formal systems that is the source of its power and influence on modern life. All finitely computable functions, including non-mathematical ones such as airline reservations, chess games, and speech synthesizing, can (theoretically) be performed. Not all get performed for practical reasons like time or financial constraints, but that's a resource problem.

This power is available in nearly all modern programming languages. The differences between programming languages is in the ease by which particular kinds of computations can be arranged. For instance, it is often very difficult, but not impossible, to write procedures (easily constructed in Pascal using recursion) which are equivalent to recursion in Fortran (which has no built in recursion facilities).

The Fibonacci number sequence is best defined as a recursive process ($fib(n)$ is the value of the n th number in the sequence):

$$\begin{aligned} fib(0) &= 1 \\ fib(1) &= 1 \\ fib(n) &= fib(n-1) + fib(n-2) \end{aligned}$$

A Pascal Fibonacci function is simple and elegant:

```

FUNCTION fib (n: integer): integer;
BEGIN
  IF n < 0
    THEN fib := 0 { Error flag for negative input }
    ELSE IF n <= 1
      THEN fib := 1
      ELSE fib := fib (n-1) + fib (n-2);
END

```

The Fortran version is not as elegant:

```

      INTEGER FUNCTION FIB (N)
      INTEGER N, I, BACK1, BACK2, SUM
      IF (N .LT. 0) THEN
C      Error flag for negative input
        FIB = 0
      ELSEIF (N .LE. 1) THEN
        FIB = 1
      ELSE
        BACK1 = 1
        BACK2 = 0
        DO 10 I = 2,N
          SUM = BACK1 + BACK2
          BACK2 = BACK1
          BACK1 = SUM
10      CONTINUE
        FIB = SUM
      ENDIF
      RETURN
      END

```

The quest is to find computer languages that will allow us to explain our problem (i.e., define it formally) in a “natural” way. FORTRAN is often used effectively for

numerical calculations, because it was designed for that; languages like PROLOG, which are based on logic, are more suitable for problem domains such as expert systems.

4.3 An Information System as a Formal Model

A very abstracted view of an information system is as a formal system together with an interpretation that links the formal symbols in it to reality. A computerized information system, therefore, can be seen logically as a formal model of (a part of) reality. The formal system (executed by the computer) operates on symbols which have a specific interpretation in the model (perceived by people).

Users of information systems assume implicitly that they gain the same information (i.e., the same mental models) by consulting the information system as they would by going out and gathering the information themselves through direct perception of reality.

- You assume that the telephone number you receive from directory assistance is the same you would obtain by going to a person's home and reading it from their phone.
- The tax assessors consulting their lists of parcels and frontages assume that the figures are the same as if they went out and measured for themselves.

This is similar to the homomorphisms we discussed previously, linking reality and symbols in a model together. An information system is thus a model of (a certain part) of reality. Information systems that do not conform to this implied assumption of a homomorphism are useless: what can you do with an information system that informs you about people's telephone numbers, but the numbers given are (more often than not) not the ones at which the persons can be reached? The general apprehension of quality in an information system is closely related to this assumed homomorphism.

Often symbols in a computerized form are called "data" and in logic terminology these data can be considered as ground axioms within the theory given by the program which deals with them. Moreover, within the formal system we can differentiate a collection of axioms (rules and facts) from the methods used for deducing

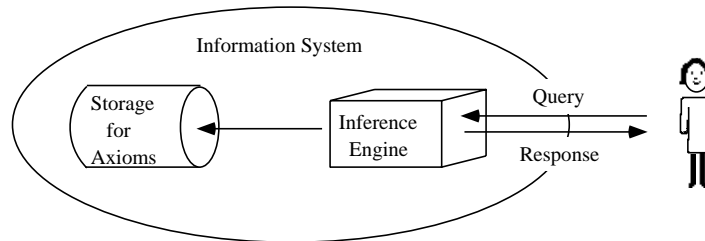


Figure 4.3: Computerized Information System

new theorems. This latter “active” part is often called the inference engine because it infers new results from old ones.

A user can query such a system by proposing a theorem and asking that it be proven (Figure 4.3). If the theorem follows from the stored axioms then it is considered a true statement; if it does not, it is a false one.

It is possible that the inference engine will not be able to prove the theorem one way or the other. Unrestricted logic systems are undecidable; theorems always exist that lead to logical paradoxes in which the results are neither true nor false. This is not much of a problem in simple systems that only contain facts and in which the inference engine merely searches through them for the answers. In more powerful systems where the inference engine applies some rules the appropriate restrictions are critical. We will discuss some of these restrictions later; for now we will just assume that they are in place.

4.4 Databases as Formal Systems

In common terminology, a database is a mechanism established for the storage and retrieval of data (long-term storage is implied). To be technically correct, a database is only a repository for data; access to it is provided by database management systems (DBMSs). A DBMS may or may not be tightly integrated, highly sophisticated, or very efficient. When the access method is tightly integrated with the storage facility, people will often use (misuse) the term database system (or worse, just

database) to refer to the combination of the database and its management system. We will continue the abuse in this text.

If symbols can be viewed as data, the DBMS that accesses them can be viewed as a formal system for their manipulation.

We will now introduce a database system, called a proof-theoretic database, which accesses data using formal logical operations. The database is considered a theory; data are the axioms, and queries are theorems to be proved by the database. The database produces answers that are considered true statements in that database.

The data stored are, at least conceptually, in the form of Horn clauses. The inference engine (the DBMS) employs a technique called *resolution* in which the various terms of the theorem to be proved are matched against the consequent sides of the resident axioms. If every term of the theorem is successfully matched the theorem is proved, otherwise it fails.

Matching a term of a theorem is a fairly straightforward procedure if the axiom is a ground axiom (a fact). Remember that facts have only consequences; they have no antecedents; therefore, all that is necessary is that the various components of the term and those of the consequent of the fact appropriately correspond.

The query (i.e., theorem) is:

$$\text{fa}(a, s)$$

The database contents (i.e., theory axioms) are:

$$\text{fa}(a, s)$$

$$\text{mo}(i, s)$$

$$\text{fa}(h, a)$$

$$\text{fa}(g, h)$$

In this example a simple sequential search will find that the 3rd axiom matches the query term. The database will report that $\text{fa}(a, s)$ is a true statement. If the query had been $\text{fa}(m, s)$, no match would have been found and the report would have been that the query statement was false.

Matching a term of a theorem with a logical axiom (a rule) not only requires the appropriate correspondence between that term and the rule's consequent, but, in addition, all the terms of the rule's antecedent must then also find matches in the

database. If the rule has multiple terms in its antecedent, each one must be successfully matched. If one or more of them connects further with other rule axioms the search process can get expensive.

The query (i.e., theorem) is:

$$pa(a, s)$$

Here the database contents (i.e., theory axioms) are more complex; now there are rule axioms:

$$\begin{aligned} mo(i, s) \\ fa(a, s) \\ pa(i, s) \quad \text{if} \quad mo(i, s) \\ pa(a, s) \quad \text{if} \quad fa(a, s) \end{aligned}$$

A sequential search will find a match for the query in the consequent of the 4th axiom, but now a match must also be found for the 4th axiom's antecedent term: $fa(a, s)$. This search (which is quite independent from the first one) finds a match for that at axiom 2. Thus, the original query: $pa(a, s)$ is *true*.

It is easy to see that these searches could be cyclic and turn into infinite processes.

The query, again, is:

$$pa(a, s)$$

If the database were:

$$\begin{aligned} pa(a, s) \quad \text{if} \quad fa(a, s) \\ fa(a, s) \quad \text{if} \quad pa(a, s) \end{aligned}$$

The query would find the 2nd axiom, the antecedent of which would then find the 1st axiom; however, the antecedent of the 1st axiom is the same as the original query and the cycle runs forever!

Infinite recursion can often be very difficult to spot and prevent when the relationships which are being modeled are complex. Mechanical checks on cyclic

activity can be performed to cancel infinite searches, but they can get quite expensive in long search processes.

If a theorem contains variables, the reporting mechanism is slightly different. Instead of a straightforward match and a report of success, the search process will report the value of each term in the database that would make that theorem true.

The query is:

$$\text{pa}(a, X)$$

The database contents are:

$$\begin{aligned} &\text{fa}(a, s) \\ &\text{fa}(a, as) \end{aligned}$$

This query would result in the database reporting: $\text{fa}(a, s)$ and $\text{fa}(a, as)$, because if s or as had been substituted for X the theorem would have been true. Failure is reported when no possible matches are found.

The power of the proof-theoretic database is demonstrated in the following:

The query is ("Who is the grandfather of stella?"):

$$\text{gfa}(X, s)$$

The database contents:

$$\begin{aligned} &\text{fa}(a, s) \\ &\text{fa}(h, a) \\ &\text{fa}(g, h) \\ &\text{gfa}(X, Y) \quad \text{if} \quad \text{fa}(X, XY), \text{fa}(XY, Y) \end{aligned}$$

The database will be searched for $\text{gfa}(X, s)$ and a partial match will be found in the 4th axiom. It is a rule axiom, so the antecedent terms must now also be matched. The variable Y end test in the argument list of the gfa axiom is associated with the value of the constant s (from the original query) in a process called binding. The resolution process will now be to find appropriate values for the X variable by matching up the antecedent terms.

Matching the antecedent terms proceeds from left to right, so first $\text{fa}(X, XY)$ must be matched. Since all the arguments of this term are variables any axiom with a fa predicate will match with it; the first found (searching sequentially from top to bottom) is $\text{fa}(a, s)$. If we use it, the X variable is bound with a and the XY with s . The second term must also be matched, but by now all the variables have been bound. This means that we must try to find a match for a second term with the values: $\text{fa}(s, s)$, which fails.

The search will continue. Going back to the first term of the antecedent of the gfa rule axiom, we find another axiom which fits. The 2nd axiom matches as well as the 1st, but with the 2nd axiom the variables X and XY are bound to different values (h and a , respectively). The second term is again completely bound to values: $\text{fa}(a, s)$; however, this time a search for a match will succeed. This means that there exists some set of values which will make a true statement in this database/theory:

$$\begin{aligned} X &= h, \\ XY &= a, \text{ and} \\ Y &= s. \end{aligned}$$

The original query only contained a variable in the first argument. It is only this value that gets reported. The initial result of this query would, therefore, be:

$$X = s$$

The database could continue searching for other matches in the same way from the point at which it reported its last success. Some implementations automatically produce all possible results; others will prompt the user for search instructions after each success.

In these last few chapters we have shown how processing a query in a database can be understood as theorem proving within a formal theory. This theoretical view of database operations is very general as well as very powerful. It also leads to a better understanding of how information can be processed.