A knowledge engineering logic for smarter, safer and faster (expert) systems

Hajek, J.

Published: 01/01/1988

Publisher's Version
Published: 01/01/1988

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 10. Dec. 2018
A knowledge engineering logic for smarter, safer and faster (expert) systems.

Jan Hajek

Eindhoven, February 1988
ABSTRACT: A neoclassic logic was defined for and tested in a new expert system QuiXpert (in Pascal). New logical operators were designed so as to allow semi-intuitive thinking and yet to provide extra semantic self-checking and speed by avoiding asking of superfluous (hence silly) questions and evaluations. Our logic is an extension of Boolean logic, rather an evolution by upgrading than a hard to swallow revolution. Therefore it is easy to understand, to implement and to work with in any programming language and in systems other than expert systems. Potential users are data bases, retrieval, management information systems, decision support systems, etc. The power of our logic was tested on realistic examples shown here. The syntax (trivial), semantics, and pragmatics are extensively covered. Hence the report has over 50 pages.
Those long chains of reasoning, so simple and easy, which enabled geometricians to reach the most difficult demonstrations, had made me wonder whether all things knowable to men might not fall into a similar logical sequence.

( Rene Descartes )

Herein too may be felt the powerlessness of mere logic, the insufficiency of the profoundest knowledge of the laws of the understanding, to resolve these problems which lie nearer to our hearts, as progressive years strip away from our life the illusions of its golden dawn.

( George Boole )

Somebody has classified people into three categories: into the uneducated, who see only disorder; into the half-educated, who see and follow the rules; and into the educated, who see and appreciate the exceptions. The computer clearly belongs to the category of half-educated.

( Heinz Zemanek, IBM Vienna, IFIP President 1971-74 )

It is not reasonable to develop a new logic, but to be reasonable is for the simple-minded.

Je ne cherche pas, je trouve. (= I don't search, I find.)

( Pablo Picasso )

( QuiXpert )

An expert is a man who has stopped thinking - he knows.

( Frank Lloyd Wright )

I know you believe you know what I know, but I don't know whether you know what I don't know.

( A private thought of every expert system )

Murfstadter's law: Whatever may go wrong it will, even if you take into account Murfstadter's law.

( Murphy's law recursively Hofstadterized by Hajek )

Give me a fruitful error any time, full of seeds, bursting with its own corrections.

( Vilfredo Pareto )
ABREVIATIONS AND AUXILIARIES:

"ES" stands for "expert system".
"lhs" means "left hand side"; lhs is a single variable or ES's rule.
"rhs" means "right hand side": rhs is a logical variable or expression.
":=" means "lhs is a logical function of rhs".
"rule" stands for a logical assignment statement lhs := rhs which abbreviates lhs := LogicalFunction(A, B, ... U, W).
"hypothesis" means the goal-rule which a user wants to evaluate.
"#" stands for a number.
"iff" means "if and only if".
"=" stands for "the same", "equal", "equivalent".
"<>" stands for "not the same", "not equal", or "not equivalent".
"set" = "class". Here we use both to denote any collection of items, objects, entities, attributes, regardless of whether they are uniquely (re)present(ed) in that collection or not. Hence we do not need to distinguish between a "set" (with no duplicates) and a "bag" (duplicates allowed), for our purposes.
"complementary" are disjoint (= nonoverlapping) sets such that their union forms a whole i.e. exhaustively covers all possibilities intended to be covered by the lhs := rhs rule in the given context.
"graph" means a bunch of nodes (alias vertices) connected by arcs (alias edges i.e. links).
"multitree" means a directed loopless graph with one or more roots.
"root rule" means a rule (in a multitree of rules) whose lhs does not appear on the rhs of any other rule.
"leaf" means an elementary rule which cannot be inferred from other rules (Ask-, COMPUTe-rules below, but not Infer-rules).
"arboricide" means "cutting of the (sub)trees". In this report arboricide is largely due to conditional evaluations. This tree-cutting activity is approved even by hard-line ego- & eco-logicians.

Hint: Recall that any (part of an) algebraic formula may be visualized as (a part of) a tree. Any logical (sub)expression may be viewed as a decision (sub)tree. Feel free to think about ES' rules as it suits to you. Either in linear algebraic way, or "graph"-ically. A bunch of (dis)connected logical expressions may be viewed as a decision multitree.
WHY AN EXTENDED LOGIC?

Beginnings of logic go back at least to Aristoteles (382-322), and to Dignaga, Dharmakirti and Nagarjuna (cca 600 A.D.) - Indian Buddhists. The good old Boolean logic is only 140 years old. It has worked well since 1938 when Claude Shannon has realized that it is suitable for modeling of switching circuits. Note that it took the genius of Shannon to realize what nowadays even junior programmers consider as obvious. Some ten years earlier two Polish logicians Lukasiewicz and Tarski have done theoretical work on multiple-valued logic, but nothing practical. After some 90 + 50 years after its (re)discovery, there is an urgent need for a less black-or-white logic and for new logical operators. The need springs from the ever increasing importance of nonnumeric and highly interactive applications. We cannot afford the luxury of asking superfluous hence silly questions from people and from huge data bases. We also must get more grip on the semantics (= meaning). We must be able to express and let the machine check certain elementary semantic dependencies among i.e. constraints on logical variables or expressions. My extensions of Boolean logic are practical and easy enough for all programmers who feel need for a logic less crude than the yes-or-no, black-or-white, take-it-or-leave-it Boolean logic. Hence this report aims at a much wider audience than just at the expert systems experts.

Our new logical operators have proved themselves so successful in avoiding asking of superfluous questions, in avoiding unnecessary evaluations and in catching certain semantic errors, that I would like to have them available in programming languages, even for a two-valued logic, because they lead to sharper (expert) systems.

WHY EXPERT SYSTEMS?

Much of the computing is a costly GIGO = Garbage In, Garbage Out. Why? Because of:
- Too big demand for versus short supply of educated personnel facing:
- too dynamic nature of the computer business, science and information.
- Ever changing requirements (a specific dynamism), plus:
- the promise of software to be "easy to change", despite of:
- the high costs of software development and maintanence. Modifications are often not done by the originator, hence introduce new errors.

Expert systems promise:
- To help us to cope with information explosion & knowledge inflation which creates and is created by the increasingly chaotic complexity.
- To deliver automated "reasoning" (inferences) and questioning from:
- a knowledge supplied to a computer in "chunks", semiautonomous parts. Of course disconnected chunks are of little use. The synergistic effects and emergent properties (for the lucky dragons) are solely due to interactions of many (implicitly) related/connected chunks.
- Inference engines (= knowledge interpreters) which make sense out of all those chunks of knowledge, regardless of their ordering, i.e. without need for any (explicit) flow-control command-structure.
- To allow non-computer-specialists to put knowledge into computers without all that hustle with stacking, hashing, bit-fiddling.
Applications of expert systems:
- identifications of all kinds, especially:
- diagnoses of machines, processes, plants, animals, and (wo)men:
- automated inferencing (questioning plus deductions):
- automated decision making, i.e. advising on actions to be taken in administration, business, management, military or other enterprises.

It is my hands-in-and-on experience that a good expert system:
- Provides an attractive alternative for certain kinds of the increasingly important programming tasks mentioned above.
- May provide solutions (both in terms of means & ends i.e. programming methods/techniques/style & results) superior to those obtained by classical, traditional programming.
- Trades off a "hard-to-change but 'easy' to follow" (because explicit) flow of control in a classical program, for an "easy to change but hard to follow" implicit flow of control in an expert system. Indeed, if all the flow of control is implicit (done behind the screens in that little magic black-box called inference engine) then only little work needs to be done when a chunk of knowledge must be inserted, deleted or changed. But the consequences of a change are much harder to follow than in a classical program. Unless the designer of an expert system (ES) has taken special care to design his ES with SAFETY FIRST in his mind, as I did it with QuiXpert, and unless good tools for analysis of rules are available. My KNOWLEDGEXPLORER finds certain inconsistencies and redundancies in knowledge-rules, but primarily it is a knowledge acquisition tool. KNOWLEDGEXPLORER extracts rules from examples, hence it is an inductive "learning" tool. Knowledge acquisition and knowledge representation are the major obstacles in filling and exploiting all those wonderful but empty ES-shells. Indeed an ES-shell without knowledge is like a record player without records, or as a computer without any software to run. But the difficulties with making knowledge explicit and so formalized that it becomes machine processable are not specific to ESs. On the contrary, ESs should make the process easier and better.

WHY QUIXPERT ?
- No amount of reading, talking, toying and tooling around with black boxes can replace a hands-in-and-on design-it-yourself experience. Unless you have done it, you can hardly judge commercial products in such a fast changing, edge-cutting domain. Only if you have done it from scratch, from design philosophy down to the last semicolon, only then you may learn what are the pros & cons, what you want, what "they" do (not) have, what the troubles & dangers are. Without doing it, you will be like a virgin discussing sexual life.
- I do not like poorly designed GIGO black-boxes full of bugs, delivered in machine code or in slow, unstable, nonstandardized, unportable languages. To convert from PASCAL into, say, C is much simpler than from PROLOG or LISP or v.v. The same holds for interfacing.
- It is flexible, modular, LEGO-like, and in PASCAL. Therefore it is also ideal for tailoring of dedicated, special versions. Also for
embedding into other "normal" programs, 99% of which will never be written in PROLOG or LISP.
- It offers a choice of several kinds (="levels") of logic.
- It offers a choice of new logical operators designed with their user, the knowledge engineer, in mind. (S)he can think semi-intuitively, concentrate on the task proper, without becoming a "logic freak".
- It is quick. QuiXpert doesn't search for rules, it knows which to use when. Its design has been inspired by Picasso's statement above. No joke. It will run at top speed in PASCAL on your favorite PC. It is fast enough for embedded real-time ES-applications, and if necessary, I could make it beat even faster.
- It is strongly self-checking. QuiXpert checks "everything": itself, the knowledge-rules, and consistency of user's answers.
- It decreases number of questions asked. QuiXpert has an omnidirectional conditional ("shadow") evaluation. It has also an internal "think what to ask" phase, followed by an external "ask iff needed" phase. Both phases do not necessarily alternate. They are activated as reasonable. E.g. it makes no sense to run an internal phase after a fresh restart, unless some facts have been loaded. On the other hand it makes no sense to activate an external phase after a restart with reuse of the earlier (sub)conclusions, if the newly required hypothesis is one of the (sub)conclusions from the previous round.
- It has carefully designed (self)debugging, errors reporting, testing and printing facilities. All simple but effective.
- It is relatively simple, but not simplistic ( re: KISS design rule ).
- It is an expert system without tears: with less undetected bugs, without PROLOG, without LISP, even without recursion. PROLOG cannot derive negative conclusions, supports neither conditional evaluations nor any plausibilistic logic, is totally unsuitable for numeric computations, provides little syntactic and runtime checking, and is hopelessly slow. LISP is somewhat better in some respects, but provides no typechecking. PASCAL is a strongly typed language which reduces GIGO = garbage in, garbage out computing to minimum, and it runs at top speed on any PC.
- It is implemented so that PASCAL compiler (and not some obscure tool) delivers all the following goodies for free:
  + typing errors (cause undeclared identifiers);
  + type checking (of variables and functions);
  + missing rule-functions ("semantic" incompleteness via syntax);
  + order of rules;
  + looped rules (hence "forward" declarations are undesirable), thus (in)directly mutually recursive rules are prevented;
  + split rules prevented by the compiler-enforced unique naming: X := F and H; ... X := J and K is disallowed, we must use X := (F and H) or (J and K) which prevents fragmentation & scattering of knowledge, gives good eye-checks.
  It is a piquant detail that Horn clauses in PROLOG use split rules.
  + compiler checks all the remaining syntax ( no work, no worries );
  + speed: very fast, compiled "semantic net" generated, no scanning and pattern matching of rules needed. We just utilize the hidden fast system-stack via PASCAL's function-parameter mechanism. Recall Picasso's "I don't search, I find."
- Last but not least QuiXpert has a proven educational value. The first three persons mentioned in the Acknowledgements got their first hands-on-and-in experience with an ES from QuiXpert. Some other folks too. None of them has ever complained, on the contrary.

HOW DOES AN EXPERT SYSTEM WORK?

Rule based ESs are based on a solid ground. Knowledge is expressed as an (un)ordered collection of logical statements alias rules. E.g. in QuiXpert:

- \[ F := A \text{ and } B; \]
- \[ G := C \text{ and } D; \]
- \[ H := \text{not}(F); \]
- \[ J := B \text{ and } G; \]
- \[ X := (F \text{ and } H) \text{ or } (J \text{ and } K); \]
- \[ K := G \text{ and } H; \]

If a user (man/machine) asks for a truth-value of any of the left-hand-side (lhs) variables (= consequences), e.g. for \( X \), then:

0. If \( X \) is known to be true/false, then ES answers instantly.
1. If e.g. \( B=C=D=\text{True} \) and \( A=E=\text{False} \), then ES answers "\( X=\text{True} \)", also without asking any question. The trick is clean & simple. A rule-based back-chaining ES substitutes the rhs for its lhs as long as necessary & possible. Like any bunch of algebraic expressions, our rules may be graphically viewed as a forest of (dis)joint (multi)trees, i.e. as a more or less connected loopless graph (= net) of calls. When an unevaluated leaf (= terminal term) is reached, a question is asked or a data base is interrogated.

2. If none of the lhs terms has a value, then ES asks the user (man or machine) "What is the value of A ?"; "What is the value of B ?" ... etc, until it can evaluate \( X \).

Of course knowledge engineering done by real people for realistic tasks has its own real problems and pitfalls. I have tried to design QuiXpert so as to prevent and to detect many of the lurking dangers, most of which are common to any kind of programming, classical or ES-like.

HOW DOES QUIXPERT WORK?

QuiXpert's unique set of neoclassical and new logical operators allows to express (and if necessary to mix) back- or forth-evaluation of any (sub)set of rules. There is no fixed, prescribed mode of inference. The whole magic is done by logic, by means of our operators, which may be combined freely to form any desired mix of inferencing. QuiXpert is a rule-based ES where rules are compiled as real-functions together with the kernel code (all in PASCAL) to form a directly executable code-net of rules. All is handled as uniformly as possible, therefore extensions are not difficult to do. The built-in self-checking of rules, answers and inner invariants is so tight that it is not too easy to introduce a bug which will go undetected when it occurs.
There are 2 basic kinds of rules in QuiXpert:

- Data-rules:
  a) Ask-rule returns a logical value obtained by asking a user an one-line question and reading his logical answer. "Say" displays one line of text (like Ask does), but does not wait for and returns no answer, hence is no rule. It is useful for asking questions longer than one line. E.g.: Say("..."); Say("..."); Ask("...").
  b) COM-rule returns a logical value obtained by some computation and/or by comparison ( <, >, =, etc).

- Inference-rules:
  Infer-rule (the deductive rule proper) is a logical assignment statement which returns a logical value resulting from the evaluation of Infer-rule's right-hand-side (rhs).

Any rule may contain an optional Show("...") call, to show its result. Besides the data-rules which are called on dynamically, static a priori known facts may be loaded. The empty function FACTSINIT could be filled with a code which reads some logic-valued facts from some fact-file. During the internal phase Ask-ing of questions is blocked, but not the COM- and Infer-rules. If and only if the required goal-hypothesis cannot be deduced without Ask-ing some question(s), the external phase will be activated, otherwise it will be skipped. Hypothesis is chosen from a menu (which may stay hidden). A general "no-idea-hypo" is also allowed, if on menu (which needs not to show everything).

After a "run" a user may choose to:

    Quit: or Go on & reuse the answers; or Fresh run.

If no Quit then the user must choose a new goal-hypothesis to evaluate. There is little point in describing QuiXpert or any ES in detail. Those who only read or teach and never try or design, are like the philosophers who disputed about how many teeth a horse has, without opening its mouth. However it is a must to describe in full detail how knowledge is represented in an ES. For rule-based ESs like QuiXpert, the way how the logic rules are written (with which logical operators) is what matters.

******************************************************************************
* * While reading look at the examples at the end of this document. *
* Think about the semantics & logic, not about syntactic sugar. *
* *
******************************************************************************
RULES WITH MULTIPLE CONCLUSIONS:

Expert's rule has a general form: lhs := rhs.
E.g.:

A := (C1 and C2) or C3;  which is equivalent to:

if (C1 and C2) or C3 then A:=true else A:=false.

More than often additional conclusions follow from the result. We need a multiple assignment, something like (A, B, C) := rhs, where B, C are rules existing besides the rule A. That would not be powerful enough to express multiple consequences like e.g.:

if A then begin B:=true; C:=false; ... end
else begin D:=true; E:=false; ... end;

This is one of the important things which PROLOG does not offer. Therefore in QuiXpert-like form (the fine syntactic sugar aside; see the Examples below) the complete rule from above looks very much like:

A := (C1 and C2) or C3;
ThenTrue (# of rule B);  ThenFalse(# of rule C); ...
ElseTrue (# of rule D);  ElseFalse(# of rule E); ...

where THELSE's arguments must be the numbers (2nd names) of rules.

Q: What are these Then... , Else... (henceforth "THELSE"s) good for?
A: 1. THELSEs facilitate explicit expression of chunked knowledge as multiple conclusions. That is very natural for people.
2. THELSEs prevent asking superfluous questions, and evaluations.
3. THELSEs are likely to speedup arriving to final conclusions, especially if we want multiple final conclusions.
4. THELSEs reach (sub)conclusions across the levels of a (multi)tree of rules hierarchy (of calls); see the examples at the end.

Advice: The nearer to the root(s) in a (multi)tree, the more THELSEs should be used. At the leaves THELSEs will be rare.

The ability to express multiple conclusions is a must, not a luxury. Otherwise more rules would have to be written, all with the same rhs, thus spoiling the chunking (= modularity) of knowledge. As implemented, THELSEs do not propagate automatically. The pros & cons are:
+ it is easier because there is no need to check & cut cycles; see the QuiXnote #3 in the Appendix;
+ it is a "lazy" propagation, i.e. it does not propagate what may not be needed, but also:
- it does not propagate what may be useful, hence extra evaluations may take place, but this has an advantage that:
+ it causes more crosschecks among the conclusions caused by THELSEs the conclusions caused by the primary lhs in the rules.

Note: For the 3 logical values True, Maybe, False it is easy to define
additional Thelses: ThenMay, ElseMay, MayTrue, MayFalse, MayMay.

Recalling Goethe's dictum "In der Beschränkung zeigt sich der Meister" we leave these new freedoms and responsibilities (!) to those who will feel a real need for these Thelses.

KINDS OF LOGIC IN QUIXPERT

Real world of real people is not a black-or-white, yes-or-no world. Neither clean cut is the world of real numbers on real machines. A nonconverging computation must be broken neither with the result RootExists = true, nor with RootExists = false. Systems coupled to the real world via sensors must be able to signal "don't know" if the vital sensors return strange values or are (known to be) damaged. Hence an absolute minimum of realism is to allow for an answer "maybe" i.e. "don't know". Besides that there is a "to be (re)evaluated" value (here shown as "?") hidden from the users. Another line of extension are new logical operators which allow to specify certain common semantic or functional relations. Both lines of extension are synthesized so that their synergistic effects allow building (expert) systems which are smarter, safer, faster and more self-documenting.

QuiXpert’s users have a choice from three kinds of logic: #1, 2, or 3.

Input:
- #1 = Boolean: 0 / 1
- #2 = from 0% to 100%
- #3 = 0 / 50 / 100%

Output:
- #1 = Boolean: 0 / 1
- #2 = from 0 to 1
- #3 = 0 / 0.5 / 1

Inside:

Internally used are only the proper "output" values plus a so called "undefined" value. All kinds of illegality are checked at run time. ? = "undefined" value (implemented as e.g. 2 or -22 or "?!"), which is also the virgin initial value of all logical (sub)expressions and variables (e.g. in expert system’s rules).

Users (human or inhuman) have nothing to do with the "?", they cannot enter it and it will never come out (except from the debugger).

The "?" allows an evaluator-algorithm (expression interpreter) to look ahead and see if the value of a (sub)expression E is "defined". If it is value = ? then the evaluator may try to (re)evaluate E so that a "defined" value <> ? may result. If such an attempt fails, evaluator asks for or computes a value of a (sub)term within E until E can be evaluated to a "defined" value ( <> ? ).

There is no end in making an evaluator smarter:
- semi-symbolic evaluations, which is my term for simplification rules involving one variable only, e.g. 0 and A = 0; 1 and A = A; etc; full list further below. Heuristics for smarter evaluations are:
  - the most repeated variable should be asked/evaluated first;
  - key terms first: within ((A or B) and K and (C or D)) it is wise to evaluate K as first, provided it is not a complex term!
  - I have realized that:
    if A = .5 = B in (A or (B and C)), (A and (B or C)), then C needs not to be asked because the result = .5 regardless of C’s value.
- general symbolic simplification: \(((A \text{ and } B) \text{ or } (A \text{ and } \text{non}(B))\) is \(A\), so that \(B\) is never asked.
- discovery of tautologies: \(((\text{non}(A) \text{ or } B) \text{ or } (A \text{ or } \text{non}(B))\) is always true, for all possible "definitive" interpretations ("any mix of assignments of True and/or False"), hence nothing has to be asked:
- numero-logical tautologies: \(((N - N) = \log(1))\) is always true; this looks simple, but there is no end to complexity and subtlety of (in)equality which may be indentically true/false:
- etc.

But even a simple & fast "look left & right" evaluator asks less questions than a classical conditional left-to-right evaluation, especially when our new logical connectives are used.

Human answer is input as an integer so that the first nondigit (except for the leading blanks) and whatever follows it is truncated away. Here we often use .5 instead of 0.5 for psycho-typographical reasons.

Logic-kinds #1 and #3:

"Indefinite" values:  ? = "to be (re)evaluated"; for logic #1, 2, 3.

plus for logic #3: .5 = maybe, don’t know, unsure, possible, some, sometimes, definitive value unavailable; .5 is a result of asking/evaluation.

"Definite" values: 0 = false, no, impossible, never, none;
1 = true, yes, guaranteed, always, all.

"Defined" values: 0, .5, 1. "Undefined" value:  ? .

A "definitive" value of a variable cannot be changed anymore. An "indefinite" value may be changed only if the "informativeness i", defined as: \(i(?) < i(.5) < i(0) = i(1)\), will increase. The "definite" values are also "certain" values, while the "indefinite" values are "uncertain" values. The measure of "certainty" plays a practically more important role for logic # 2.

Logic-kind # 2: Plausibilistic threshold logic

A finer grained multivalued logic was mapped i.e. projected upon the logic #3. A new problem appeared: The choice of a calculus to perform and to retain meaningfully fine grain results of operations.

Again, the "?" is the "indefinite" & "undefined" value. Users are free to choose a threshold value (= pcMIN) beyond which a "definitive" value is reached, and evaluation may stop.
Two plausibilistic thresholds are defined for conditional evaluations, so that:

\[ 0 \leq pc_{\text{MIN}} < 0.5 < pc_{\text{MAX}} \leq 1 \]

and

\[ 0 \leq \text{false} \leq pc_{\text{MIN}} < \text{maybe} < pc_{\text{MAX}} \leq \text{true} \leq 1. \]

Hence

\[
\begin{array}{cccc}
\text{pc_{MIN}} & \text{pc_{MAX}} & \text{false} & \text{maybe} \\
0 & 1 & \text{maybe} & \text{true} \\
\end{array}
\]

The terms "indefinite", "definitive" and "defined" are now clear too.

The "certainty" measure is defined as

\[ \text{cert}(p) = |p - 0.5| \]

for "plausibility" p.

Thus the minimal certainty is \( 0 = \text{cert}(0.5) = \text{cert}(?) \),

the maximal certainty is \( 0.5 = \text{cert}(0) = \text{cert}(1) \),

which justify the following postulates for plausibilities:

- \( 0.5 \text{ and } 0 = 0 \)
- \( 0.5 \text{ or } 0 = 0.5 \)
- \( 0.5 \text{ and } 1 = 0.5 \)
- \( 0.5 \text{ or } 1 = 1 \)

which added to the classical Boolean truth-tables almost forced us to define:

\[
\begin{align*}
\text{U and W} &= \min(U, W) \\
\text{U or W} &= \max(U, W).
\end{align*}
\]

From these we have developed more complex formulas for our new logical operators (we shall not discuss these formulas here). Just remember:

\[
\begin{align*}
\text{non}(V) &= 1 - V \\
\text{U and W} &= \min(U, W) \\
\text{U or W} &= \max(U, W)
\end{align*}
\]

for partly overlapping alternatives U, W.

The advantages of these simple formulas are:

+ they work well for logic #2;
+ they are correct for logic #3 (see the tables below) which was formally derived from Boolean logic and then crosschecked by several methods as well as by common sense;
+ they are correct for Boolean logic:
+ they satisfy DeMorgan's laws:

\[
\begin{align*}
\text{U and W} &= \text{non}(\text{non}(U) \text{ or } \text{non}(W)) \\
\text{U or W} &= \text{non}(\text{non}(U) \text{ and } \text{non}(W))
\end{align*}
\]

proof:

\[
\begin{align*}
\text{U or W} &= 1 - \min((1-U),(1-W)) = \max(U,W) \text{ for } 0 \leq U, W \leq 1; \\
\text{U and W} &= 1 - \max((1-U),(1-W)) = \min(U,W) \text{ for } 0 \leq U, W \leq 1.
\end{align*}
\]

+ they are easy to work with.

The more complex formulas for our new operators are shown below after the operators were explained.
PROGRESSIVE REEVALUATION.

"Informativeness i" is defined as: \( i(?) < i(.5) < i(0) = i(1) \), in agreement with common sense.
"Progressive" means nondecreasing informativeness, which must change monotonically, i.e. it must not decrease but it may stay unchanged.
"Multipath" means that there may be more than one single path (of deduction) via which a (plausibility) value of a (logical) variable may be inferred. Whether such a reevaluation will actually occur depends on many unpredictable factors (e.g. the answers).

Overview of progressive changes:

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>is defined as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>0</td>
<td>legal</td>
</tr>
<tr>
<td>?</td>
<td>.5</td>
<td>legal</td>
</tr>
<tr>
<td>?</td>
<td>1</td>
<td>legal</td>
</tr>
<tr>
<td>.5</td>
<td>0</td>
<td>legal, reported as interesting</td>
</tr>
<tr>
<td>.5</td>
<td>1</td>
<td>legal, reported as interesting</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>illegal, reported as CONTRA-diction error</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>illegal, reported as CONTRA-diction error</td>
</tr>
</tbody>
</table>

All other changes are "nonprogressive" and skipped, i.e. not done.

How progressive changes occur:

0. Virgin initial state after a fresh start with all values set to the 0 value. Thereafter facts, if any, may be read & loaded.
1. The ? may change into any other value, but never vice versa.
2. A multipath evaluation may change "maybe" to a "definitive" value.
   Then repeated singlepath evaluations may propagate the corresponding changes (considered as legal and reported by QuiXpert).
   As long as no multipath is executed (regardless of if it exists), no "maybe" will change to a "definitive" value.
   A "definitive" value will not be changed to "indefinitive" value (we do not let QuiXpert to report such a reevaluation).
3. A reevaluation to a different "definitive" value is reported as an error (a CONTRA-diction).
4. Any reevaluation to the same value (also ? to ?) is allowed and unreported. However it is impossible to finish with the final result = ?, but .5 is allowed.

Multipath evaluations may be present only implicitly, i.e. not by design. Explicit programming of multipath evaluations is possible by means of the new "alt" operator (see below). It provides an alternative "path of knowledge". Any number of alternative paths is easily expressed by further "alt"-ing.

Users have nothing to do with "?", they cannot enter it, and will never see it as an output (except of a debugger). The "?" allows the evaluator algorithm to look ahead at both U, W in any (sub)expression U op W. The "?" allows the evaluator to try to evaluate a (sub)expression so that a value other than ? will result.
A smart evaluator would have to be a symbolic one. It would have to
algebraically simplify e.g. (A and B) or (A and non(B)) into A. It would have to recognize tautologies i.e. expressions which are always true/false, e.g. Y or (non(Y) and non(X)) or (non(Y) and X). There is no general optimal strategy for evaluation, such that the minimal number of questions (to man or machine) will be asked. Although a "fast format" of rules (which evaluates each rule only once) is available for QuiXpert, I prefer to reevaluate all Infer-rules (but of course not the Ask- and COM-rules) for the following reasons:
- propagation of progressive changes is possible (mainly via THELSEs);
- more checking (of possible inconsistencies in the rules) is done.
This is not so inefficient as it may seem because, unlike most of the other expert systems, QuiXpert never searches for matching rules; it always knows instantly which to use when.

NEW LOGIC OPERATORS ARE THE HEART & BRAIN:

I have tried hard to design new logical operators which fit, support and allow semiintuitive thinking as well as strictly rational thinking. I have spent considerable effort to check-as-check-can their consistency in abstracto, as well as their correct implementation. However I do welcome any comments, ideas or findings from the reader.

SINGLE-ARGUMENT I.E. MONADIC OPERATORS:

Concise overview:

If V=? {or in the table result=?} then ask V, and evaluate as follows:

non : result := 1 - V;
neg : if V = 0 then result := 1 else result := 0;
may : if V = .5 then result := 1 else result := 0;
yes : if V = 1 then result := 1 else result := 0.

Quiz: Why non(yes(V)) = neg(yes(V)), despite of that in general it holds non(V) ≡ neg(V) ?
Examples of definitions:
Separate:
\[
\text{male} := \text{man}; \quad \text{female} := \text{woman}; \quad \{\text{neutral definition}\}
\]
Coupled:
\[
\text{male} := \text{man}; \quad \text{female} := \text{non(male)}; \quad \{\text{macho definition}\}
\]
\[
\text{female} := \text{woman}; \quad \text{male} := \text{non(female)}; \quad \{\text{feminist definition}\}
\]
\{the use of non() preserves .5 (transsexuals)\}
Cyclic (forbidden):
\[
\text{male} := \text{non(female)}; \quad \text{female} := \text{non(male)};
\]
Examples:
\[
\text{guarantee} := \text{neg(RiskyBusiness)};
\]
\[
\text{clearance} := \text{yes(secure)};
\]
\[
\text{identifiable} := \text{yes(Object1 or Object2 or Object3)};
\]
\[
\text{FireMissile} := \text{yes(EnemyAircraft)}; \quad \{\text{nonaggressive defense}\}
\]
\[
\text{FireMissile} := \text{non(neg(EnemyShip))}; \quad \{\text{aggressive defense}\}
\]
Note that in the old Boolean logic there is no need for neg, may, yes.
Frankly speaking, I think that they will be used only on rather special
occasions, because our new dyadic operators provide for typical use.

LOGICAL CONNECTIVES I.E. DYADIC OPERATORS:

First a concise overview, details and examples follow further below.
McCarty's (1960) classic conditional evaluation (in C, Modula and in
some Pascals) proceeds strictly from the left to the right as follows:
\[
R := U \text{ and } W; \quad \text{ works as: if } U \text{ then } R := W \text{ else } R := \text{false};
\]
\[
R := U \text{ or } W; \quad \text{ works as: if } U \text{ then } R := \text{true else } R := W;
\]
hence the 2nd operand W is not always evaluated/asked, only if needed.
Our connectives ANDc, ORc (table below) are conditional too. But the
special value "?" makes it possible to "look ahead at both U, W" and
thus e.g. also to avoid asking for U, if W is known and suffices:

If in the table result <> ? then U, W will not be asked anymore, else
if in the table result = ? then if U <> ? then ask W, else ask U.

However our logic is useful even for the left-to-right evaluation.
Our unconditionally evaluated ORu, ANDu are not shown in the table
simply because they yield the same results as their conditionally
evaluated cousins ORc, ANDc, except for the last 6 lines in the table
where ORu, ANDu produce the result = ? only. The result = ? means that
at least one of the inputs U=? or W=? will be asked/evaluated.

Not only the value .5 (= don't know = maybe) begs for new operators.
New operators allow the knowledge engineer/programmer to specify his
or her knowledge/assumptions about certain semantic constraints between
the operands U, W. These common semantic constraints are best visual-
ized by means of Venn diagrams for OR-like alternatives:
Partly (!) overlapping
alternatives:

\[
\begin{array}{c}
U \text{ or } W \\
\hline
1 \quad U \quad W \\
\hline
1 \quad U \quad I \\
\hline
1 \quad I \quad W \\
\hline
I \quad I \quad I \\
\end{array}
\]

Disjoint i.e. mutually exclusive
alternatives:

\[
\begin{array}{c}
U \text{ Xalt } W \\
\hline
1 \quad I \quad I \\
\hline
1 \quad U \quad I \\
\hline
1 \quad I \quad W \\
\hline
I \quad I \quad I \\
\end{array}
\]

Semantically/functionally
equivalent alternatives:

\[
\begin{array}{c}
U \text{ alt } W \\
\hline
1 \quad U = W \\
\hline
1 \quad I \\
\hline
1 \quad I \\
\hline
I \quad I \\
\end{array}
\]

Superalternative:

\[
\begin{array}{c}
U \text{ Salt } W \\
\hline
1 \quad U \quad I \\
\hline
1 \quad I \quad W \\
\hline
1 \quad I \\
\end{array}
\]

Subalternative:

\[
\begin{array}{c}
U \text{ altS } W \\
\hline
1 \quad U \quad I \\
\hline
1 \quad I \quad W \\
\hline
1 \quad I \\
\end{array}
\]

The advantages:

+ The very existence of new operators with nice pay-offs encourages
  a programmer to think explicitly about the semantic or functional
  relations and constraints, if any, between the operands U, W. Any
  a priori known constraint is a source of useful information for man
  and machine. Here we get safer, smarter and faster (expert) systems.
  + The knowledge/assumptions are self-documented because the logical
    operators specify them explicitly. This is of great help for main­
    tenance, extensions and changes, which are the very essence of a grow­
    ing and adapting (expert) system.
  + Partial semantic checking is automated (see ERR in the table below).
  + Less questions are asked from human users and from data bases, be­
    cause more conditional evaluations occur than with classical COR,
    CAND, due to the semantic information carried by new connectives.
  + Advanced implementations may often infer the value of one operand
    in advance from the value of the other one, because new connectives
    specify how they are dependent. See "Advance Inferences" below.

Examples of new ORc-like unifiers:

\[
\text{parent} := \text{father} \text{ Xalt } \text{ mother} \quad \{ \text{for mutually exclusive alternatives} \} \\
\text{heavy} := (\text{"weight} > 5\text{"}) \text{ alt } (\text{"volume }\times \text{ specific weight} > 5\text{"}) \\
\text{safe} := (\text{"voltage} = 0\text{"}) \text{ alt } (\text{"well insulated"}) \\
\text{up} := (\text{"volume increased"}) \text{ alt } (\text{"temperature increased"}) \\
\text{down} := (\text{"volume decreased"}) \text{ alt } (\text{"pressure increased"}) \text{ alt }
\text{"temperature decreased"}; \\
\text{carnivore} := \text{EatsMeat} \text{ Salt } (\text{HasPointedTeeth} \text{ or } \text{HasClaws}); \quad \{ \text{people have no pointed teeth or claws} \} \\
\text{c} := \text{fish} \text{ altS } \text{ cake} \quad \{ \text{1st is a Superalternative for 2nd} \} \\
\text{c} := \text{fish} \text{ altS } \text{ cake} \quad \{ \text{1st is a subalternative for 2nd} \}
\]
Summary:
ORc-like unifiers (all conditionally evaluated; Venn diagrams below):
oc = "or" conditionally evaluated; the general unifier
Xalt = "or" for "eXclusive alternatives" U, W: a special case unifier
Xalt is NOT the Exclusive OR alias xor alias nonequivalence;
compare "Xalt" with "xor" in the next two tables below.
al = "or" for "equivalent alternatives" U, W; a special case unifier
"alt" is NOT the classical "equivalence" operator "eqv";
compare "alt" with "eqv" in the next two tables below.
Salt = "or" for U is "Superalternative" to W; a special case unifier
altS = "or" for U is a "subalternative" to W; a special case unifier

Examples of ANDc-like intersects:
mild := non(cold) andc non(hot) { or: via DeMorgan's law: }
mild := non(cold orc hot)
mild := ('temperature > 10 deg C') andc non('temperature > 30 deg C')
right := a triangle Truethen ((90 degrees angle) alt (A*A + B*B = C*C))
{ Truethen is like ANDc except for one practically important case: if U = "don't know" then W is not asked for, and the result := "don't know".
If a necessary precondition (= 'key') U \_ true
then do not ask further details W (= what is in the house, what kind of bug is in the machine, etc.) }
GoingOut := ('she is ill') Falsethen ('theater tickets available');
{ If he doesn't know if she is still ill he will not consider the problematique of money, flowers, diamonds, etc. }
{ Like Truethen with inverted precondition, i.e. non(U).
Only if the necessary precondition (= 'key') U = false,
then it makes sense to ask/evaluate W }

Summary:
ANDc-like intersects: U is the necessary precondition (= 'key') for W:
andc = "and" conditionally evaluated.
Truethen = "U is a positive-key to W" ( ◇ yes(U) andc W !)
Maythen = "U is a doubtful-key to W" ( ! = ! may(U) andc W !!)
Falsethen = "U is an inverted-key to W" ( ◇ neg(U) andc W !)
( ◇ non(U) andc W !)
The only difference between (U falsethen W) and the (non(U) andc W) is for U = 0.5 & W = '?', in which case the first yields 0.5 (marked by *) and the last yields the '?'.
The legend:
The .5 stands for 0.5 (to enhance the readability of the tables).
ERR reports a serious error, a reason to stop and fix it. However a
pragmatic decision for some applications may be to (try to) go on.
It is for such non-stop systems that we return result =.5, besides the
ERR-or message and/or ERR-or flag.
Commutative connectives are: andu, oru, andc, orc, Xalt, alt.
It follows from the table's last 10 lines, or from Venn diagrams.
Unconditionally evaluated andu, oru yield "?" in table's last 6 lines.
Conditional evaluation occurs wherever result <> ? in the last 6 lines.
If the result <> ? then U, W are not asked anymore, else:
if the result = ? then if U <> ? then W asked else U asked.
+ The result=1 per definition (hence not derived; explained below).
Note that the lines 7 to 10 are analogical to the lines 11 to 14,
except for the result marked by the "+".
*

The result=.5 per definition. According to our derivation principle
the result should be the ? , the same as for (U andc W). That is not
desirable if the meanings of $U$, $W$ are such that if $U=.5$ then it makes little sense to evaluate/ask questions about $W$. One might even argue that $(U=.5, W=0)$ should yield result=.5, which would mean radical departure from AND. Since I did not want to introduce yet another bunch of operators, I've done what I’ve done.

"Similar" means "subtly different". It does not mean that the difference is negligible, on the contrary. It signals "watch out"!

I have implemented all operators as functions which handle all 3 kinds of logic in QuiXpert in as unified way as possible. However if only the logic #3 would be used, then it could be implemented by a simple and fast table lookup with integer or char-acter values.

**ADVANCE INFERENCES**

Implementations more advanced than my current prototype may produce "advance inferences". The information carried by new connectives can be used to make many definitive inferences (here shown as assignment statements := ), but no more than the following ones:

- if $0 \text{ Xalt } W$ then $W := 0$
- if $U \text{ alt } 0$ then $U := 0$
- if $0 \text{ Salt } W$ then $W := 0$
- if $U \text{ Salt } 1$ then $U := 1$
- if $1 \text{ Xalt } W$ then $W := 0$
- if $1 \text{ Xalt } U$ then $U := 1$
- if $U \text{ Xalt } 1$ then $U := 1$
- if $1 \text{ Xalt } W$ then $W := 1$

Note: $0 \text{ Xalt } 0 = 0$ i.e. $\text{Xalt}$ does not mean $U = 1 - W$ i.e. $W = 1 - U$. Such a complementarity is too rare to justify a special operator. It is easily expressed as: $U \text{ alt non}(W)$, or as: $\text{non}(U) \text{ alt } W$. This is explained in more detail below.

See how advance inferences match with ERR-ors and with conditional evaluations in the table of connectives (lines 11 to 14, or 7 to 10).

A practical implementation of advance inferences makes sense only for those inferred operands ($U:=...$; or: $W:=...$) which are simple variables or their inversion ($\text{non}(U):=...$; $\text{non}(W):=...$), and if neg / may / yes yields 1. It makes no sense to infer a value of a nameless subexpression (unless it is a common one, and we would detect it).

**SEMI-SYMBOLIC CONDITIONAL EVALUATIONS**

Much more frequently than advance inferences, conditional evaluations cut superfluous questions to users and to data bases. Only seldom it is necessary to evaluate/ask the values of all the variables involved. An all-way unrestricted conditional evaluation would stop at the first occasion when the result could not be changed by evaluating further variables in an algebraic/logical expression. Unlike the classical conditional evaluation, we do not want to proceed rigidly left-to-right.
Our internal-only value "?" allows an evaluator to look ahead and thus to return the result = 0 also for (U andc W) where (U = ?, W = 0). Obviously the result = ? must not occur if (U <> ?, W <> ?).

If the result = ? then if U <> ? then W asked else U asked/evaluated. Conditional evaluations occur wherever there is a result <> ? in the last 6 lines of our tables of (non)standard logical connectives. The unconditionally evaluated ORu, ANDu are not shown in the table simply because they result in the same values as their conditionally evaluated cousins ORc, ANDc, except for the last 6 lines, where both ORu, ANDu always yield the result = ? only.

The better the conditional evaluation, the smarter and faster the (expert) system. While writing this document I have generalized the notion of conditional evaluation so that it includes checking of all kinds of tautologies, initial (= total) and "residual" (partial) ones. The internal-only value "?" allows to evaluate e.g. (U orc W) as 1 if (U = ?, W = 1) without asking U. The left-to-right evaluation is not a rigid rule anymore, just a general tendency (from the initial state of all variables = ?).

Let the "omnidirectional" conditional evaluation mean the same as the "semi-symbolic" simplification. Both are defined as:

An exhaustive application of all the relevant rules (for algebraic simplification) involving only constants and/or at most one appearance of at most one variable on lhs, while rhs is the simpler equivalent. Exhaustive application proceeds until no more rules can be applied. Thus A or A = A is not "semi-symbolic", because we want to keep it simple, but still powerful.

Our neoclassical rules for "semi-symbolic" evaluation are:

Monadic:

\[
\begin{align*}
\text{non}(0) &= 1 \\
\text{neg}(0) &= 1 \\
\text{may}(0) &= 0 \\
\text{yes}(0) &= 0 \\
\text{non}(0.5) &= 0.5 \\
\text{neg}(0.5) &= 0 \\
\text{may}(0.5) &= 1 \\
\text{yes}(0.5) &= 0 \\
\text{non}(1) &= 0 \\
\text{neg}(1) &= 0 \\
\text{may}(1) &= 0 \\
\text{yes}(1) &= 1 \\
\end{align*}
\]

We do not need the rule \( \text{non(\text{non}(A))} = A \).

Dyadic: the rules for Xalt are analogical to rules for orc, except for 1 Xalt 1 = ERR to be checked first.

\[
\begin{align*}
0 \text{ andc } W &= 0 \\
U \text{ andc } 0 &= 0 \\
1 \text{ andc } W &= W \\
U \text{ andc } 1 &= U \\
.5 \text{ andc } .5 &= .5 \\
\{ .5 \text{ andc } 1 &= .5 \} & \{ .5 \text{ andc } 0 = .5 \} & \{ .5 \text{ andc } 1 = 1 \} & \text{ are not needed} \\
\end{align*}
\]

The semi-symbolic simplification/evaluation rules can be distilled from the table of operators above. For example the rule \( U \text{ Salt } 0 = U \).

Further all "commutations" of the following rules formulated while writing this document, and which did occur to me in practice:
The number of all the (commutations of these) rules is not small enough hence we shall consider the technique of "truth-tree" analysis as a single uniform way how to obtain the effect of application of all the non-monadic rules, without really implementing them as such.

**TRUTH-Tree Analysis for Conditional Evaluation and Tautologies:**

A full symbolic simplification would be only seldom economically sound for all but a few (expert) systems. We can do the most, if not always all the run-time simplification, most of the time by means of a truth-tree analysis as introduced by W. V. O. Quine [1]. He has used it for a static analysis of logical expressions in symbolic form.

I have realized that truth-trees can be profitably employed in expert systems for run-time semi-symbolic simplification and for checking of (residual) tautologies.

Like Quine we must use only the two "definitive" values True and False despite that our logic does allow the third value "Maybe". Obviously with enough Maybes all branches would yield a Maybe too easily, which does not mean that an expression or rule is a tautology.

I do not know whether the following is a known theorem but here it is. Like many a theorem ours is trivial once it is spelled out, but useful nevertheless.

**Theorem:** A logical (sub)expression cannot be identically true or identically false (i.e. under all interpretations) if:
- it does not contain a constant which reduces it to a constant, and
- it does not contain more than one appearance of the same variable, and all variables are formally and semantically independent.

**Proof:** By common sense or by intimidation.

**Use:** This theorem allows us to skip a tautology check.

**Problem:** The semantic (in)dependence of variables. Formal (in)dependence can be established by full substitution down to the most elementary values.

**Note:** Just before this report was put to print Mr. P.M.G. Boon has pointed out his belief that an expression cannot be a tautology if it does not contain at least one variable X together with its direct or indirect negation. By indirect negation we mean that after all operators are (converted into) AND, OR, NOT, there must be also NOT( BooleanFunction(X) ) present. I believe that he is right, but since I have introduced other operators irreducible to AND, OR, NOT, I would not bet on it.
RULES (TRANS)FORMATION RULES:

non(non(U) andc non(W)) = U orc W \{ \text{DeMorgan's law} \}
non(non(U) orc non(W)) = U andc W \{ \text{DeMorgan's law} \}

non(U) false then W = U true then W \{ \diamond \text{yes(U) andc W} \}
may(U) true then W = U may then W \{ \text{may(U) andc W} \}
non(U) true then W = U false then W \{ \diamond \text{neg(U) andc W} \}
\{ \diamond \text{non(U) andc W} \}

non(neg(U)) true then W = neg(U) false then W \{ \text{if U} \leftrightarrow 0 \text{ then ask W} \}
Note:
\text{results as for } U \text{ true then W except for: result=} 1 \text{ for } U=.5 \& W=1
\text{and: result=} ? \text{ for } U=.5 \& W=?

non(yes(U)) true then W = yes(U) false then W \{ \text{if U} \leftrightarrow 1 \text{ then ask W} \}
Note:
\text{results as for } U \text{ false then W except for: result=} 1 \text{ for } U=.5 \& W=1
\text{and: result=} ? \text{ for } U=.5 \& W=?

\text{It holds:}
(U Salt W) = (W alt S U)
U alt W = (U Salt W) andc (W Salt U)
U alt W = (U Salt W) andc (U alt S W)
U alt W = (U alt S W) andc (W alt S U)
HOW TO CONSTRUCT NONSTANDARD OPERATORS:

From the classical Boolean formulas (see the Appendix) we can construct nonstandard neoclassical plausibilistic connectives. For example:

\[ U \text{ imp } W = \text{non}(U) \text{ orc } W \quad \{ \text{U implies } W \} \]
\[ U \text{ pm} i = U \text{ orc non}(W) \quad \{ \text{U is implied by } W \} \]
\[ U \text{ eqv } W = (U \text{ orc non}(W)) \text{ andc } (\text{non}(U) \text{ orc } W) = (U \text{ pm} i W) \text{ andc } (U \text{ imp } W) \]
\[ U \text{ xor } W = U \text{ neg } W = \text{non}(U \text{ eqv } W) \]
\[ U \text{ dif } W = U \text{ andc } \text{non}(W) = U \text{ inhibited by } W = U \text{ unless } W \{ \text{more below} \} \]

\[ U \text{ alt } \text{non}(W) \text{ for OR-ing of complementary alternatives } \{ U = 1 - W \} \]
\[ \text{non}(U) \text{ alt } W \text{ for OR-ing of complementary alternatives } \{ 1 - U = W \} \]

\[ U \text{ Salt } \text{non}(W) \text{ for } (U \text{ or } W) \text{ on overlapping } U, \text{ W, if } U \text{=} 0 \text{=} W \text{ cannot occur,} \]
\[ W \text{ Salt } \text{non}(U) \text{ for } (U \text{ or } W) \text{ on overlapping } U, \text{ W, if } U \text{=} 0 \text{=} W \text{ cannot occur.} \]

Because "the proof of the pudding is in eating it", and because "seeing is believing" I went through the hustle of showing it. Our insight and confidence are further increased.

\[ U \text{ xor } W = (U \text{ orc } \text{non}(W)) \text{ andc } (\text{non}(U) \text{ orc } W) = (U \text{ pm} i W) \text{ andc } (U \text{ imp } W) \]
\[ U \text{ xor } W = U \text{ neg } W = \text{non}(U \text{ eqv } W) \]

Because "the proof of the pudding is in eating it", and because "seeing is believing" I went through the hustle of showing it. Our insight and confidence are further increased.

\[ U \text{ Salt } \text{non}(W) \text{ for } (U \text{ or } W) \text{ on overlapping } U, \text{ W, if } U \text{=} 0 \text{=} W \text{ cannot occur,} \]
\[ W \text{ Salt } \text{non}(U) \text{ for } (U \text{ or } W) \text{ on overlapping } U, \text{ W, if } U \text{=} 0 \text{=} W \text{ cannot occur.} \]

Because "the proof of the pudding is in eating it", and because "seeing is believing" I went through the hustle of showing it. Our insight and confidence are further increased.
Quiz: what about may(U) ?

Conditional evaluation occurs if result <> ? in table's last 6 lines. This table of nonstandard neoclassical connectives can also be derived directly from its first 6 lines by means of our algorithm explained below. But it is easier to use the classical Boolean formulas which still hold.

CAPTURING THE COMMON SEMANTIC DEPENDENCIES

Knowledge-rules are logical statements where operands (i.e. arguments) are logical terms, which often represent a class/set of objects. Since there is a strong analogy between logic and a class/set membership, we can visualize the latter by means of Venn diagrams. For a pair of inputs U, W we can draw all the important special cases. Why important? Because any a priori knowledge about relationship (if any) between two classes/sets yields multiple advantage:
+ The knowledge and assumptions are made explicit. This is vital for maintenance, extensions (ESs grow), for self-documentation.
+ Partial semantic checking (see ERR-or in the table above).
+ Conditional evaluations are potentially more frequent (see the table above). Hence asking of superfluous questions (which cannot influence the result) is avoided. Even if no questions are asked, speedups may occur due to "arboricide" (= "cutting of the tree"; re: every expression is evaluated as a "tree"). The above table shows that alt, Salt, altS lead to more conditional evaluations than ORc (Xalt does not, but catches ERR-or).

************** Advice: *****************
* Use new operators alt, Salt, altS, Xalt instead of ORc *
* wherever reasonable. They cut asking of questions, they *
* cut computations, and catch ERR-ors in logic. *
* *
***********************

+ Advance inferences of plausibility values of some variables/rules may be performed as useful side-effects.
+ Alternatives provide for robustness. E.g. if one method of computation of a root of a function does not converge (hence can yield neither rootexists = true, not rootexists = false), then the result rootexists = maybe is the proper one, and an alternative method (e.g. a more expensive one) may be tried, etc.

***********************
* Don’t think that what follows is a complication. It is the complex world we want to capture SHARPLY with our logic. *
* This logic allows to convey some knowledge about your microworld to your computer. You’ll see the difference if you compare. *
In the following examples you should try to answer a term with "don't know" (= .5) versus with "false" (= 0), and compare the questioning and the results with the classic COR, or ORc only.

Our new ORc-like operators Xalt, alt, Salt, altS allow to express our a priori knowledge about the semantic (non)overlapping or semantic equivalence of the operands which quite often represent some mutually (non)disjoint (sub)set or (super)class dependencies.

Case 0: Partially intersecting i.e. partly overlapping alternatives:

In general X is non empty, i.e. U=O=W may occur. This case cannot be ERR-checked because it is not subject to any semantic constraints like the following cases 1, 2, 3a, 3b. Just use the ORc.

E.g:

\texttt{Sexy := LongLegs ORc MonoKini}

The extreme subcase is when we know a priori that U=O=W cannot occur in the given context. Then X is empty (because the union of partly overlapping U, W covers all the known/intended possibilities) and we get:

\begin{itemize}
  \item[a)] U Salt non(W) \{ = non(W) altS U \}
  \item[b)] W Salt non(U) \{ = non(U) altS W \}
\end{itemize}

The choice between a) and b) depends on the semantics of the union. Example: Let the sign of the zero be always positive (handy for square root etc.):

\texttt{PositiveSign := NonNegativeNr Salt non( NonPositiveNr );}\n
\texttt{NonNegativeNr = false suffices to yield result = false. Just think how this might be done with classic COR, CAND, or with our neoclassic ORC, ANDC correctly for all inputs! NonNegativeNr = maybe \& NonPositiveNr = true yields maybe, while ORc yields an incorrect result = false.}\n
Remember that now we talk about OR-like (super/sub)alternatives, and not about AND-like intersections, e.g. not about:

\texttt{Zero := NonNegativeNr andc NonPositiveNr ; MilitaryDuty := (\textquote{Age > 18}) andc (\textquote{Age < 60}).}
Case 1: Disjoint i.e. mutually exclusive alternatives:

If a priori known case, then use Xalt instead or ORc.
In general the X is not empty, i.e. U=W=W may occur.

E.g:

bird := FlyingBird Xalt (penguin Xalt ostrich)

Quiz: what about kiwi, dodo, Kentucky fried chicken, and the like?

The first Xalt is due to the sad fact that some birds cannot fly.
The other Xalt express our wish to identify uniquely, i.e. a bird cannot be a penguin and an ostrich simultaneously. Of course it all depends on what (kind of knowledge) we wish to express, for which microworld is the knowledge expressed under which assumptions.

E.g:
SexuallyNormal := man Xalt woman; { think about U=W }

The extreme subcase is when we know a priori that U=W cannot occur in the given context. Then disjoint, mutually exclusive alternatives are "complementary" i.e. their union exhausts all the known/intended possibilities. Then the extraneous X between U, W in our Venn diagram is empty and void, so that Xalt degenerates into:

\[ \text{lhsU} := \text{U alt non(W)}; \]
\[ \text{lhsW} := \text{W alt non(U)}; \]

E.g:

male := man alt non(woman)
female := woman alt non(man)

Quiz: Consider men, women, transsexuals, androgynous, eunuchs, etc. Face the fact that the world is not so simple.

Q: What if a lhs is a union of more than two disjoint operands (e.g. M, W, T) which together exhaust the whole known or intended "micro-universe" i.e. all the possibilities of a lhs, and M = W = T = 0 is a priori known to be wrong?

A: Split the operands into "(one of them) alt non(all the rest)"

\[ \text{lhsM} := \text{M alt non(W Xalt T)}; \]
\[ \text{lhsW} := \text{W alt non(M Xalt T)}; \]
\[ \text{lhsT} := \text{T alt non(M Xalt W)}; \]

E.g:

male := Man alt non(Woman Xalt Transsexual)
female := Woman alt non(Man Xalt Transsexual)
tsex := Transsexual alt non(Man Xalt Woman)

Caveat:
It is the requirement of the impossibility of U=W=W (for 3 variables the impossible M = T = T = 0 ) which dictates such a solution. Suppose that the set of all birds is exhaustively covered by flying birds penguins, ostrichs, kiwis, and nothing else. Then we must still write:
Bird := FlyingBird Xalt (Penguin Xalt Ostrich Xalt Kiwi)

If an animal we trying to identify may be no bird at all. It all depends on the semantics and on the context of call/use.

Case 2: Alternatives semantically or functionally equivalent in the given context/application.

If a priori known case, then use ALT instead of ORc.

\( U = W \)

c := cetaceous alt (porpoise alt (dolphin Xalt whale))
equilateral := (A=B=C) alt (3 angles of 60 degrees)
CloseRelative := (parent alt (father Xalt mother)) Xalt (child alt (son Xalt daughter)).

Case 3a: Superalternative: U includes W, but not vice versa:

If a priori known case, then use Salt instead of ORc.

\( U \subseteq W \)

E.g:

\( U \equiv W \)

ungulate := hoofs Salt (chews Xalt piggy);
c := cetaceous Salt (dolphin Xalt orca);
w := whale Salt orca;
o := ostrich Salt (emu Xalt nandu);
H := hairy Salt bat: { see the comment at 3b !! }
Q := quadrilateral Salt (trapezoid Salt (parallelogram Salt rectangle Salt square));
Mammal := milky Salt ( (hairy Salt bat) Xalt (cetaceous Salt (whale Xalt dolphin)));

In general if U is simpler expression than W, then (U Salt W) is to be preferred over (W altS U). If U, W are comparably complex, then that one more frequently "true" should come first (also for ORc, Xalt); this helps us to decide between (U Salt W) versus (W altS U). But sometimes we may wish "semantically smooth questioning". See also "Practical Patterns" below.

Case 3b: Subalternative: U included in W, but not vice versa:

If a priori known case, then use altS instead of ORc.

\( U \subset W \)

E.g:

\( U \equiv W \)

w := orca altS whale; o := emu altS ostrich;

h := Bat altS Hairy; is safer than Hairy Salt Bat due to the sad fact that some folks may answer Hairy = 0 while thinking of a Bat. The commutative ORc could be better in such a rare case; frequent 1st: Hairy ORc Bat, thus "leaving the problematique of (non)overlap to the bats". See also the "Practical Patterns".

Remember: \( U \subseteq W \equiv W \subseteq U \). The result will be same for \( 1hsU := U \subseteq W \) as well as for \( 1hsU := W \subseteq U \).
Both serve exactly the same purpose: to express that

\[
\text{lhs}U := U \text{ if } U \text{ is "definitive"/"certain"}, \text{ otherwise } \text{lhs}U := W.
\]

They differ only in the order of conditional evaluation/questioning, which is left-to-right in general (initially all values = ?). Since the less questions the better, we choose between \( U \text{ Salt } W \), \( W \text{ altS } U \) accordingly.

\( U \text{ Salt } W \) works as: if \((U = 1) \text{ or } (U = 0)\) then result := U 
else result := W.

\( W \text{ altS } U \) works as: if \( W = 1 \) then result := 1 
else result := U.

Quiz: What are the differences among the following formulations of (y)our limited knowledge about those lovely beasts?

\[
c := (\text{cetaceous orc porpoise}) \text{ orc (whale orc dolphin)}
c := (\text{cetaceous alt porpoise}) \text{ alt (whale Xalt dolphin)}
c := (\text{cetaceous alt porpoise}) \text{ Salt (whale Xalt dolphin)}
c := (\text{cetaceous alt (porpoise Salt (whale Xalt dolphin)})
\]

Hint: Compare an answer .5 with 0, per term, per line, per pair.

Hint: If \( B, D \) together do NOT make a full i.e. complete \( A \), then write

\[
R := A \text{ Salt (B Xalt D)} \quad \text{for disjoint } B, D
R := A \text{ Salt (B orc D)} \quad \text{for overlapping } B, D.
\]

Why? Because if \( A=.5 \text{ & } B=0=D \) then result \( R=.5 \), as it should be.

But if \( B, D \) together do make a full i.e. complete \( A \), then write

\[
R := A \text{ alt (B alt non(D))} \quad \text{for disjoint } B, D
R := A \text{ alt (B Salt non(D))} \quad \text{for overlapping } B, D.
\]

Why? Because if \( B=0=D \) then result \( R = \text{ERRor} \), as it should be, because \( B, D \) together exhaust \( A \) fully.

Note that ALT may be viewed both as ORc or ANDc. Quiz: Why?

In analogy to the ORc-like operators Xalt, Salt, altS, similar special case operators Xand, Sand, andS could be defined. But Xand makes little sense because: Xand would yield all results = 0 except if \( U=0=W \) then ERR. Only Sand, andS yield nontrivial results (shown "down" from \( U=0=W \)):

\[
U \text{ Sand } W: \quad 0, \text{ ERR}, 0, 1, .5, ?, 0, 0, .5, 1, 0, 0, ?, 1, ?, ?.
\]

Note that Sand cannot be obtained from Salt (or vice versa) by means of DeMorgan's law. Quiz: Why? To try is to understand. As long as I do not see any typical use for Sand, I will not implement it. Can you think up an example?

Considering the fact that knowledge-engineer's a priori knowledge
(about the constrained relation between U, W) is available, it makes little sense for him/her to use Sand, andS, Xand because it is trivial to avoid their use. Quiz: How? This is not so for ORc-like unifiers.

UNDERSTANDING THE LOGICAL TABLES:

Comutativity: andu, andc, oru, orc, alt, Xalt are commutative. This follows from the last 10 lines in the table, or from Venn diagrams.

DeMorgan's laws hold between andc, orc (also between andu, oru), iff we use "non". Quiz: Why not for "neg"? Hence it does hold:

\[ \text{non(non(U) andc non(W))} = \text{U orc W} \]
\[ \text{non(non(U) orc non(W))} = \text{U andc W} \]

It does hold: U false then W = non(U) true then W

It also holds:

\[ \text{U true then W} = \text{non(U) false then W} \]
\[ \text{U may then W} = \text{may(U) true then W} \]

as well as

\[ \text{(U Salt W)} = \text{(W altS U)} \]
\[ \text{U alt W} = \text{(U Salt W) andc (W Salt U)} \]
\[ \text{U alt W} = \text{(U Salt W) andc (U altS W)} \]
\[ \text{U alt W} = \text{(U altS W) andc (W altS U)} \]

Crosschecks like these boost up our confidence that the logic-table is consistently defined. In fact only a part of the table was defined, and the more difficult rest was derived (see below). This is another source of confidence. The ultimate check is by common sense, which is not so easy for some pairs of U, W. Nevertheless it all started with my practical applications-oriented "wishful thinking". The derivation algorithm and the crosschecks came much later. C'est la vie. On the other hand a formal "glass-bead game" not rooted in practice would be like "Those hieroglyphs once so significant ... silently they vanish in the sand". Recall one of Hermann Hesse's poems in his Nobel Prize winning novel "Das Glasperlenspiel" i.e. in "The Glass Bead Game" alias "Magister Ludi" (= Master of the Game).

Caveat:

The non(U) Xalt non(W); neg(U) Xalt neg(W)
non(U) Salt non(W); neg(U) Salt neg(W)
non(U) altS non(W); neg(U) altS neg(W)

are allowed but should not be casually used instead of ORc, even if U, W have a special relationship (mutually exclusive alternatives; Superset). This is so simply because non(U), non(W) are not mutually exclusive (the same argument holds for Salt, altS).

Quiz: Why non(U) alt non(W); neg(U) alt neg(W) are no problem?
I believe that our set of logical operators will suffice for 99% of rules. If the need would arise for some special operators like, say, an "implication", then most of them can be constructed from non, and, or according to the table in the Appendix. However not every arbitrary operator can be constructed. Only after you have tried to construct "True then", "False then" from the other operators, you will appreciate having them. See their use in the examples below.

HOW IS QUIXPERT's LOGIC DESIGNED:

The black-or-white Boolean logic was "humanized" by:
1. Extending the range of the plausibility values to 3 "defined" values for logic #3, and to 101 values (0 to 100%) for logic #2.
2. New monadic and dyadic logical operators were defined to facilitate formulation, self-checking and evaluation of knowledge-rules.
3. Our conditional evaluation is symmetric, neither left-to-right, nor right-to-left, except for U=?=W, in which case U is asked first.
   Thus if U = ?, W = "defined" there should be no question asked about U, if not necessary (see the table). Implementations may vary from a "shadow" evaluation to a (semi)symbolic optimized evaluation.
   A "shadow" evaluation means that asking questions/reading data bases is blocked, and an attempt is made to get a result = "definitive".
   Note: my current implementation is not perfect in this respect, hence U will be superfluously evaluated if it is a subexpression which is not defined as a separate rule. A hawk-eyed user may notice it iff W contains a yet unanswered question.
   Consider PASCAL's function-parameter mechanisms + strong typing and you will see that it was a "sad fun". However it has generated few useful byproduct-ideas for QuiXpert. Every disadvantage has its advantage (thought the guy who had to arrange for his mother-in-law's funeral; this is a favorite joke of Dr. Viado Stubic).
4. The ERR values are defined so that a semantic error in the formulation of knowledge-rules may be detected at run time. Any user-supplied semantics for self-checking are highly desirable, hence the "strong" OR-operators (with ERR-check) should be preferred over the "weak" OR-operators (without ERR-check), wherever possible.
   ERR reports error, and returns the result = 0.5. It is the user's application dependent responsibility or freedom either to react upon an ERR-message, or let the result=.5 to propagate or to be changed and thus vanish during further inferences.
5. Q: How was the table of logical operators constructed and checked?
   A: Monadic ops-table was defined as a whole.
   Dyadic ops-table was partly defined and the rest derived.
   The results of the first 6 input pairs (U, W) were defined by the "application oriented commonsense" reasoning. The rest was then derived (except for those marked by +, * which were defined too).

An algorithm for a derivation of our logical table:
This algorithm is easy to execute by hand.

Let the "informativeness $i$" of results be: $i(?) < i(0.5) < i(0) = i(1)$.

IF $U = ? = W$ THEN result $=?$ ELSE IF $U = 0.5 = W$ THEN result $=0.5$ ELSE FOR every of the remaining 10 possible input pairs of values $U, W$ DO:
BEGIN
WHILE both $U, W$ are not "definitive" values DO:
BEGIN
For each input $= ?$ substitute 0, 1, .5 separately.
For each input $= .5$ substitute 0, 1, separately.
Each separate substitution yields a "new unique in-pair".
IF a "new unique in-pair" consists of "definitive" values only, THEN we call it a "definitive in-pair".
END;
Look up the result of each "definitive in-pair" in table's first 4 lines.
IF result is ERR THEN [skip it, forget it] ELSE:
IF one input was $?$ THEN the result is $?$ iff the "informativeness" of the result may still increase ELSE:
IF results differ THEN fill the original input (? or .5) with minimal "informativeness" into the table as a result; ELSE:
IF all results are the same THEN fill in that result.
END.

QUIXPERT'S LOGIC IN DETAIL:

Semiotics = Syntax + Semantics + Pragmatics.
Syntax = formal grammar (here very simple).
Semantics = the meaning (I mean whatever that may mean). Here we speak about the meaning of the logical operators, and about the meaning of the operands.
Pragmatics = practical use, hints, prefabricated modules, examples.

LOGICAL OPERATORS IN DETAIL:

The $U, V, W$ are arbitrarily complex logical (sub)expressions, i.e. knowledge-rules or subexpressions within the rules. They can be viewed as (sub)trees to be traversed and evaluated. The syntax shown here includes the fine syntactic sugar specific to the current implementation in QuiXpert.

NON: inventor
Syntax: non($V$)
Semantics: Inverts a logical value $V$ into its complement.
Pragmatics: Used the same way as standard Boolean "not".

NEG: negator
Syntax: neg($V$)
Semantics: If $V = false$ then result := true, else result := false
Pragmatics: For construction of special operations.
YES:
Syntax:  yes(V)
Semantics: If V = true then result := true, else result := false
Pragmatics: For construction of special operations.

MAY:
Syntax:  may(V)
Semantics: If V = maybe then result := true, else result := false
Pragmatics: For construction of special operations.

ANDc:
"and" conditionally evaluated (one of both U, W)
Syntax:  op(U, andc, W)
Semantics: If U, W intersect i.e. semantically overlap, then op := true else op := false;
Pragmatics: Avoids an unnecessary evaluation, i.e. cuts a subtree, thus avoids asking superfluous questions.
Should be used almost always instead of "andu".

ANDu:
"and" unconditionally evaluated (both operands U, W)
Syntax:  op(U, andu, W)
Semantics: See your first computer kids' coloring book!
Pragmatics: If we wish to evaluate both U, W.

ORc:
"or" conditionally evaluated (one of both U, W)
Syntax:  op(U, orc, W)
Semantics: Unification (= "merging") of two semantic classes U, W.
Pragmatics: Avoids an unnecessary evaluation, i.e. cuts a subtree, thus avoids asking of superfluous questions.
Should be used almost always instead of "oru".

ORu:
"or" unconditionally evaluated (both operands U, W)
Syntax:  op(U, oru, W)
Semantics: See your first computer coloring book!
Pragmatics: If we wish to evaluate as many (sub)rules as possible.

ALT:
alternatives
Syntax:  op(U, alt, W)
Semantics: Conditionally evaluated connective for two alternatives which (for an application) are considered as logically equivalent (but not necessarily identical).
Pragmatics: Use "alt" to provide alternative evaluation/reasoning paths in general, and alternative questioning in particular. If the user answers with "maybe" to one question (perhaps because s/he does not understand that formulation), then s/he will get a second chance to answer a different yet equivalent alternative.
Inconsistency checks are the bonus (ERR in the table).
Caveat: ALT is not the "equivalence" operator! See the tables!
XALT:
Syntax: \( \text{op}(U, \text{xalt}, W) \)
Semantics: Conditionally evaluated connective for two mutually exclusive alternatives.
Pragmatics: If the (sub)results/conclusions should be mutually exclusive, then "xalt" should be used instead of the "orc" which is very similar, but does not check the expected mutual exclusivity. XALT will be often handy at the root-rule (root of the whole 'tree'), if any. XALT is better than ORc for expressing the "exception by extension" i.e. the "inclusive unless" (see below).
Caveat: XALT is not the "exclusive OR"! See the tables!

SALT:
superalternative.
Syntax: \( \text{op}(U, \text{Salt}, W) ; \text{also: op}(U, \text{SuperAlt}, W) \)
Semantics: Conditionally evaluated connective to be used if and only if \( U \) is a superalternative for \( W \), i.e. iff \( U \) includes \( W \) but not vice versa. In other words if \( W \) is a subset of \( U \), but not if \( U \) alt \( W \).
Pragmatics: The first operand \( U \) must be more general than \( W \).
Use SALT instead of ALTS if \( U \) is simpler than \( W \). Hence \( U \) will be often a single "global" term, while \( W \) will tend to be a term combined from several special cases. Thus \( U \) may be more difficult to answer, but if \( U \) is answered, there is no need to ask special cases.
Q: What if we use \( U \text{ Salt} \ W \), instead of \( U \text{ alt} \ W \)?
A: If \( U=0.5 \) & \( W=0 \) then result=.5, instead of 0 (like for OR).
If \( U=0 \) & \( W=0 \) then a superfluous question will be asked.
If \( U=1 \) & \( W=0 \) then no ERR-check will be reported.
Hence no catastrophe will occur, while to use ORc would be even worse because \( U=0 \) & \( W=0 \) is a very frequent case (see the table). Other expert systems do not have such operators.

ALTS:
subalternative.
Syntax: \( \text{op}(U, \text{altS}, W) ; \text{also: op}(U, \text{subalt}, W) \)
Semantics: Conditionally evaluated connective to be used if and only if \( U \) is a subalternative for \( W \), i.e. iff \( U \) is included in \( W \) but not vice versa. In other words if \( U \) is a subset of \( W \), but not if \( U \) alt \( W \).
Pragmatics: The first operand \( U \) must be less general than \( W \).
Use ALTS instead of SALT if \( U \) is simpler than \( W \). This is the case when there are only 1 or 2 special cases to ask, while the general term \( W \) is more complex to ask.

TRUETHEN:
positive key precondition
Syntax: \( \text{op}(U, \text{truethen}, W) ; \text{also: op}(U, \text{allows}, W) ; \text{also: op}(U, \text{enables}, W) ; \text{also: op}(U, \text{permits}, W) \)
Semantics: Without \( U=\text{true} \) it makes little sense to evaluate/ask \( W \).
Pragmatics: Note the subtle difference between TRUETHEN and ANDc in the table of operators above (marked by *).
See also the "Practical Patterns" below.
Caveat: \( (U \text{ truethen } W) \land (\text{yes}(U) \land c \ W) \) !
FALSETHEN: inverted key precondition

Syntax: \( \text{op}(U, \text{falsethen}, W); \text{ also: op}(U, \text{disallows}, W); \text{ also: op}(U, \text{blocks}, W) \)

Semantics: \( \text{op}(U, \text{falsethen}, W) = \text{op}(\neg(U), \text{truethen}, W) \)

Pragmatics: Analogical to the pragmatics of "truethen".

Caveat: See also the "Practical Patterns" below.

\((U \text{ falsethen } W) \leftrightarrow (\neg(U) \text{ andc } W) \leftrightarrow (\neg(U) \text{ andc } W)\)

The only difference between \((U \text{ falsethen } W)\) and the \((\neg(U) \text{ andc } W)\) is for \(U = .5 \& W = ?\), in which case the first yields .5 (see *) and the last yields the "?".

Caveat: I have avoided the word "inhibits" just because the word "inhibition" is traditionally used for the certain, less common Boolean operations (see the Appendix) defined as:

\[ \text{X inh Y} = \text{X inhibits Y} = \neg(X) \text{ and } Y \]
\[ \text{X dif Y} = \text{X inhibited by Y} = X \text{ and } \neg(Y) \]

The "dif" is sometimes called "logical difference", and it corresponds with the "exclusive unless": \(U \text{ andc non}(W)\) shown in the last column of the table above.

See also "How to express exceptions?" below.

MAYTHEN: doubtful key precondition

Syntax: \( \text{op}(U, \text{maythen}, W) \)

Semantics: Without \(U=\text{maybe}\) it makes no sense to evaluate/ask \(W\).

Pragmatics: See the "Practical Patterns" below.

Caveat: \((U \text{ maythen } W) = (\text{may}(U) \text{ andc } W) \leftrightarrow (\neg(U) \text{ andc } W)\)

Quiz: Why?

Note that here = holds, but not for TRUETHEN, FALSETHEN which return .5 also for inputs other than \(U=.5=W\). I do not see how MAYTHEN could be even formally symmetrized, because .5 is alone and 0, 1 are two "definitive" values (should 0 or 1 play the formal role of .5; forget it).
PRACTICAL PATTERNS OF KNOWLEDGE-RULES:

0. TRIVIA (if there is anything trivial):
   a) Combine questions iff you never have to identify the individual components. E.g:
      Ask('Is it a whale or a dolphin?').
   b) Combine common subexpressions (occurring within several rules) into separate auxiliary rules.
   c) Use as many of the multiple consequences (THELSEs) as near the root(s) of a (multi)tree as possible. The nearer to the leaves, the less THELSEs will be used. But this is a maxim, not a rule.

1. WHICH OPERAND FIRST?

Several criteria should be considered by a serious knowledge engineer when deciding which of the two terms connected by a commutative conditional operator (or deciding between altS, Salt) should be the first (= U) and which the second (= W). The criteria are:
- Structural simplicity (constructed from less subterms/questions).
- Conceptual simplicity (= clarity = chance for wrong answer 0, 1).
- Chances for answer 0.5 which tend to prolong asking/evaluations.
- Chances for answers 0, 1 (ANDc-like versus ORc-like in reverse).
- Need for a semantically smooth questioning.

Iff an operator is commutative and conditionally evaluated, then the first operand in U op W should be the simpler (sub)expression or rule. This tends to minimize number of questions asked and it "commits arboricide" (= cuts the tree) due to conditional evaluations.

If U, W are comparably complex, then:
- For commutative conditional ANDc-like operators (ANDc only) the first should be the operand which is more frequently "false".
- For commutative conditional ORc-like operators (ORc, Xalt) the first should be the operand which is more frequently "true".

Although Salt, altS are not commutative themselves, having them both we are free to choose either of them and thus to get the same freedom of ordering, because U Salt W = W altS U.
- For ALT makes true/false no difference, but "maybe" prolongs questioning/evaluation (disallows arboricide i.e. tree cutting).

Examples with alt-like operators:
   a) (simple term) ALT (complex alternative)
      E.g:
      ungulate ALT (mammal ANDc (hoofs ALT chews))
   b) (simple superterm) Salt (more of the subcases)
      E.g:
carnivore Salt (pointedteeth ORc claws)

c) (simple subterm) altS (complex superterm)

E.g:

ostrich := emu altS (bird andc verylarge andc verylongneck)

Note that a simple term is often a single question which may not be so simple to answer. Nonspecialists will often answer with "maybe" (=.5 = Don't know) and will be questioned in more specific terms (W).

2. HOW TO EXPRESS EXCEPTIONS?

To say that "90% of all real-life common-sense reasoning is nothing but exceptions handling" is only a slight exaggeration. There are 2 opposite ways how people reason in terms of exceptions. When we say/think "unless" (Dutch "tenzij"), we mean "except if/for" (Dutch "behalve"). That may be meant/used in two ways.

a) "By extension" i.e. "inclusive unless":

Text: A BIRD is an animal which FLIES, unless it is a PENGUINE.
Logic: BIRD := FLIES orc PENGUINE; { no check }
BIRD := FLIES xalt PENGUINE; { ERR-check, hence better }

b) "By exclusion" i.e. "exclusive unless":

Text: ASPIRIN is used if in PAINS, unless your STOMACH IS ILL.
Logic: ASPIRIN := PAINS andc non(STOMACH IS ILL); { or slightly }
ASPIRIN := PAINS andc neg(STOMACH IS ILL). { different }

Note the subtle difference between the two rightmost columns in the table above.

Q: When to use ("unless" with) "non" and when "neg"?
A: The "non" leaves more uncertainty/possibilities, while "neg" leads to more "conservatively rejective", forbidding (pseudo) certainty. The decision-action requirement is what matters.

Hence there are 2 opposite ways how to use/think "unless". Since computers do not have common sense, I consider it too dangerous to introduce an operator like "unless", especially if its equivalents are so simple, straight (after you read this) and safe.

3. BACK- AND FORTH-EVALUATIONS:

In QuiXpert there are no fixed modes of evaluation, although the back-chaining is the most natural way. Back-chaining reaches its conclusion at the very root from which it started to evaluate (by substituting rhs for lhs). Forth-evaluation in QuiXpert reaches conclusion at some leaf i.e. at some terminal term. Hence it is very much like the traditional top-down (= root-up) decision-tree-like process.
The forth-mode is based on the idea of a "necessary (= key) precondition" i.e. on the use of new logical connectives

\[ \text{U TRUETHEN } W ; \quad \text{U FALSETHEN } W ; \quad \text{U MAYTHEN } W. \]

where U is the "key" for "entering"/asking the W.

See the Examples below where the whole Wasmachine works in forth-mode. Yet QuiXpert allows to mix back- and forth-inferencing arbitrarily, i.e. per rule i.e. per subtree.

4. SELECTING A SUBTREE:

Any logical (sub)expression may be viewed as a decision (sub)tree. The selection of a suitable branch is done according to a general pattern:

\[ \ldots ((Q \text{ falsethen } L) \text{ xalt } (Q \text{ maythen } M) \text{ xalt } (Q \text{ truethen } R)) \ldots \]

where any pair of L, M, R may either be different or the same subtree. If Q is a separate rule (not just a nameless common subexpression), then Q will be evaluated only once. Note that

\[ \ldots ((\text{non}(Q) \text{ andc } L) \text{ orc } (Q \text{ andc } R)) \ldots \]

is not a solution, as it works like:

\[
\begin{align*}
\text{if } Q=0 & \text{ then result := L else} \\
\text{if } Q=1 & \text{ then result := R else} \\
{\text{if } Q=.5 & \text{ then } \text{ if } L=0 \& R=0 \text{ then result := 0} \\
& \text{ else result := .5};}
\end{align*}
\]

Hence if \( Q = "\text{maybe}" \) then:
- the result will never become "true", despite of that
- both L, R are always evaluated (if \( Q = "\text{maybe}" \))!

Recall:

\[
\begin{align*}
\text{neg}(U) \text{ falsethen } W & \quad \text{ works like: if } U \diamond 0 \text{ then ask } W, \\
\text{yes}(U) \text{ falsethen } W & \quad \text{ works like: if } U \diamond 1 \text{ then ask } W.
\end{align*}
\]

5. MULTIPLE HYPOTHESES AND "NO IDEA" HYPOTHESIS:

Rules form a bunch of logical assignments like \( \text{lhs:=rhs} \). Those RULES WHOSE LHS DOES NOT APPEAR ON THE RHS OF ANY OTHER RULE ARE THE ROOTS in a multitree of rules. Each root is a hypothesis which may be investigated/evaluated apart. Other rules may be hypotheses too, but a root must be an accessible one. Often it is desirable to be able to:
- Evaluate more than one/all hypotheses at once. Why? Because:
  - If the hypotheses are not mutually exclusive, we want all the "true" ones. E.g. a patient (man, animal, plant, machine, organization) may suffer several illnesses simultaneously.
  - If they are mutually exclusive, then we should (be able to) test whether they are mutually exclusive. See the Appendix.
  - In general it is more efficient for both man and machine to evaluate more/all hypotheses at once, especially if they are interconnected. However recall that QuiXpert does allow to restart and to reuse the earlier answers.
To be allowed not to ask for any specific hypothesis, if we have no idea/no suspicion about the possible conclusion. There is a simple solution to the above tasks. We just connect all the root-rules into one single all-encompassing root. If the hypotheses are mutually exclusive, then we connect them with XALT (or ORc), otherwise we connect them with unconditional ORu. The ORu is also recommended for testing of the mutual exclusiveness (see the Appendix). For example:

ROOT := Hypo1 Xalt Hypo2 Xalt Hypo3. { or with ORc or ORu }
or:
triangle := rightangled ORu (scalene Xalt (isoceles Xalt equilateral))

Q: Why the ORu here? A: We want all the properties.

Such a solution is based on a unifying view, hence clean & simple. It does not require any special mechanisms. Of course it holds for any other rule (= subtree) as well, but although "all rules are equally important, but some (like roots) are more important than the other".
NON-BEAUTIFIED EXAMPLES OF KNOWLEDGE-RULES IN QUIXPERT:

Watch the logic and the semantics, not the syntactic sugar.
Rules are compiled with the rest of Quixpert. This all is PASCAL.

{---- The rules below are not optimal in any respect. There are just too many ways how to formulate rules and many criteria of optimality which themselves are context (including the user, the use, the microworld) dependent. What counts is correctness and avoidance of superfluous (hence silly) questions.
}

{---- Triangles: ---- ( rules must precede their "callers" ) ******

function BIGS: real; begin RULE(18);
BIGS:=Ask("Is one of the sides C = A+B ?");
show("Big side C = A+B exists ?");
end;
function ZERO: real; begin RULE(19);
ZERO:=Ask("Is some side of zero length ?");
show("Some side has zero length ?");
end;
function DIP3: real; begin RULE(20);
DIP3:=Ask("Are all 3 sides of different length ?");
ThenFalse(22){EQV3}; show("All sides are different ?");
end;
function PYTH: real; begin RULE(21):
PYTH:=Ask("Does it hold A*A+B*B=C*C for A < C > B ?");
ThenFalse(23){GR90}; ThenFalse(24){AL60};
show("A*A+B*B=C*C for A < C > B ?");
end;
function EQV3: real; begin RULE(22);
EQV3:=Ask("Is it true that A=B=C ?");
ThenFalse(16){RIGH}; ThenFalse(13){SCAL}; ThenTrue(22){EQV3};
ThenFalse(20){DIP3}; ThenFalse(14){ISOC}; ThenTrue(24){AL60};
show("A=B=C ?");
end;
function GR90: real; begin RULE(23);
GR90:=Ask("Is one of the angles GREATER than 90 degrees ?");
ThenFalse(16){RIGH}; ThenFalse(15){EQUI};
show("Angle > 90 ?");
end;
function EQ90: real; begin RULE(26);
EQ90:=Ask("Is one of the angles EQUAL EXACTLY 90 degrees? ?");
ThenFalse(15){EQUI}; show("Angle = 90 ?");
end;
function AL60: real; begin RULE(24);
AL60:=Ask("Are ALL angles equal EXACTLY 60 degrees? ?");
ThenFalse(16){RIGH}; ThenFalse(13){SCAL}; ThenTrue(22){EQV3};
ThenFalse(20){DIP3}; ThenFalse(14){ISOC}; show("3*60 deg ?");
end;
function XL59: real; begin RULE(25);
XL59:=Ask("Has the LARGEST angle LESS than 60 degrees ?");
show("Largest angle < 60 degrees ?");
end;
function ATRI: real; begin RULE(17);
  ATRI:=Infer(op(non(XL59),andc,op(non(ZERO),andc,non(BIGS))));
  ElseFalse(15){EQUI}; ElseFalse(13){SCAL}; ElseFalse(14){ISOC};
  ElseFalse(16){RIGH};
  show('A 3-angle');
end;

function EQUI: real; begin RULE(15);
  EQUI:=Infer(op(ATRI, TrueThen, op(EQV3, alt, AL60)));
  ThenFalse(23){GR90}; ThenFalse(20){DIF3}; ThenFalse(16){RIGH};
  show('Equilateral');
end;

function SCAL: real; begin RULE(13);
  SCAL:=Infer(op(ATRI, TrueThen, DIF3));
  ThenFalse(15){EQUI}; ThenFalse(14){ISOC}; ThenFalse(22){EQV3};
  show('Scalene');
end;

function ISOC: real; begin RULE(14);
  ISOC:=Infer(op(ATRI, TrueThen, op(non(SCAL), andc, non(EQUI))));
  ThenFalse(20){DIF3};
  show('Isosceles');
end;

function RIGH: real; begin RULE(16);
  RIGH:=Infer(op(ATRI, TrueThen, op(non(GR90), andc, op(PYTH, alt, EQ90))));
  ThenFalse(22){EQV3};
  ThenFalse(24){AL60};
  show('Right angled');
end;

{ TRIA is the root }
function TRIA: real; begin RULE(98);
  TRIA:=Infer(op(RIGH, oru, op(SCAL, alt, op(ISOC, alt, EQUI))));
  show('It is an identifiable triangle');
end;
{ WASMACHINE:
    ---- Forth-evaluation = root-up i.e. semantic top-down Ask-ing: ------
}

function qB: real; begin RULE(27);
    qB:=Ask("qB: Oorzaak: Er zit een knik in de afvoerslang?");
    show("qB?");
end;

function qC: real; begin RULE(28);
    qC:=Ask("qC: Oorzaak: Pomp is (gedeeltelijk) verstopt?");
    show("qC?");
end;

function qE: real; begin RULE(29);
    qE:=Ask("qE: Oorzaak: Pompdeksel niet goed bevestigd?");
    show("qE?");
end;

function qF: real; begin RULE(30);
    qF:=Ask("qF: Oorzaak: Watertoevoerslang zit niet aan kraan of machine vast?");
    show("qF?");
end;

function qG: real; begin RULE(31);
    qG:=Ask("qG: Oorzaak: Afvoerpijp verstopt?");
    show("qG?");
end;

function qI: real; begin RULE(32);
    qI:=Ask("qI: Oorzaak: Pomp gedeeltelijk verstopt?");
    show("qI?");
end;

function qA1: real; begin RULE(133);
    qA1:=Ask("qA1: Er blijft water in de machine?");
    show("qA1?");
end;

function qD: real; begin RULE(134);
    qD:=Ask("qD: Is er water op de vloer onder de machine?");
    show("qD?");
end;

function qH: real; begin RULE(135);
    qH:=Ask("qH: Duurt het te lang voordat het centrifugeren begint?");
    show("qH?");
end;

function qJ: real; begin RULE(136);
    Say("qJ: Machine maakt lawaai en schudt erg tijdens centrifugeren");
    qJ:=Ask("qJ: en wasgoed is erg nat na het centrifugeren?");
    show("qJ?");
end;

function qK: real; begin RULE(137);
    qK:=Ask("qK: Water in of onder de machine?");
    show("qK?");
end;

function qL: real; begin RULE(138);
    qL:=Ask("qL: Problemen centrifugeren?"); show("qL?");
end;

function cA: real; begin RULE(33);
    cA:=Infer(op(qA1, l'orue1'hen, op(qB, orc,qC));
    show("cA: Pomp/afvoerslang verstopt");
end;
function cD: real; begin RULE(34);
    cD:=Infer(op(qD, TrueThen, op(op(qE,orc,qF),orc,qG)));
    show("cD: Pompeksel/toe-/af-voer fout:");
end;
function cH: real; begin RULE(35);
    cH:=Infer(op(qH, TrueThen, op(qI,orc,qB))); show("cH: Pomp/afvoerslang verstopt");
end;
function cJ: real; begin RULE(36);
    cJ:=Infer(op(qJ, TrueThen, op(qC,orc,qB))); show("cJ: Pomp/afvoerslang verstopt");
end;
function cK: real; begin RULE(37);
    cK:=Infer(op(qK, TrueThen, op(cA,orc,cD))); show("cK: Waterprobleem");
end;
function cL: real; begin RULE(38);
    cL:=Infer(op(qL, TrueThen, op(cH,orc,cJ))); show("cL: Centrifugeprobleem");
end;
function WAS: real; begin RULE(39);
    WAS:=Infer(op(cK,orc,cL)); show("Wasmachine diagnosticeerbaar");
end;
{
{ ZOO XPERT ------------------------------------
We use ThenFalse, ThenTrue; Duda & Gashnick, Winston does-/can-not.
We have ElseFalse, ElseTrue: e.g: If FLIE=false then FLWE:=false;
QuiXpert's ThenFalse, ElseFalse, ElseTrue (but not ThenTrue) are
negations not available in PROLOG.
Use ThenFalse, ThenTrue, ElseFalse, ElseTrue as near the root as
possible (for economy of Ask-ing & Infer-ing).
Here we define non-trivial beasts too, not just dogs, cats and cows.
Try to add fish, snakes; butterflies, ••• angels.
} {
function HAIR: real; begin RULE(40);
    HAIR:=Ask("Has the animal hair?"); show("hairy?");
end;
function MILK: real; begin RULE(41);
    MILK:=Ask("Gives the female milk?"); show("milky?");
end;
function qMAM: real; begin RULE(142); { optional general question }
    qMAM:=Ask("Is it a mammal?"); show("mammal?");
end;
function DOLP: real; begin RULE(74);
    DOLP:=Ask("Is it a dolphin?"); show("dolphin?");
end;
function WHAL: real; begin RULE(75);
    WHAL:=Ask("Is it a whale?"); show("whale?");
end;
function PORP: real; begin RULE(76);
    PORP:=Ask("Is it a porpoise?"); show("porpoise?");
end;
function qCET: real; begin RULE(77);
qCET:=Ask('Is it a cetaceous animal?'); show('cetaceous?');
end;

function CETA: real; begin RULE(78);{ no Salt iff these cover all }
CETA:=Infer(op(op(qCET,alt,PORP), alt ,op(WHAL, xalt, DOLP));
ThenFalse(73){qINS}; ThenFalse(46){BIRD}; ThenTrue(42){MAMM};
ThenFalse(83){qFIS}; ThenFalse(82){qREP}; show('cetacean');
end;

function qREP: real; begin RULE(82);
qREP:=Ask('Is it a reptile?'); show('reptile?');
ThenFalse(46){BIRD}; ThenFalse(83){qFIS};
ThenFalse(78){CETA}; ThenFalse(73){qINS}; ThenFalse(42){MAMM};
end;

function qFIS: real; begin RULE(83);
qFIS:=Ask('Is it a fish?'); show('fish?');
ThenFalse(46){BIRD}; ThenFalse(82){qREP};
ThenFalse(78){CETA}; ThenFalse(73){qINS};
end;

function qBAT: real; begin RULE(175);
qBAT:=Ask('Is it a bat?'); show('bat?');
ThenTrue(42){MAMM}; ThenFalse(46){BIRD};
end;

function MAMM: real; begin RULE(42);
MAMM:=Infer(op(op(qMAM, alt, MILK), Salt, op(op(HAIR, orC, qBAT),
Xalt, CETA)));
ThenFalse(73){qINS}; ThenFalse(46){BIRD};
ThenFalse(82){qREP}; ThenFalse(83){qFIS};
ElseFalse(62){UNGU}; ElseFalse(87){MONO};
ElseFalse(78){CETA}; show('Mammal');
end;

function FEAT: real; begin RULE(43);
FEAT:=Ask('Has the animal feathers?'); show('feathers?');
end;

function FLIE: real; begin RULE(44);
FLIE:=Ask('Can the animal fly?'); show('flies?');
ElseFalse(61){FLWE};
end;

function EGGS: real; begin RULE(45);
EGGS:=Ask('Does the animal lay eggs?'); show('eggs?');
end;

function qBIR: real; begin RULE(145);{ optional }
qBIR:=Ask('Is it a bird?'); show('bird?');
end;

function qINS: real; begin RULE(73);
qINS:=Ask('Is it an insect?'); show('insect?');
ThenFalse(42){MAMM}; ThenFalse(46){BIRD};
end;

function BIRD: real; begin RULE(46);
BIRD:=Infer(op(qBIR, alt, op(FEAT, alt, op(non(op(qINS, xalt, qBAT))))),
ThenFalse(42){MAMM}; ThenFalse(73){qINS}; show('Bird');
ThenFalse(82){qREP}; ThenFalse(83){qFIS};
{ Winston’s LISP, Duda & Gashnick poor; they use FLIE and CANT_FLIE. }
end;

function MEAT: real; begin RULE(47);
MEAT:=Ask("Does the animal eat meat?"); show('meateater?');
end;

function PTEE: real; begin RULE(48);
PTEE:=Ask("Has it pointed teeth?"); show('pointed teeth?');
end;

function CLAW: real; begin RULE(49);
CLAW:=Ask("Has it claws?"); show('claws?');
end;

function FEYE: real; begin RULE(50);
FEYE:=Ask("Has it forward eyes?"); show('forward eyes?');
end;

function qCAR: real; begin RULE(151); { optional general question }
qCAR:=Ask("Is it a carnivore?"); show('carnivore?');
end;

function CARN: real; begin RULE(51); { could be MAMM / BIRD / FISH }
CARN:=Infer(op(qCAR,alt.op(MEAT, Salt, op(PTEE,orc,
op(CLAW, andc, FEYE)))));
ThenFalse(62){UNGU}; show('Carnivore');

{ Q: Why "Salt" instead of "alt"?
A: If answered MEAT=True and later in SEXY answered PTEE=False,
then alt would cause CONTRA-dictions because PTEE & CLAW & FEYE imply MEAT but not always vice versa !!!
MEAT may then ... would yield CARN=False if MEAT=True, but neg(MEAT) false then ... should be ok }
end;

function HOOF: real; begin RULE(52);
HOOF:=Ask("Has it hoofs?"); show('hoofs?');
end;

function CHEW: real; begin RULE(53);
CHEW:=Ask("Does it chew?"); show('chews?');
end;

function TAWN: real; begin RULE(54);
TAWN:=Ask("Has it tawny color?"); show('tawny?');
end;

function DARK: real; begin RULE(55);
DARK:=Ask("Has it dark spots?"); show('dark spots?');
ThenFalse(56){BSTR};

function BSTR: real; begin RULE(56);
BSTR:=Ask("Has it black stripes?"); show('black stripes?');
ThenFalse(55){DARK};

end;

function NECK: real; begin RULE(57);
NECK:=Ask("Has it long neck?"); show('long-necked?');
ThenFalse(69){ALBA};
end;

function LEGS: real; begin RULE(58):
LEGS:=Ask("Has it long legs?"); show("long-legged?");

function BLWH: real; begin RULE(59);
BLWH:=Ask("Is it black & white?"); show("black & white?");
end;

function SWIM: real; begin RULE(60);
SWIM:=Ask("Can it swim?"); show("swims?");
ThenFalse(69){ALBA};
end;

function FLWE: real; begin RULE(61);
FLWE:=Ask("Can it fly well?"); show("flies well?");
ThenTrue(44){FLIE};
end;

function qUNG: real; begin RULE(162);
qUNG:=Ask("Is it an ungulate?"); show("ungulate?");
end;

function UNGU: real; begin RULE(62);
UNGU:=Infer(op(qUNG,alt,op(MAMM, and, op(HOOF, alt, CHEW)))); show("Ungulate"); ThenFalse(51){CARN};
end;

function CHEE: real; begin RULE(63);
CHEE:=Infer(op(MAMM, and, op(CARN, and, op(TAWN, and, DARK)))); show("Cheetah");
end;

function TIGR: real; begin RULE(64);
TIGR:=Infer(op(MAMM, and, op(CARN, and, op(TAWN, and, BSTR)))); show("Tiger");
end;

function GIRA: real; begin RULE(65);
GIRA:=Infer(op(UNGU, and, op(NECK, and, op(LEGS, and, DARK)))); show("Giraffe");
end;

function ZEBR: real; begin RULE(66);
ZEBR:=Infer(op(UNGU, and, BSTR)); show("Zebra");
end;

function OSTR: real; begin RULE(67);
OSTR:=Infer(op(BIRD, and, op(NECK, and, op(BLWH, and, non(FLIE))))); show("Ostrich");
end;

function PENG: real; begin RULE(68);
PENG:=Infer(op(BIRD, and, op(BLWH, and, non(FLIE)))); show("Penguin");
end;

function ALBA: real; begin RULE(69); { Not in original: }
ALBA:=Infer(op(BIRD, and, op(SWIM, and, non(FLIE)))); { & no SWIM & no NECK } show("Albatros"); { see SWIM, NECK }
end;

function FEMA: real; begin RULE(70);
FEMA:=Ask("Is it female?"); show("female?");
end;

function NAIL: real; begin RULE(79);
NAIL:=Ask("Has it highly polished nails?"); show("nails?");
end;
function KINI: real; begin RULE(80);
  KINI:=Ask("Has she bikini or nokini?"); show("kini?");
end;

function DRES: real; begin RULE(81);
  DRES:=Ask("Is she well dressed?"); show("well dressed?");
end;

function SEXY: real; begin RULE(71); { not on menu, a surprise }
  SEXY:=Infer(op(MAMM, andc, op(CARN, andc, op(non(PTEE), andc, op(FEMA, TrueThen, op(op(LEGS, orc, KINI), orc, op(DRES, andc, NECK)))))));
  show("Sexy non-vegetarian lady");
end;

function MCVP: real; begin RULE(72);
  MCVP:=Infer(op(MAMM, andc, op(non(FEMA), TrueThen, non(CARN))));
  show("Male chauvinist vegetarian Pig");
end;

function SMIL: real; begin RULE(84);
  SMIL:=Ask("Is the animal smiling?"); show("smiling?");
end;

function STEK: real; begin RULE(85);
  STEK:=Ask("Heeft het stekels?"); show("stekels?");
end;

function qMON: real; begin RULE(86);
  qMON:=Ask("Behoort tot Monotremata?"); show("monotremata?");
  { makes no sense, within MONO: } ThenTrue(87){MONO}; ElseFalse(87){MONO};
  { makes sense These in MAMM to avoid CONTRA-diction on alt in PLAT } end;

function MONO: real; begin RULE(87);
  MONO:=Infer(op(qMON, alt, op(MAMM, andc, EGGS)));
  show("Monotremata");
end;

function PLAT: real; begin RULE(88);
  PLAT:=Infer(op(op(MONO, andc, non(STEK)), andc, SMIL));
  show("Smiling platypus (duckbill) = lachende vogelbekdier");
end;

function MEGL: real; begin RULE(89);
  MEGL:=Infer(op(MONO, andc, STEK));
  show("Tachyglossus=mierenegel or Zaglossus bruijni=vachtegel");
end;

function ANIM: real; begin RULE(99); { an optional root-rule/tree: }
  ANIM:=Infer(op(CHEE, xalt, op(TIGR, xalt, op(GIRA, xalt, op(ZEBR, xalt,
      ,op(OSTR, xalt, op(PENG, xalt, op(ALBA, xalt, op(SEXY, xalt, op(MCVP, xalt,
      ,op(PLAT, xalt, MEGL))))))));
  show("It is an (identifiable) animal");
end;
APPENDIX:

Back to Boole (1854) - or - The Magick of Logick

0 = false
1 = true

Arithmetic representation and evaluation is useful for probabilities and plausibilistic logic.

0101 X X op Y
0011 Y

Names of functions of 2 Boolean variables X, Y suggest their use in retrieval, expert systems.

<table>
<thead>
<tr>
<th>f#</th>
<th>X op Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Total negation; None function; Zero function.</td>
</tr>
<tr>
<td>0001</td>
<td>X and Y</td>
</tr>
<tr>
<td>0010</td>
<td>X inh Y</td>
</tr>
<tr>
<td>0011</td>
<td>Affirmation of Y; assertion of Y.</td>
</tr>
<tr>
<td>0100</td>
<td>X dif Y</td>
</tr>
<tr>
<td>0101</td>
<td>Affirmation of X; assertion of X.</td>
</tr>
<tr>
<td>0110</td>
<td>X xor Y</td>
</tr>
<tr>
<td>0111</td>
<td>X or Y</td>
</tr>
<tr>
<td>1000</td>
<td>X nor Y</td>
</tr>
<tr>
<td>1001</td>
<td>X eqv Y</td>
</tr>
<tr>
<td>1010</td>
<td>X imp Y</td>
</tr>
<tr>
<td>1011</td>
<td>X not X</td>
</tr>
<tr>
<td>1100</td>
<td>X imp X</td>
</tr>
<tr>
<td>1101</td>
<td>X nand Y</td>
</tr>
<tr>
<td>1110</td>
<td>Total affirmation; All function; One function.</td>
</tr>
</tbody>
</table>

X op Y: f 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

<table>
<thead>
<tr>
<th>X op Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1</td>
</tr>
</tbody>
</table>

c means commutative (see this & this line)
Some of the "confusing" operators:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>f11</td>
<td>f13</td>
<td>f4</td>
<td>f2</td>
<td>f14</td>
<td>f8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X -&gt; Y</td>
<td>Y -&gt; X</td>
<td>Y inh X</td>
<td>X inh Y</td>
<td>X nand Y</td>
<td>X nor Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X imp Y</td>
<td>Y imp X</td>
<td>X dif Y</td>
<td>Y dif X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X op Y</td>
<td>X &lt;= Y</td>
<td>Y &lt;= X</td>
<td>absent in Pascal</td>
<td>absent in Pascal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0 0 0 1</td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 0 1 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1 0 1 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 0 0 0 0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constant functions are: f0, f15.

Functions of 1 variable only are: f3, f5, f10, f12.

Hence nontrivial are: f1, f2, f4, f6, f7, f8, f9, f11, f13, f14.

Non-commutative are: f2, f4 (differences alias inhibitions) and f11, f13 (implications).

Pairs of mutual negations: f # = not f 15-#, f 15-# = not f #
1hs = not(rhs), not(lhs) = rhs
f 0 = not f15 total negation = not(total affirmation)
f 1 = not f14 X and Y = not(X nand Y)
f 2 = not f13 Y dif X = not(Y <= X) = X inh Y
f 3 = not f12 affirmation of Y = not(negation of Y)
f 4 = not f11 X dif Y = not(X <= Y) = Y inh X
f 5 = not f10 affirmation of X = not(negation of X)
f 6 = not f9 X xor Y = not(X xnor Y) = X <> Y = not(X=Y)
f 7 = not f8 X or Y = not(X nor Y)
HOW TO CONSTRUCT UNAVAILABLE OPERATORS:

<table>
<thead>
<tr>
<th>f#</th>
<th>op</th>
<th>Boolean expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>X and not X</td>
</tr>
<tr>
<td>1</td>
<td>X and Y</td>
<td>not(not X or not Y)</td>
</tr>
<tr>
<td>2</td>
<td>X inh Y</td>
<td>not X and not Y; not(X &lt;= Y)</td>
</tr>
<tr>
<td></td>
<td>Y dif X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Y inh X</td>
<td>X and not Y; not(Y &lt;= X)</td>
</tr>
<tr>
<td></td>
<td>X dif Y</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X &lt;&gt; Y</td>
<td>(X and not Y) or (not X and Y)</td>
</tr>
<tr>
<td></td>
<td>X neqv Y</td>
<td>(not X or Y) and (X or not Y)</td>
</tr>
<tr>
<td></td>
<td>not(X or Y)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>X or Y</td>
<td>not(not X and not Y)</td>
</tr>
<tr>
<td>8</td>
<td>X nor Y</td>
<td>(not X and not Y); not(X or Y)</td>
</tr>
<tr>
<td>9</td>
<td>X eqv Y</td>
<td>(X and not Y) or (not X and not Y)</td>
</tr>
<tr>
<td></td>
<td>(not X or Y) = (Y imp X) and (X imp Y)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>not X</td>
</tr>
<tr>
<td>11</td>
<td>X &lt;= Y</td>
<td>not X or Y ; not(Y dif X)</td>
</tr>
<tr>
<td>12</td>
<td>-</td>
<td>not Y</td>
</tr>
<tr>
<td>13</td>
<td>Y &lt;= X</td>
<td>not Y or X ; not(X dif Y)</td>
</tr>
<tr>
<td>14</td>
<td>X nand Y</td>
<td>not(X and Y); not X or not Y</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>X or not X ; Y or not Y</td>
</tr>
</tbody>
</table>

DUALITY OF RULES:

Interchange: 0 with 1, and also "and" with "or", and let the "not" and the parentheses unchanged.

Duality rule: Every valid rule has its valid dual rule.

\[
\begin{align*}
0 \text{ and } 1 &= 0 \quad := \quad 1 \text{ or } 0 = 1 \\
(A \text{ or not } B) \text{ and } B &= A \text{ and } B \quad := \quad (A \text{ and not } B) \text{ or } B = A \text{ or } B \\
(A \text{ or } B) \text{ and not } A &= A \text{ and } B \quad := \quad (A \text{ and } B) \text{ or } \text{not } A = A \text{ or } B \\
(\text{not } A \text{ or not } B) &= \text{not}(A \text{ and } B) \quad := \quad (\text{not } A \text{ and not } B) = \text{not}(A \text{ or } B)
\end{align*}
\]

Hawkeyed pattern recognizes have already noted that those functions which are mutual negations have basic Boolean formulas which are "mirror images" of each other according to this duality-rule. Since they are mutual negations it holds:

\[
f# = \text{not}(f 15-#); \quad \text{and vice versa}
\]

Examples of the use of DeMorgan's laws:

\[
\begin{align*}
X \text{ and } Y &= \text{not}(\text{not}(X \text{ and } Y)) = \text{not}(\text{not}(X) \text{ or not}(Y)) \\
X \text{ inh } Y &= \text{not}(\text{not}(Y) \text{ or } X) = \text{not}(\text{not}(Y \text{ and not}(X))) \\
&= Y \text{ and not } X
\end{align*}
\]
TAUTOLOGIES:

Boolean expressions which are always true (hence their negations are always false) are called tautologies. A tautological rule in an expert system signals a likely trouble. We should check for them. Examples of tautologies:

\[
\begin{align*}
& \text{A or true} \\
& \text{not(A and false)} \\
& ((P \text{ imp } Q) \text{ and } P) \text{ imp } Q \\
& ((P \text{ imp } Q) \text{ and not Q}) \text{ imp not Q} \\
& \text{lhs = rhs}
\end{align*}
\]

\{ \text{ modus ponens } \}
\{ \text{ modus tolens } \}
\{ \text{ iff a valid rule } \}

Seeing is believing:

\[
\begin{array}{cccccc}
 & \text{Modus ponens} & \text{!} & \text{Modus tolens} & \text{!} & \text{rhs} \\
\text{P Q: } & ((P \rightarrow Q) \text{ and } P) \rightarrow Q & & ((P \rightarrow Q) \text{ and not Q}) \rightarrow \text{not Q} & & \\
0 0 & 1 & 0 & 1 & ! & 1 & 1 & 1 \\
0 1 & 1 & 0 & 1 & ! & 1 & 0 & 1 \\
1 0 & 0 & 0 & 1 & ! & 0 & 0 & 1 \\
1 1 & 1 & 1 & 1 & ! & 1 & 0 & 1 \\
\end{array}
\]

EQUIVALENCE CHECKING À LA QUINE

The basic idea is simple. If two expressions (or circuits) E1, E2 are equivalent, then E1 eqv E2 must be a tautology. W.V.O. Quine [1] has described a semi-symbolic reduction method of equivalence checking which is much faster than any brute-force checking.
MISCELLANEA:

QuixNote # 2/JH

GETTING EXPERT'S RULES RIGHT - MUTUAL EXCLUSIVENESS

Motto: Give me a fruitful error any time, full of seeds, bursting with its own corrections. (Vilfredo Pareto)

$$$$$$ Rules debugging algorithm for knowledge engineers $$$$$$$

Note: ok means a good set of rules; ko means bad rules.

AGAIN: (Fresh restart)

Make a reasonable set of rules with a MULTI-root menu.

Switch to Boolean answering mode (#1, not plausibilistic 2);

REPEAT

Select the next hypothesis on the menu;

Answer (with 0 or 1) so that the hypo will be true (=1);

FOR all hyps on the menu DO

begin Select the next hypo on the menu;

if a question is asked then

begin if hyps are MUTUALLY EXCLUSIVE then { ko }

begin Insert some THENTRUE(), THENFALSE(),

ELSETRUE(), ELSEFALSE() into

as low placed rules (= near a root) as reasonable;

GOTO AGAIN;

end else ok:

end else NOTHING asked

if hyps are NOT mutually exclusive then { ko }

begin Delete some THENTRUE(), THENFALSE(),

ELSETRUE(), ELSEFALSE() from

as low placed rules (= near a root) as reasonable;

( if there is nothing to Delete then

there is something very wrong with rules )

GOTO AGAIN;

end else ok;

Command: Go on & Reuse answers (#1, no Fresh restart);

end

UNTIL ok OR (ko AND (wo)man exhausted);

if ko then

begin The set of rules needs fundamental revision; GOTO AGAIN

end:

$$$$$$
PRECHECKING THE "SEMANTIC" CONSISTENCY OF EXPERT'S RULES

Motto: Murphstadter's law: Whatever may go wrong it will, even if you take into account Murphstadter's law.

(Murphy's and Hofstadter's laws amalgamated by Hajek)

Although QuiXpert does check almost everything at run time, and all formal syntax plus cyclic references (recursion) among rules at compile time, some "semantic" checks can be done by precompiler. Such checks would avoid most of the in-flight emergencies.

What could be checked at precompile time? A lot. Here are some hints which form a labeled graph.

Check0: Rule-less is a poor knowledge base indeed. Check it.

Check1: SELF-THELSE
A rule must not THENSELSE itself (courtesy Hans Blom).

Check2: DANGLING RULES
A rule disconnected (neither called nor THENSELSED) from the rest is a "dangler". It must be either deleted or changed into a comment. Note that this bug is only a very special case of non-GROUNDEDness (see Check 6).

Check3: MULTI-THELSE
A rule X must not contain more than 1 THENSELSE of one and the same rule Y.

This prevents clashes, nonsense or harmless redundancies.

Intermezzo: Feasible propagation of consequences via THENSELSEs:
Here the 4 leading mins stand for either THEN or ELSE;
the 3 trailing dots stand for either TRUE or FALSE.

Chains: ----TrueR1: R1: Then...
        ----FalsR2: R2: Else...

Cuts:    ----TrueR3: R3: Else...
        ----FalsR4: R4: Then...

Check4: DIRECT CLASHES between a pair of rules which mutually THENSELSE each other directly.

iff rule X contains:
iff rule Y contains:

<table>
<thead>
<tr>
<th>ThenTrueX</th>
<th>ThenFalseX</th>
<th>ElseTrueX</th>
<th>ElseFalseX</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycle</td>
<td>CLASH</td>
<td>I</td>
<td>CLASH</td>
</tr>
<tr>
<td>CLASH</td>
<td>I cut</td>
<td>cycle I</td>
<td>CLASH</td>
</tr>
<tr>
<td>CLASH</td>
<td>I cycle</td>
<td>I cut</td>
<td>CLASH</td>
</tr>
<tr>
<td>cut</td>
<td>CLASH</td>
<td>I cut</td>
<td>CLASH</td>
</tr>
</tbody>
</table>
CLASHes (True contra False) are UNIdirectional (i.e. cannot start at both X,Y). This 1-directionality is due to the above mentioned chain/cut asymmetry (see the intermezzo above).

Cycles cause no clash of True contra False, but must be broken by implementation of the THELSE-Fire-Once-Only rule-of-thumb.

Cuts are self-disconnecting in both directions, hence safe.

Implementation: in a Rules by Rules matrix there must not be a Rule(j,k)=value1 and Rule(k,j)=value2 such that value1 \not= value2, iff one value is True and other is False.

No value against value is allowed (e.g. IF flies well THEN flies; but not vice versa). Nevertheless a value against no value in such a rules-matrix is worth looking at and possibly adding The Same THELSE across the diagonal so that more complete set of THELSEs will be obtained.

Wrong value1 = value2 across the diagonal cannot be checked directly, but they may be detected as indirect clashes (below).

Check5: INDIRECT CLASHES (derived from the propagation chain/cut and from the rationale behind direct clashes).

Transitions from rule X via rule Y to rule Z etc. possibly back to X rule:

<table>
<thead>
<tr>
<th>Rule X contains:</th>
<th>Rule Y contains:</th>
</tr>
</thead>
<tbody>
<tr>
<td>II ThenTrueZ</td>
<td>II ThenFalseZ I ElseTrueZ I ElseFalseZ</td>
</tr>
<tr>
<td>ThenTrueY II check</td>
<td>ThenFalseZ I ElseTrueZ I ElseFalseZ</td>
</tr>
<tr>
<td>ThenFalseY II cut</td>
<td>ThenFalseZ I ElseTrueZ I ElseFalseZ</td>
</tr>
<tr>
<td>ElseTrueY II check</td>
<td>ThenFalseZ I ElseTrueZ I ElseFalseZ</td>
</tr>
<tr>
<td>ElseFalseY II cut</td>
<td>ThenFalseZ I ElseTrueZ I ElseFalseZ</td>
</tr>
</tbody>
</table>

Legend: check means that a path must be checked further until either a CLASH or cut occurs (see Check4 above).

cut means that a path needs not to be checked further. No clash can occur anymore. This is a kind of conditional symbolic evaluation.

Example of 1 path (as a part of a multi-tree):
R1(ThenTrueR2); R2(ThenFalseR3); R3(ElseFalseR4); R4(ElseFalseR5); R5(ElseTrueR6); R6(Else... cut!

Check6: GROUNDEDness of rules in terminals.

All rules must be "totally grounded" i.e. (in)directly connected (via calls to other rules, not just via THELSEs) to "terminals" only.

A terminal is a primitive rule: an a priori FACT, a question (hopefully an answerable one), or a computable COM-rule.

By "totally" grounded I want to make clear that all forks from a rule must lead to leaves, i.e. not just some paths! All forks because the values might be such that both forking paths are needed.
This may be too harsh for an ALT-operator, but it cannot harm.

Example: a rule like \( R_1(R_2 \& R_3) \) is not totally grounded if even only one of its arguments \( R_2, R_3 \) is not totally grounded.

Q: How to check this?
A: TOPOLOGICAL SORT will check that:

1. There are no cyclic calls (of rules by other rules) i.e. no recursive reference (direct or indirect alias mutual).
2. If ok then it may also sort the rules into some (more may exist) topological order (= callees precede their caller). Each Infer-rule is allowed to call only rules declared before it. Only the terminal (= leaf) rules Ask, COM and FACT do not call any other rule.
3. Empty Infer-rule (without any parameter) is syntactically bad.

Luckily Quixpert prescribes rule-syntax such that Pascal compiler checks it all: recursion (avoid forward declarations), and "groundedness" (due to illegality of an empty Infer-rule).

Ideally a full (what is it?) symbolic evaluation, analysis and simplification of the rule-set should be done. W. V. O. Quine's book Methods of Logic can help a lot, e.g. with detection of tautologies. Further hints might be found in Warshal's algorithm (also used to check if a grammar is connected and grounded). Ideal would be a reduction process which leaves none or some residuum to be exhaustively checked further.

-------------------- The End ------------------------

ACKNOWLEDGEMENTS

QuiXpert's logic was rapidly prototyped in early 1986 and then carefully ripenned. Too many people made too many suggestions. The most constructive ones came from Hans Blom, Edwin Delsing, Leon van Gorp, Paul van Loon, Jos Schoenmakers and Jack Wiersma. Theo Bruning and Marten van der Woude have kindly read and unkindly criticized the drafts. Last but not least my daughter Monica Hajek has performed 100% mechanical and therefore 100% merciless and unbiased checks of the table of logical operators. Thanks everybody.

ONE REFERENCE AND FURTHER READING *

* Aristoteles: Categoriae; De Interpretatione; Analytica, (382-322).
* Boole G, An Investigation of the Laws of Thought, On which are founded the mathematical theories of logic and probabilities, 1854.
* Venn J, Symbolic Logic, 1881.
Copyright (C) 1986, J. Hajek, Eindhoven, Netherlands

For the purpose of illustration of the line 6 on page 14, this copy of the part of the original QuinExpert source code of December 1986 was added on May 2, 1990, the second day of May 1990 and is an integral part of this report on "A Knowledge engineering logic for smart ep, safer and faster (expert) systems.

No Q-rule means No question asked - rule, e.g. Inter-rule

\[ \text{show('Angle > 90')} \]

02 MEI 1990

\[ \text{function E90: real; begin RULE(26):} \]
\[ \text{E90: GASK('Is one of the angles \text{EQUI} EXACTLY 90 degrees?');} \]
\[ \text{THENFALSE(15) (EQUI); show('Angle = 90');} \]

02 MEI 1990

\[ \text{function ALG0: real; begin RULE(24):} \]
\[ \text{ALG0: GASK('Are all three angles \text{EQUI} EXACTLY 60 degrees?');} \]
\[ \text{THENFALSE(13) (RIGH); THENFALSE(15) (SURL);} \]
\[ \text{THENTRUE(22) (SURL); THENFALSE(15) (RIGH); THENFALSE(14) (LISTR); show('3 $\times$ 60 deg');} \]

function KL59: real; begin RULE(23):
\[ \text{KL59: GASK('Has the LARGEST angle \text{LESS} than 60 degrees?');} \]
\[ \text{show('Largest angle is $< 60$ degrees');} \]

02 MEI 1990

[Illegible handwriting]

function ATRI: real; begin RULE(17):
\[ \text{ATRI=NOQ}; \]
\[ \text{IF [NOT (KL59)] AND [OJ AND NOT (KL59)]:} \]
\[ \text{ELSEFALSE(15) (EQUI); ELSEFALSE(13) (RIGH); ELSEFALSE(14) (SURL);} \]
\[ \text{ELSEFALSE(13) (RIGH); ( <- fine for root); show('A 3-angle');} \]

02 MEI 1990

[Illegible handwriting]

On PERFORMANCE: THENTRUE, THENFALSE, ELSETRUE, ELSEFALSE are semantic & logical shortcuts allowed across the levels. They always DECREASE nr. of questions actually asked. They save extra CPU-time if the BLITZ-format is used (which is fast even without them).

Now you know how to write rules in a REAL-TIME format. You call your local Space Defender and we together can deliverity on "The Ballistic First-In Defense" BLIXPERT (1990).