The logical specification of computer systems

Stalnaker, R. C.; Leshniowski, S.

The logical specification of a problem to be solved by the combination of the parts in fact.

The notations are selected from a subset of the original problem. The notations are not the parts of a large program, and for a solution by computer program it is also useful for the specification of subproblems to be solved.

Summary: A notation is suggested for the rigorous logic.


Department of Computer Science University, Bellat. Leshniowski

October, 1976.

Specication of Computer Programs

C.A.R. Hoare

Specification for the

J-R. Abrial
1. Introduction.

A computer program is, in a sense, a complete and rigorous specification of the problem which it is intended to solve. However, it is for most purposes quite unsuitable; it appears far too late in the timescale of a project; it is far too long and detailed; nor that it is impossible to check intuitively its correspondence with the intentions of its user; and it appears far too late in the timescale of a project, so and it is difficult to change if it is wrong. Does not

For these reasons, there is a clear need for some rigorous method of specifying the solution of a programming problem in a manner independent of the program itself. Such a specification should be in abstract terms, in order to avoid the lengthy detail of computer programs; it should be constructed as the first stage in the development of a program; and it should be capable of intuitive validation by the user of the program, in the same way as an architect’s plans can be checked by his client.
In addition, the initial problem specification should be suggestive of the subsequent development of the program, by a systematic progression of more detailed design decisions. At each subsequent stage, the same specification technique should be used to describe the function of each part of the program, and the interfaces between them. Finally, the specification should serve as the formal criterion of correctness of a program and of its parts, so that it is possible to prove, with any desired degree of formality, that a program meets its specification.

A notation for expressing such specifications has been known as a "specification language", an "assertion language", or even a "system description language". Many "languages" have been proposed for this role, and are acknowledged in the references. This paper attempts a synthesis of the best ideas of its predecessors, but attempts to improve upon them by avoiding all concepts and notations which more properly belong to the programming language itself.
The design of a "specification language" seems to give even more scope to the whims of its designer than the design of a conventional programming language, since there is no longer the pose obligation of efficient implementation to restrain his flights of fancy. For this reason, the design of a specification of language should be judged most strictly by the remaining design criteria listed below:

1. Simplicity: the length of the description, and the number of basic symbols should be less than (say) ALGOL 60.

2. Clarity: the language itself should be completely and unambiguously defined by an intuitively acceptable set. We shall adopt standard predicate calculus with function symbols and equality; also Peano axioms for number theory, and a Zermelo-Fraenkel set theory.

3. Modularity: the language should split into the parts of the language should be logically independent, like the branches of mathematics and logic.

4. General purpose: every feature of the language should be relevant for the description of problems in every major computer application area.

5. Convenience: the notations should be available on conventional typewriters, and in the standard character set for computers. Subscripts and superscripts must be avoided.
(6) Definitional capability: the language should be "extensible" by the normal definitional methods of logic and mathematics.

(7) Good comment conventions: a specification should be an interleaving of formalism and informal commentary, like a good mathematical text.

(8) Correspondence with mathematical intuition: the normal methods conventions of mathematical reasoning should be valid.

(9) Strictly typed structure: the assignment of a unique type to every variable and expression is a practical aid to understanding and reasoning.

(10) Avoidance of defaults: implicit quantified inclusions, default types, etc., must be avoided; they tend to confuse the unsuspecting reader.
It is hoped that the content of this paper will be of interest to programmers, mathematicians, logicians, programming language designers, and hardware designers; in the following ways:

1. It will enlarge the imaginative and inventive capabilities of programmers by introducing them to the most useful abstractions of mathematics and logic.

2. It will encourage mathematicians and logicians to regard computer programming as a natural field of application for their theories and methods.

3. It will inspire programming language designers to maintain design languages in which programs can be developed naturally from their specifications, and can be proved to consistent meet them.

4. It will challenge hardware designers to provide sound and efficient implementations for more abstract problem-oriented concepts, thereby reducing the number of steps required for preparation of a program for execution by computer.

5. It may be possible to construct an automatic proof checker for a proof expressed in a notation similar to the one proposed here.
However, there are two dangers that must be avoided:

(1) The programmer's imagination should not be restricted to the concepts of any fixed "language"; he must be prepared to invent, define, and implement whatever new concepts are required by his problem.

(2) The designer of hardware and software languages should not be too ambitious in the direct implementation of abstract concepts, for which no uniformly satisfactory implementation is possible.

In this design, I have wholly ignored real numbers and their floating point approximations. I do not know enough about this subject to make a design proposal.
1. Comments.

A text consists of commentary interspersed with formal texts. In the sequel, only the formal text will be treated.

Apart from its role in separating formal text, the commentary plays no formal significance, and will be ignored hereafter.
2. Definitions

\[
<\text{definition}> \quad \{<\text{definition}>\}
\]

\[
<\text{formal text}> \quad ::= \quad <\text{simple definition}> \quad | \quad <\text{coordinate definition}>
\]

\[
<\text{definition}> \quad ::= \quad <\text{noun}> \quad ::= \quad <\text{formula}>
\]

\[
<\text{simple definition}> \quad ::= \quad <\text{noun}> \quad ::= \quad <\text{formula} > \quad <\text{defining formula}>
\]

\[
<\text{defining formula}> \quad ::= \quad <\text{formula}>
\]

\[
<\text{noun}> \quad ::= \quad <\text{identifier}>
\]

\[
<\text{coordinate definition}> \quad ::= \quad <\text{no simple definition}>
\]

A definition introduces a noun to stand for the formula on the right-hand side of the definition. The meaning of the text is unchanged if every occurrence of the noun other in subsequent formulae is replaced by its defining formula, enclosed in brackets, if necessary.

The defining formula of a simple definition must be closed. All occurrences of nouns in formulae must be replaceable by the rule given above, and the result of replacement must be a valid formulae.* Hereafter, we shall describe only formulae in which these replacements have been made; we can therefore ignore the existence of nouns and definitions.

*consequently, no noun can be defined directly or indirectly in terms of itself.
Brackets.

Brackets are used to enclose formulae. A bracketed formula begins with an open bracket and ends with a matching close bracket; the following table gives the matching pairs:

\[
\begin{array}{ccc}
( & ) & \quad [ & ] & \quad \{ & \}
\end{array}
\]

The choice of bracket pairs is arbitrary; by convention, the latter pairs are used to enclose larger formulae.

A close bracket must appear in the same line or the same column as its matching open bracket.

All text enclosed in brackets must be written in columns to the right of the opening bracket.

For the syntactic definitions which follow, only round brackets will be used. The round open bracket may be replaced by any of the other open brackets, provided that the corresponding close bracket is replaced by the matching pair.
Omission of brackets.

In principle, the operands of every infix operator should be surrounded by brackets. However, the following concessions are made:

1) When an operator is associative and occurs in a chain, internal bracketing may be omitted, e.g.

\[(x + y + z) = ((x+y)+z) = (x+(y+z))\]

The associative operators are:

\[\equiv \& ; \circ ; \div ; u \cap ; u \setminus ; u \wedge ; u \vee \]

2) When one operator distributes through another, it has a lower precedence (binding power), e.g.

\[a + b \times c = a + (b \times c)\]

In the case of "mutual distributivity", the operator that gives the "smaller" result takes precedence.

3) Operators defined on values of higher order types bind looser than operators on their more elementary components, and propositional operators bind loosest of all.

4) In some cases, long established conventions have been followed.

5) And in other cases, arbitrary decisions have been taken.
These principles lead to the following precedence chain.

To indicate precedence between operators of the same precedence, brackets must be used.
4. Sets.

4.1 Syntax

<set definition> ::= (declarative formula > $ <proposition>)

<declarative> ::= <variable>:<set>

<variable> ::= <identifier>

A declarative formula F is one which may contain declarations

\[ x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n \]

The variables \( x_1, x_2, \ldots, x_n \) must be unique.

The scope of these variables is just the following proposition.

Any other occurrence of the same identifier in F must refer to variables declared in some enclosing formula.

4.2 Semantics. \((F \& P)\) is the set of all values \( y \) for which there exist \( x_1 \in A_1, x_2 \in A_2, \ldots, x_n \in A_n \) which satisfy \( P \),

and for which \( y = F' \), where \( F' \) is formed from \( F \) by replacing all \( x_i : A_i \) simply by \( x_i \).

?? Abbreviation \( x_1, x_2, \ldots, x_n : A \) stands for \( x_1 : A, x_2 : A, \ldots, x_n : A \).

Special cases (1) \((x : A \& P)\) is the subset of all members \( x \) of \( A \)

such that \( P \)

(2) If \( F \) contains no declarations, \((F \& P)\) is empty.

If \( P \) is false (and \( A \) is undefined), the singleton set of \( F \) if \( P \) is true.

and \( F \) is defined.

4.3 Examples:

\((x:NN \& x/3 = 7)\) is the set consisting of 21, 22, and 23.

\((x:NN+y:NN \& x=y)\) is the set of even numbers.
5. Functions.

5.1 Syntax

\[ \text{<function definition> ::= \langle definition\rangle\ \text{formulas} \rightarrow \langle formulas\rangle \ \text{$\langle proposition\rangle$}} \]

5.2 Semantics.

A function definition \((F \rightarrow R $ P)\) is the function that maps \(y\) to \(z\), where there exists an \(x_0, x_1, \ldots, x_n\) such that \(P\) and \(y = F'\) and \(z = R\).

Note that the scope of the declarations in \(F\) is both \(R\) and \(P\).

---

**Abbreviation:** \((F \rightarrow R)\) stands for \((F \rightarrow R $ \text{true})\).

5.3 Examples.

\((x: \mathbb{N} \rightarrow x + y)\) is a function which adds \(y\) to any natural number \(x\).

\((0 \rightarrow 1)\) is the function that maps 0 to 1 but is undefined elsewhere.

\((x: \mathbb{N}, y: \mathbb{N} \rightarrow x - (x/y) \times y)\) is the function \(x\) modulo \(y\).
1. Greatest common divisor.

The divisors of a number $x$ are all numbers $y$ which can be multiplied by some number $z$ to give $x$.

$$\text{divisors} = \{y : \forall z : y \cdot z = x\}$$

The common divisors of two numbers $x$ and $y$ are in the intersection of both their sets of divisors, and we require the greatest of these:

$$\gcd = \max(\{y : \forall z : y \cdot z \in \text{divisors}(x) \cap \text{divisors}(y)\})$$
2. Longest ascending scattered subsequence.

A scattered subsequence of a sequence \( x \) will be represented by the set of indices of the selected elements of \( x \). Every such index is less than the length of \( x \):

\[
\text{subseq}(x) = \{ x : \forall y. y \leq x \} 
\]

"subseq(\( x \)) is the set of all subsequences of \( x \)."

(The ascending subsequences are those whose elements are sorted:

\[
\text{ascend}(x) = \{ x : \forall y. \text{subseq}(x) \implies \exists i, j. y \in x \land i < j \}
\]

We are interested in the length of the longest of them greater than the \( x \)."

\[
\text{longlong}(x) = \{ x : \forall y. \text{subseq}(x) \implies \exists i, j. y \in x \land i < j \}
\]
3. Equivalence classes.

A relation \( r \) which is transitive, reflexive, and symmetric is said to be an equivalence relation.

\[
\begin{align*}
\text{transitive} & \quad = \quad (x \in r \quad r y \quad r z \implies x \in r) \\
\text{symmetric} & \quad = \quad (x \in r \quad y \in r) \\
\text{reflexive} & \quad = \quad (x \in r)
\end{align*}
\]

If \( x \) is an equivalence relation defined on elements of a set \( Y \), the equivalence classes of \( y \) (with \( x \)) is

\[
\text{equivalences} \quad = \quad (x : \text{equivalence} \quad \rightarrow \quad Y(z : \text{Equivalence} \quad \rightarrow \quad A(\text{w}, s), v : \text{E}))
\]
4. Maximal strong components:

A graph can be represented as a relation between nodes

The transitive closure of this graph is just the ancestral of this relation among the nodes of a graph $g$:

$$\text{equiv } \subseteq (g : \text{graph} \Rightarrow \text{equi classes} \circ (\text{equiv}(g)))$$

The strong components of a graph are the equivalence classes induced by this relation.
Maximal strong components.

A graph is identified as a relationship between its nodes. The transitive closure of a relation \( x \) is the relation which \( y \) bears to \( z \) if and only if there is a chain \( y_1, y_2, \ldots, y_n \) such that \( y = y_1 \) and \( x = y_n \) and \( y_i \in x \land y_{i+1} \) for \( i = 1 \ldots n-1 \). (Note: if \( n = 1 \), \( y = 2 \)).

This can be defined more economically, using relational powers:

\[
\text{transitive closure } = \quad (x : s \rightarrow s) \cup (x : p \rightarrow s) \\
\]

Two nodes are strongly connected in a graph if they are related both by the transitive closure and by its converse.

\[
\text{strong connection } = \quad (x : s \rightarrow s) \cap \text{transitive closure}(x) \cap \text{transitive closure}(x \cup p) \\
\]

Strong connection is an equivalence relation. The strong components of a graph are simply the equivalence classes induced by it.

\[
\text{strong components } = \quad (x : s \rightarrow \text{equivalence classes}(\text{strong connection}(x))) \\
\]
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Reading</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \cdot y$</td>
<td>product</td>
<td>$x$ times $y$</td>
<td>NN</td>
</tr>
<tr>
<td>$x / y$</td>
<td>quotient</td>
<td>$x$ over $y$</td>
<td>NN</td>
</tr>
<tr>
<td>$x + y$</td>
<td>sum</td>
<td>$x$ plus $y$</td>
<td>NN</td>
</tr>
<tr>
<td>$x - y$</td>
<td>difference</td>
<td>$x$ minus $y$</td>
<td>NN</td>
</tr>
<tr>
<td>$\min x, y$</td>
<td>minimum</td>
<td>$x$ min $y$</td>
<td>NN</td>
</tr>
<tr>
<td>$\max x, y$</td>
<td>maximum</td>
<td>$x$ max $y$</td>
<td>NN</td>
</tr>
</tbody>
</table>

(2) Structures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Reading</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y$</td>
<td>ordered pair</td>
<td>$x$ paired with $y$</td>
<td>t, s, st</td>
</tr>
<tr>
<td>$1x$</td>
<td>first alternative</td>
<td>tag one $x$</td>
<td>s, st</td>
</tr>
<tr>
<td>$2x$</td>
<td>second alternative</td>
<td>tag two $x$</td>
<td>t, st</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>empty sequence</td>
<td>null</td>
<td>Qs</td>
</tr>
<tr>
<td>$Qx$</td>
<td>unit sequence</td>
<td>queue $x$</td>
<td>s, Qs</td>
</tr>
<tr>
<td>$x \cdot y$</td>
<td>concatenation</td>
<td>$x$ joined $y$</td>
<td>Qs, Qs, Qs</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
<td>$</td>
<td>sequence length</td>
</tr>
<tr>
<td>TYPES</td>
<td>x</td>
<td>y</td>
<td>result</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>Sets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ø</td>
<td>empty set</td>
<td>empty</td>
<td>Ps</td>
</tr>
<tr>
<td>Sx</td>
<td>singleton set</td>
<td>singleton x ∈</td>
<td>Ps</td>
</tr>
<tr>
<td>x ∩ y</td>
<td>intersection</td>
<td>x and y</td>
<td>Ps, Ps</td>
</tr>
<tr>
<td>x + y</td>
<td>union</td>
<td>x or y</td>
<td>Ps, Ps</td>
</tr>
<tr>
<td>x − y</td>
<td>symmetric difference</td>
<td>x without y</td>
<td>Ps, Ps</td>
</tr>
<tr>
<td>T x</td>
<td>description</td>
<td>the (only) x</td>
<td>Ps</td>
</tr>
<tr>
<td>C x</td>
<td>cardinality</td>
<td>size</td>
<td>Ps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>the natural numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x × y</td>
<td>direct product</td>
<td>x and y</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x + y</td>
<td>disjoint union</td>
<td>x or y</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x ⊇ y</td>
<td>sequence over x</td>
<td></td>
<td>Ps</td>
</tr>
<tr>
<td>P x</td>
<td>power set</td>
<td>subsets of x</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x → y</td>
<td>relation set</td>
<td>all relations between x and y</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x ↷ y</td>
<td>function set</td>
<td>all functions from x to y</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x ↳ y</td>
<td>map set</td>
<td>all maps from x to y</td>
<td>Ps, Pt</td>
</tr>
<tr>
<td>x ↡ y</td>
<td>measure set</td>
<td></td>
<td>Ps ⊆ Ps</td>
</tr>
<tr>
<td>P y</td>
<td>set of all subsets of y</td>
<td></td>
<td>Ps</td>
</tr>
<tr>
<td>N y</td>
<td>set of all natural numbers</td>
<td></td>
<td>Ps</td>
</tr>
</tbody>
</table>
(5) Relations.

\[ R \times \] function from sequence \( f \) to function \( g \) \( \times f \times g \)

\[ I \times \] identity \( I \times x \) \( I \times y \)

\[ \times \text{relational power} \times y \text{ to the minus } y \times x \text{ to the minus } y \times \text{star} \times \]

\[ \times \text{composition} \times y \text{ composed with } x \text{ then } y \times \]

\[ \times \text{update} \times \text{else } y \times y \]

\[ \times \text{domain} \times \text{domain of } x \times \]

\[ \times \text{range} \times \text{range of } x \times \]

\[ \times \text{image} \times \text{of } y \times \]

\[ \times \text{summed relation} \times x \text{ of } y \times \]

\[ \times \text{functional application } x \text{ at } y \times y^x \times \]

\[ \times \text{sequence from function} \times \text{of } x \times \]

\[ \text{NN} \times \text{NN} \times Q \times \text{of } x \times \text{of } y \times \]
\( x \odot y \)
multiplication
the \( \times \) of \( x \) and \( y \)
\( \odot \rightarrow \odot \)
set \( Q \) s \( \odot \) t
\( Q_3 \odot (s \odot t) \odot Q_t \quad Q_u \)
\( \sum \)
sum \( S \)
\( \prod \)
product \( P \)

\( \cup \)
union
\( \cup \rightarrow \cup \)
\( \exists \)
exists \( \exists \)
\( \cap \)
intersection
\( \cap \rightarrow \cap \)
\( \exists \)
exists \( \exists \)
\( \min \)
minimum
\( \min \rightarrow \min \)
\( \exists \)
exists \( \exists \)
\( \max \)
maximum
\( \max \rightarrow \max \)
\( \exists \)
exists \( \exists \)
<table>
<thead>
<tr>
<th>x &amp; y</th>
<th>membership</th>
<th>\textit{$x\text{ is a member of }y$}</th>
<th>Ps</th>
<th>PRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = y</td>
<td>equation</td>
<td>\textit{$x\text{ equals }y$}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x \subseteq y</td>
<td>containment</td>
<td>\textit{$x\text{ is a subset of }y$}</td>
<td>Ps</td>
<td>Ps</td>
</tr>
<tr>
<td>x \leq y</td>
<td>inequality</td>
<td>\textit{$x\text{ does not exceed }y$}</td>
<td>NN</td>
<td>NN</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>strict inequality</td>
<td>\textit{$x\text{ is less than }y$}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x \preceq y</td>
<td>initial subsequence</td>
<td>\textit{$x\text{ begins }y$}</td>
<td>Qs</td>
<td>Qs</td>
</tr>
<tr>
<td>x \preceq y</td>
<td>infix relation</td>
<td>\textit{$x\text{ is }z\text{ of }y$}</td>
<td>s, \text{ s}R^+</td>
<td>s, \text{ s}R^+</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>NAME</td>
<td>READING</td>
<td>TYPES</td>
<td>DEFINITION</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>------------------</td>
<td>-------</td>
<td>--------------------</td>
</tr>
<tr>
<td>~ P</td>
<td>negation</td>
<td>not P</td>
<td>PROP</td>
<td>P &amp; (~P)</td>
</tr>
<tr>
<td>P &amp; Q</td>
<td>conjunction</td>
<td>P and Q</td>
<td>PROP</td>
<td>P &amp; (~P) &amp; ~Q</td>
</tr>
<tr>
<td>P v Q</td>
<td>disjunction</td>
<td>P or Q</td>
<td>PROP</td>
<td>P &amp; (~P) &amp; ~Q</td>
</tr>
<tr>
<td>P =&gt; Q</td>
<td>implication</td>
<td>if P then Q</td>
<td>PROP</td>
<td>~ P x Q</td>
</tr>
<tr>
<td>P = Q</td>
<td>equivalence</td>
<td>P if and only if Q</td>
<td>PROP</td>
<td>(P =&gt; Q) &amp; (Q =&gt; P)</td>
</tr>
</tbody>
</table>

| z P     | set abstraction | set of z such that P | PROP | P e |
| z -> x  | functional abstraction | function from z to x | s t | s e t |
| z -> x $ P | restriction | function such that P | s t | s e t |

| A x     | generalisation  | Ps                | PROP | P $ \sim y \in \mathbb{R} = \emptyset |
| E x     | existence       | Ps                | PROP | \forall x = \emptyset |
Index

disjoint union
begins
is contained in
is a member of
relational image

join of sequences
else
min
intersection
relational composition
relational power
sequence defined from function
relation
curried relation
union
or
maximum
direct product

A
all

B

C
cardinality
domain
equals
exists
function defined by sequence

D
digit

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

length of sequence
minimum
intersection
empty set
powerset
greateset
range
singleton set
the unique
union

maximum

recursive