Some

A Notation for the Specification of Computer Programs

J.-R. Abrial.

C. A. R. Hoare.

~~S.R.C. Senior Fellow~~

Department of Computer Science

The Queen's University, Belfast

October, 1976

E. C. Luschei The logical Systems of Lesniewski. North-Holland 1962.

Słupecki S. Lesniewski's calculus of names Studia Logica 1956?

Summary. A notation is suggested for the rigorous ~~for~~ specification of a problem preparatory to its solution by computer program. It is also intended for the specification of subproblems to be solved by the parts of a large program, and for a proof that the combination of the parts in fact solves the original problem. The notations are selected ~~as a [illegible]~~ to express a subset of the concepts of mathematics and logic, and a superset of the facilities of a general-purpose programming language.

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; ~~so 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 ~~very~~ difficult to change ~~if~~ when 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~~ highly abstract, ~~terms,~~ in order to avoid the ~~lengthy~~ voluminous 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 discerning client.

In addition, the initial problem specification ~~should~~ may be ~~the first step in the~~ 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 contains no logical errors, meets its specification.

A notation for expressing such specifications has been known as a "specification language", an "assertion language" or even a "system description" "systems analysis 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 ~~pros~~ obligation of efficient implementation to restrain his (more elaborate) 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 ~~num~~ length of the description, and the number of basic symbols should be less than (say) ALGOL60

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

Linden's paper SIGSOFT Apr. 78

(7) Good comment conventions: a specification should be an interleaving of formalism ~~text~~ 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~~ 9) Avoidance of defaults: implicit ~~quand~~ quantifications, 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 excecution 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 ~~softw~~ programming 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 reasonable design proposal.

1. Comments.

<text> ::= <commentary> {<formal text> <commentary>}

<commentary> ::= <comment line> {<comment line>}

<formal text> ::= <formal line> {<formal line>}

<formal line> ::= <a ~~non-blank~~ line with space in its first column>

<comment line> ::= <a blank line> |
<a line with a printed character in its first column>

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

Apart from its role in (explaining and) separating formal text, the commentary ~~plays~~ has no formal significance, and will be ignored hereafter.

2. Definitions

<formal text> ::= <definition> {<definition>} ~~<simple definition>~~ | ~~<subordinate definition>~~

<definition> ::= <noun> == <formula>

~~<simple definition> ::= <noun> <= <formula> <defining formula>~~

~~<defining formula> ::= <formula>~~

~~<noun> ::= <identifier>~~

~~<subordinate definition> ::= <no <simple definition>~~

A definition introduces a noun to stand for its defining ~~the~~ formula ~~on the right hand side of the definition~~. The meaning of the text is unchanged if every occurrence of the noun in other ~~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 (already) 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.

The mean[illegible]

3. 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:

$($ $)$

$[$ $]$

$\{$ $\}$

$\underline{(}$ $\underline{)}$

$\underline{[}$ $\underline{]}$

$\underline{\{}$ $\underline{\}}$

The choice of bracket pairs is arbitrary; by convention, the later 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.

In 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:

$$\equiv \quad \underline{\vee} \quad \& \quad ; \quad \underline{\circ} \quad \underline{:Z} \quad \underline{\cup} \quad \underline{\cap} \quad ++ \quad \underline{\uplus} \quad \underline{m} \quad + \quad *$$

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

$$a + b*c = a + (b*c).$$

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. ?

$*$ $/$

$+$ $-$

$\underline{m}$

$\underline{w}$

,

$\underline{j}$ $\quad \underline{1}, \underline{2}, \underline{3}, \ldots$ $\quad \underline{J}$

$\underline{n}$ $\quad \underline{I}$

$\underline{u}$ $++$ $--$ $\quad \underline{L}$

$\underline{x}$

$\underline{a}$

$\underline{p}$: $\underline{Q}$

$\underline{r}$ $\quad \Rightarrow$ $\Leftrightarrow$

$\underline{F}$ $\underline{I}$

$\underline{P}$ $\underline{\ell}$ $-$

; $\underline{o}$

$\underline{l}$ $\underline{f}$ $\underline{s}$ $\preceq$ $\leftarrow$ @ .

$\underline{g}$

$\underline{e}$ $=$ $\subseteq$ $\leq$ $<$ $\underline{b}$

&

$\underline{V}$

$\Longrightarrow$

$\equiv$

To indicate precedence between operators of the same precedence, brackets <u>must</u> be used.

4. Sets.

4.1 Syntax

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

<declaration> ::= <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 identifiers 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 (or F is undefined), and is 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

Same as declarative formula on previous page but with scope to end of function definition?

<function definition> ::= (<definitional formula> → <formula> \$ <proposition>)

5.2. Semantics.

A function definition $(F \to R \$ P)$ is the function that maps y to z, where there exists an $x_1, x_2, \ldots, x_n$ such that P and $y = F'$ and $z = R$. ? $z = R'$

Note that the scope of the declarations in F is both R and P

~~5.3~~

Abbreviation: $(F \to R)$ stands for $(F \to R \$ \text{true})$

5.3 Examples.

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

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

$(x : \mathbb{N}, y : \mathbb{N} \to x - (x/y) * y)$ 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

$$\text{divisors} = (x{:}\mathbb{N} \to (y{:}\mathbb{N}\ \$\ \underline{E}(z{:}\mathbb{N}\ \$\ y * z = x)))$$

Shouldn't the "z" be bound more explicitly to the "E"?

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 them:

$$\text{gcd} = (x{:}\mathbb{N}, y{:}\mathbb{N} \to \underline{W}(\text{divisors}(x)\ \underline{n}\ \text{divisors}(y)))$$

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:

$$\text{subsequ} == (x : \underline{Q}NN \rightarrow \underline{P}\,(y : NN \ \$\ y < \underline{L}x))$$

? $x : \underline{Q}NN$ or $x : \underline{*}NN$

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

The ascending subsequences are those whose elements are sorted:

?

$$\text{ascend} == (x : \underline{Q}NN \rightarrow (y : \overset{\text{subseq@x}}{\text{subsequ}(x)} \ \$\ \underline{A}(i : y, j : y \ \$\ \underline{F}x@i \leq \underline{F}x@j)))$$

ie if $z \ \underline{e}$ ascend(x) then for all i and j in z, the i^{th} element of x is not greater than the j^{th}

We are interested in the length of the longest of them

$$\text{leng long} == (x : \underline{Q}NN \rightarrow \underline{W}\,(\underline{L}\, y : (\text{ascend}@x) \ \$\ \text{true}))$$

3. Equivalence classes.

A relation r which is transitive, reflexive and symmetric is said to be an ~~a~~ equivalence relation

$$x : s \, \underline{r} \, s \; \$ \; x ; x \, \underline{\subseteq} \, x$$

$$\text{transitive} == \left(x : s \underline{r} s \; \$ \; \underline{A}\left(x:s, y:s, z:s \; \$ \; y \underline{e} x \underline{f} z \; \& \; z \underline{e} x \underline{f} w, \Rightarrow y \underline{e} x \underline{f} w \right) \right)$$

$$x : s \, \underline{r} \, s \; \$ \; x = x \, \underline{\rho}^{-1}$$

$$\text{symmetric} == \left(x \; \$ \; \underline{A} \, y, z \; \$ \; y \underline{e} x \underline{f} z, \Rightarrow z \underline{e} x \underline{f} y \right)$$

$$\text{reflexive} == \left(x \; \$ \; \underline{A}\left(y:s \; \$ \; y \underline{e} x \underline{f} y \right) \right) \qquad x : s \, \underline{r} \, s \; \$ \; \underline{I}_s \underline{\subseteq} x \; \cancel{x \subseteq I_s}$$

$$\text{equivalence} == \text{transitive} \; \underline{n} \; \text{reflexive} \; \underline{n} \; \text{symmetric}$$

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

$$\text{equivclasses} == \left(x : \text{equivalence} \rightarrow \underline{T}\left(z : \underline{PP}s \; \$ \; \underline{A}\left(u:s, v:s \; \$ \; u \underline{e} x \underline{f} v \equiv \underline{E}\left(w:z \; \$ \; u \underline{e} w \; \& \; v \underline{e} w \right) \right) \right) \right)$$

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:

$$\text{equiv} \Leftarrow (g : \text{graph} \rightarrow \underline{*}g \cap \underline{*}(g\ \underline{p}\ {-1}))$$

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

$$\text{max strong comp} \Leftarrow (g : \text{graph} \rightarrow \text{equiv classes}(\text{equiv}(g)))$$

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 $z = y_n$ and

$y_i \,\underline{e}\, x \,\underline{f}\, y_{i+1}$ for $i = 1..n-1$. (note: if $n = 1$, $y = z$)

This can be defined more economically, using relational powers:

transitive closure ==

$$\left(x : s\,\underline{r}\,s \rightarrow \underline{U}\left(x\,\underline{p}\,n : NN \,\$\, true\right)\right)$$

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

$$strongconnection == \left(x : s\,\underline{r}\,s \rightarrow transitiveclosure(x) \cap transitiveclosure\left(x\,\underline{p}\,{-1}\right)\right)$$

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

$$strongcomponents == \left(x : s\,\underline{r}\,s \rightarrow equivclasses\left(strongconnection(x)\right)\right)$$

SYMBOL	NAME	READING	TYPES		
			x	y	result
(1) Natural Numbers.					
$x * y$	product	x times y	NN	NN	NN
x / y	quotient	x over y	"	"	"
$x + y$	sum	x plus y	"	"	"
$x - y$	difference	x minus y	"	"	"
$x \underline{m} y$	minimum	x min y	"	"	"
$x \underline{w} y$	maximum	x max y	"	"	"
(2) Structures					
x , y	ordered pair	x paired with y	s	t	$s \underline{x} t$
$\underline{1} x$	first alternative	tag one x	s		$s \underline{a} t$
$\underline{2} x$	second alternative	tag two x	t		$s \underline{a} t$
$\underline{QO}$	empty sequence	null			$\underline{Q} s$
$\underline{Q} x$	unit sequence	queue x	s		$\underline{Q} s$
$x \underline{j} y$	catenation	x joined to y	$\underline{Q} s$	$\underline{Q} s$	$\underline{Q} s$
$\underline{L} x$	sequence length	length of x	$Q s$		NN

TYPES

			x	y	result	
(3) Sets						
$\underline{O}$	empty set	empty			$\underline{P}s$	$(x:s \$ \sim x = x)$
$\underline{S}x$	singleton set	singleton x	s		$\underline{P}s$	$(y:s \$ y = x)$
$x \,\underline{n}\, y$	intersection	x and y	$\underline{P}s$	$\underline{P}s$	$\underline{P}s$	$(z:s \$ z \underline{e} x \;\&\; z \underline{e} y)$
$x \,\underline{u}\, y$	union	x or y	"	"	"	$(z:s \$ z \underline{e} x \vee z \underline{e} y)$
$x +\!+ y$	symmetric difference	x plus y	"	"	"	$(z:s \$ \sim(z \underline{e} x \equiv z \underline{e} y))$
$x -\!- y$	set difference	x without y	"	"	"	$(z:s \$ z \underline{e} x \;\&\; \sim z \underline{e} y)$
$\underline{I}x$	description	the (only) x	$\underline{P}s$		s	
$\underline{C}x$	cardinality	size	$\underline{P}s$		NN	$x.\underline{Y}[F:(\underline{P}s \Rightarrow NN) \$ (\underline{O} \to 0)\, \underline{u}\, (y:\underline{P}s\, \underline{u}\, \underline{S}z:s \to f(y)+1 \$ \sim z \underline{e} y)]$
(4) Types.						
NN	the natural numbers				$\underline{P}\, NN$	$(x \$ \underline{E}y\ (y = x+1))$ $\underline{Y}(x:\underline{P}NN \Rightarrow \underline{S}0\ \underline{u}$ $(y:NN+1 \$ y \underline{e} x))$
$x \,\underline{x}\, y$	direct product	x and y	$\underline{P}s$	$\underline{P}t$	$\underline{P}(s \underline{x} t)$	$(z, w \$ z \underline{e} x \;\&\; w \underline{e} y)$
$x \,\underline{a}\, y$	disjoint union	x or y	"	"	$\underline{P}(s \underline{a} t)$	$(\underline{1}z \$ z \underline{e} x)\ \underline{u}\ (\underline{2}z \$ z \underline{e} y)$
$\underline{*}x$	sequence set	sequences over x	$\underline{P}s$		$\underline{P}\underline{Q}s$	$(y \$ \underline{s}y \subseteq x)$
$\underline{P}x$	power set	subsets of x	$\underline{P}s$		$\underline{P}\underline{P}s$	$(y \$ y \subseteq x)$
$x \,\underline{r}\, y$	relation set	all relations between x & y	$\underline{P}s$	$\underline{P}t$	$\underline{P}\underline{P}(s \underline{x} t)$	$\underline{P}(s \underline{x} t)$
$x \Rightarrow y$	function set	all functions from x to y	$\underline{P}s$	$\underline{P}t$	$s \,\underline{r}\, y$	$\underline{P}(s \underline{x} t) \;\&\; (z \$ (\underline{A}w, v, u \$ w, v \underline{e} x \;\&\; w, u \underline{e} x \Rightarrow v = u))$
$x \Leftarrow\!\Rightarrow y$	map set	all maps between x and y	"	"	$s \,\underline{r}\, y$	$= (x \Rightarrow y)\ \underline{n}\ (y \Rightarrow x)$
$\underline{Y}x$	recursive set		$\underline{P}s \Rightarrow \underline{P}s$		$\underline{P}\,\underline{Y}x$	$\underline{s}\underline{O}\,.\,\underline{U}\ (x\, \underline{p}\, n:NN \$ \text{true})$
						$\underline{N}(y:\underline{P}s \to y \$ \underline{A}(z \$ z \subseteq y \Rightarrow x(z) \subseteq y))$

(5) Relations.

Fx	Function from sequence	function x	Qs		NN ⇒ s	x. Yf: Qs ⇒ NN → (0 → 0) u x:ℰs; y:Qs → [(Lx → y) u F(x)]
Ix	identity	identity over x	Ps		s x s	y, z $ y = z & y e x
x p: y	relational power	x to the y	s r s	NN	"	y. Yf → [(0 → Ix) u (n+1 → f o x)]
x p- y	converse power	x to the minus y	"	NN	"	z, w $ (z, w) e x p u
*x	ancestral	star x	"		"	
x ; y or y o x	composition	{ y composed with x / x then y	s r t	t r u	s r u	(z, w $ E v $ z, v e x & v, w e y)
x l y	update	x else y	s r t	s r t	s r t	x u (y - Dx x Dy) (z, w $ z e Dx & z, w e y)
Dx	domain	domain of x	s r t		Ps	(y $ E z $ y, z e x)
Rx	range	range of x	"		Pt	(y $ E z $ z, y e x)
x f y	image	x of y	s r t	Pt	Ps	
x s y	curried relation		s	(s x t) x u	t x u	(z:t, w:u $ ((x, z), w) e y)
x@y or y.x	Functional application	x at y or y's x	s ⇒ t	s	t	I(z $ (y, z) e x) I(x f S y)
		"				"
x q n	sequence from function		NN ⇒ s	NN	Qs	t(s $ Ls = n & F(s) = x)

$x @ y$	maplist	the x of y	$s \Rightarrow t$	$\underline{Q}s$	$\underline{Q}t$	$: \underline{y}.(\forall f \to (\underline{q0} \to q0)\, \underline{u}(z;qw \to f(z);qx(w)))$
$x.\ \underline{y} @ z$	componentwise operation		$\underline{Q}s.(s \times t \Rightarrow u)\ \underline{Q}t$	$\underline{Q}u$		
sum	S					
prod	P					
Ux	union		PPt $*\, s \Rightarrow \underline{P}t$	P t		$(y \$ E z,w \$ z,w \in x \ \&\ y \in w)$
Nx	intersection		"	"		
Mx	minimum		$\underline{P}$ NN	NN		
Wx	maximum		"	NN		

(5)

$x \underline{e} y$	membership	x is a (member of) y	s	$\underline{P}s$	PRED
$x = y$	equation	x equals y	s	s	"
$x \subseteq y$	containment	x is a subset of y	$\underline{P}s$	$\underline{P}s$	"
$x \leq y$	inequality	x does not exceed y	NN	NN	"
$x < y$	strict inequality	x is less than y	"	"	"
$x \underline{b} y$	initial subsequence	x begins y	$\underline{Q}s$	$\underline{Q}s$	"
X $x . z\ y$	infix relation	x is a z of y	s, s$\underline{R}$t	t	"

SYMBOLS	NAME	READING	TYPES op1	op2	result	DEFINITION
$\sim P$	negation	not P	PROP		PROP	$(\sim P)\&(\sim P)$
$P \& Q$	conjunction	P and Q	PROP	PROP	PROP	$\sim(\sim P)\&\sim(\sim Q)$
$P \underline{v} Q$	disjunction	P or Q	"	"	"	$\sim(\sim P \& \sim Q)$
$P \Rightarrow Q$	implication	if P then Q	"	"	"	$\sim P \underline{v} Q$
$P \equiv Q$	equivalence	P if and only if Q	"	"	"	$(P \Rightarrow Q)\&(Q \Rightarrow P)$
$z \$ P$	set abstraction -	set of z such that P	s	PROP	$\underline{P}\,s$	
$z \rightarrow x$	Functional abstraction	Function from z to x	s	t	$s \Rrightarrow t$	
$z \rightarrow x \$ P$	restriction	Function " such that P	s	t	$s \Rrightarrow t$	
$\underline{A}\, x$	generalisation		$\underline{P}\,s$		PROP	$(y \$ \sim y \underline{\in} x) = \underline{s0}$
$\underline{E}\, x$	existence		$\underline{P}\,s$		PROP	$\sim x = \underline{s0}$

Index

a	disjoint union
b	begins
c	is contained in
d	
e	is a member of
f	relational image
g	
h	
i	
j	join of sequences
k	
l	else
m	min
n	intersection
o	relational composition
p	relational power
? q	sequence defined from function.
r	relation
s	curried relation
t	
u	union
v	or
w	maximum
x	direct product
y	
z	

A	all
B	
C	cardinality
D	domain
E	exists
F	function defined by sequence
G	
H	
I	identity relation
J	singleton sequence
K	
L	length of sequence
M	minimum
N	intersection
O	empty set
P	powerset
Q	queueset
R	range
S	singleton set
T	the unique
U	union
V	
W	maximum
X	
Y	recursive
Z	