A Abstract syntax – Normative Annex

Notes on this section of the Z Standard

Section title: Abstract syntax
Note: Updated in preparation for version 1.2.
Section editor: John Nicholls
Source file: absyn.tex
Most recent update: 11th August 1995
Formatted: 14th September 1995

A.1 Introduction

The abstract syntax is central to the definition of Z. It stands between concrete representations of Z documents, such as marks on paper or images on screens, and the abstract entities, semantic relations and semantic functions used for defining their meaning.

The abstract syntax has the following objectives:

> to identify and separately name the abstract entities of Z;
>
> to simplify and unify underlying concepts, putting like things with like, and reducing unnecessary duplication.

There are many possible ways of constructing an abstract syntax for Z, and the choice of the form given below is a matter of judgement, taking into account the somewhat conflicting aims of simplicty and economy of semantic definition, and the maintenance of a clear relationship with the concrete representation.

The syntax is presented as a set of production rules in which each entity is defined in terms of its constituent parts. For each of the entities defined in the abstract syntax, there is a subsection in the main part of the standard defining its representation and meaning.

A.2 Metalanguage for abstract syntax

::=	definition symbol
|	disjunction symbol

In the following definitions, some abstract entities are defined as lists of other entities, interspersed with *separators*. Examples of separators are: comma, ×. To denote arbitrary numbers of entities (and separators) in a list, the following notation is used:

...	ellipsis, denoting a finite (possibly zero) number of occurrences of the preceding entity together with appropriate separator(s)

Terminal entities. Terminal entities are the abstract semantic entities of Z, and are written in uppercase sans-serif font.

In addition, the abstract syntax definitions contain operators, symbols and keywords similar to those used in concrete representations of Z, as defined in the Lexis (Annex C). These are written in this way to indicate the relationship of each abstract definition with the concrete form of the notation.

The relationship between abstract and concrete forms of each entity is indicated in the entity definitions in the main body of the standard, under the heading "Representation and transformation".

A.3 Specification

SPEC ::= SECTION, SECTION, ...

A.4 Section

SECTION ::= **section** NAME **parents** NAME, NAME, ... • PARAGRAPHS

PARAGRAPHS ::= PAR, PAR, ...

A.5 Paragraph

PAR ::= GIVENSETDEF
| GLOBALPRED
| GLOBALSCHEMA
| GENERICSCHEMA
| GLOBALDEF
| GENERICDEF
| FREETYPEDEF
| CONJECTURE

GIVENSETDEF ::= **given** [NAME, NAME, ...]

GLOBALPRED ::= **where** PRED

GLOBALSCHEMA ::= **def** SCHEMA

GENERICSCHEMA ::= **gendef** [NAME, NAME, ...] **const** SCHEMA

GLOBALDEF ::= **abbr** NAME := EXP

GENERICDEF ::= **abbr** [NAME, NAME, ...] := EXP

CONJECTURE ::= ⊢ PRED

A.6 Schema

SCHEMA ::= SDECL
| SDES
| SGENDES
| SCONSTRUCTION
| SNEGATION
| SDISJUNCTION
| SCONJUNCTION
| SIMPLICATION
| SEQUIVALENCE
| SPROJECTION
| SHIDING
| SUNIVQUANT
| SEXISTSQUANT
| SUNIQUEQUANT
| SRENAMING
| EXPSCHEMA

SDECL ::= NAME, NAME, ... : EXP
SCONSTRUCTION ::= SCHEMA | PRED
SNEGATION ::= $\neg$ SCHEMA
SDISJUNCTION ::= SCHEMA $\lor$ SCHEMA
SCONJUNCTION ::= SCHEMA $\land$ SCHEMA
SIMPLICATION ::= SCHEMA $\Rightarrow$ SCHEMA
SEQUIVALENCE ::= SCHEMA $\Leftrightarrow$ SCHEMA
SPROJECTION ::= SCHEMA $\upharpoonright$ SCHEMA
SHIDING ::= SCHEMA $\setminus$ [NAME, NAME, ...]
SUNIVQUANT ::= $\forall$ SCHEMA $\bullet$ SCHEMA
SEXISTSQUANT ::= $\exists$ SCHEMA $\bullet$ SCHEMA
SUNIQUEQUANT ::= $\exists_1$ SCHEMA $\bullet$ SCHEMA
SRENAMING ::= SCHEMA [NAME/NAME, /NAME/NAME, ...]
SCOMPOSITION ::= SCHEMA $\fatsemi$ SCHEMA
SDECORATION ::= SCHEMA DECOR
EXPSCHEMA ::= EXP

A.7 Predicate

```
PRED              ::= EQUALITY
                    | MEMBERSHIP
                    | TRUTH
                    | FALSEHOOD
                    | NEGATION
                    | DISJUNCTION
                    | CONJUNCTION
                    | IMPLICATION
                    | EQUIVALENCE
                    | UNIVERSALQUANT
                    | EXISTSQUANT
                    | UNIQUEQUANT
                    | SPRED
                    | PREDSUBSTITUTION

EQUALITY          ::= EXP = EXP
MEMBERSHIP        ::= EXP ∈ EXP
TRUTH             ::= true
FALSEHOOD         ::= false
NEGATION          ::= ¬ PRED
DISJUNCTION       ::= PRED ∨ PRED
CONJUNCTION       ::= PRED ∧ PRED
IMPLICATION       ::= PRED ⇒ PRED
EQUIVALENCE       ::= PRED ⇔ PRED
UNIVERSALQUANT    ::= ∀ SCHEMA • PRED
EXISTSQUANT       ::= ∃ SCHEMA • PRED
UNIQUEQUANT       ::= ∃₁ SCHEMA • PRED
SPRED             ::= SCHEMA
PREDSUBSTITUTION  ::= EXP ⊙ PRED
```

A.8 Expression

EXP ::= IDENT
| GENINST
| NUMBERL
| SETEXTN
| SETCOMP
| POWERSET
| TUPLE
| PRODUCT
| TUPLESELECTION
| BINDINGEXTN
| THETAEXP
| SCHEMAEXP
| BINDSELECTION
| FUNCTAPP
| DEFNDESCR
| IFTHENELSE
| EXPSUBSTITUTION

IDENT ::= NAME
GENINST ::= NAME [EXP, EXP, . . .]
NUMBERL ::= NUMBER
SETEXTN ::= {EXP, EXP, . . . }
SETCOMP ::= {SCHEMA • EXP}
POWERSET ::= **Pow** EXP
TUPLE ::= (EXP, EXP, EXP, . . .)
PRODUCT ::= EXP × EXP × EXP × . . .
TUPLESELECTION ::= EXP . NUMBER
BINDINGEXTN ::= ⦇ NAME := EXP, NAME := EXP, . . . ⦈
THETAEXP ::= θ SCHEMA DECOR
SCHEMAEXP ::= SCHEMA
BINDSELECTION ::= EXP . NAME
FUNCTAPP ::= EXP (EXP)
DEFNDESCR ::= μ SCHEMA • EXP
IFTHENELSE ::= **if** PRED **then** EXP **else** EXP
EXPSUBSTITUTION ::= EXP ⊙ EXP

A.9 Name

NAME ::= WORD DECOR, . . . ,DECOR

□

B Concrete syntax – Normative Annex

Notes on this section of the Z Standard

Section title: Concrete syntax
Note: This section is based on the proposal by Will Harwood and Pete Steggles (Document 173 dated 6th March 1995).
Section editor: (John Nicholls / Pete Steggles)
Contributions by: Will Harwood, Pete Steggles, Chris Sennett, Rob Arthan, Stephen Brien, . . . (more to be added)
Source file: concrete.tex
Most recent update: 23rd August 1995
Formatted: 14th September 1995

B.1 Introduction

The syntax defined in this Annex, together with the associated Lexis in Annex C, are designed to meet the following requirements:

- to be as close as possible to existing descriptions of Z;
- to permit the substitution of equals for equals;
- to make unparsing *injective* (i.e. different legal ASTs should have different unparsed forms) and *total* (i.e. there should be a concrete syntax for every legal AST).

B.2 Metalanguage for concrete syntax

The syntax is defined as BNF production rules, in which

- terminals are fully capitalized, e.g. `ELSE`. Representations of terminal entities are defined in the Lexis, Annex C.
- non-terminals are partly capitalized, e.g. `DeclPart`

The following notation is used:

=	definition symbol
|	separator of alternative definitions
[ ]	brackets enclosing optional entities
{ }	brackets enclosing entities which may occur zero or more times

Precedence and associativity. Where appropriate, precedence and associativity of operators are defined under each subsection heading.

B.3 Specification

```
Specification = Paragraph {SEP Paragraph}
```

B.4 Paragraph

```
Paragraph  = SQBRA NameList1 SQKET
%           | Predicate                 % changed 13/9/95
           | AX [DeclPart] [ST Predicate] END
           | GEN Formals BAR [DeclPart] [ST Predicate] END
           | SCH Name [Formals] IS [DeclPart] [ST Predicate] END
           | Name [Formals] DEFINE_EQUAL Expression
           | TURNSTILE Predicate
           | Name FREEEQUALS Branch {VBAR Branch}
           | Fixity

Formals    = SQBRA NameList1 SQKET

Branch     = Name [FREEBRA Expression FREEKET]
```

The tokens `AX`, `BAR`, `END`, `GEN`, `IS`, `SCH`, `ST` have special graphical conventions associated with them.

Informal text is treated as whitespace by the lexer.

B.5 Predicate

```
Predicate  = Expression
           | Predicate CONJ Predicate
           | PRED Expression
           | EXISTS TextOrExpression DOT Predicate
           | EXISTS1 TextOrExpression DOT Predicate
           | FORALL TextOrExpression DOT Predicate
           | Predicate IFF Predicate
           | Predicate IMPLIES Predicate
           | Predicate OR Predicate
           | Predicate AND Predicate
           | NOT Predicate
           | Relation
           | Expression SUBST Predicate
           | BRA Predicate KET
           | TRUE
           | FALSE
```

B.5.1 Precedence and associativity

In the above rule, the precedence of productions increases as we go down the page.

All relevant operators are left-associative except `IMPLIES` and `SUBST`, which are right-associative.

> **Note:** Terminals in this description will be familiar with the exception of `CONJ`, a low-precedence conjunction operator which the lexer may return as the result of applying some layout rule.

B.5.2 Schemas as predicates

Because the schema expression connectives `AND, OR, NOT,` etc. are lexically identical to the corresponding predicate connectives, we need a way of clarifying the user's intentions. The precedence rules given above ensure that an expression involving these connectives is parsed as a *predicate* if it can be; to force interpretation as a *schema* expression, the expression is prefixed with the `PRED` coercion operator.

B.5.3 Relation application

```
Relation = PrefixRel Expression
         | Expression PostfixRel
         | Expression InfixRel Expression {InfixRel Expression}
         | NofixRel
```

Relation applications are parsed as above, using the extended 'grammar for operators' which uses the tokens defined by the template mechanism defined in B.8 below.

B.6 Expression and schema expression

```
Expression   : EXISTS TextOrExpression DOT Expression
             | EXISTS1 TextOrExpression DOT Expression
             | FORALL TextOrExpression DOT Expression
             | MU TextOrExpression DOT Expression
             | LAMBDA TextOrExpression DOT Expression
             | Expression IFF Expression
             | Expression IMPLIES Expression
             | Expression OR Expression
             | Expression AND Expression
             | NOT Expression
             | Expression SUBST Expression
             | IF Predicate THEN Expression ELSE Expression
             | Expression COMPOSE Expression
             | Expression HIDING BRA NameList1 KET
             | Expression PROJECTION Expression
             | Expression CROSS Expression {CROSS Expression}
             | PSET Expression
             | Prefix Expression
             | Expression Postfix
             | Expression Infix Expression
             | Expression Expression
             | Expression DECORATION
             | Expression SQBRA RenameList SQKET
             | Expression SELECT Name
             | Expression SELECT NUMBER
             | THETA Expression [DECORATION]
             | Name SQBRA ExpressionList1 SQKET
             | Nofix
             | SETBRA ExpressionList SETKET
             | SETBRA TextOrExpression DOT Expression SETKET
             | SETBRA Text SETKET
             | BRA BindList KET
             | BRA Expression COMMA ExpressionList1 KET
             | BRA MU TextOrExpression KET
             | BRA Expression KET
             | STRING
             | NUMBER
             | SQBRA [Text] SQKET   % [] added, 13/9/95
```

B.6.1 Precedence and associativity

The precedence of productions increases as we go down the page.

All operators are left-associative, except `IMPLIES` and `SUBST`, which are right-associative.

In Z as defined by this standard, schemas are expressions and expressions are schemas. Among the advantages of this are the following:

- in proof, it upholds the substitutivity of equals for equals.
- in specification, it allows the user to write sensible constructions which would otherwise be banned. For example, if a function returns sets of some binding, the result of its application can be used as a normal schema expression.

The criterion for deciding whether a construction should be evaluated in a 'declaration' way or in a 'set of bindings' way is determined by the context. (We presume that for all x,y (eval-as-set-of-bindings x) = (eval-as-set-of-bindings y) iff (eval-as-declaration x) = (eval-as-declaration y)).

> **Note:** It is possible to decorate an arbitrary expression and rename an arbitrary expression (although these only make sense in type terms when the expression involves bindings).

> **Note:** Mu expressions with no dot are bracketed because the schema text objects inside have very low precedence.

B.6.2 Names

```
Name          : BRA PrefixName Arg KET
              | BRA Arg PostfixName KET
              | BRA Arg InfixName Arg KET
              | NofixName
```

B.6.3 Declarations and schema texts

> **Note:** There is no explicit abstract syntax for schema texts but it is important to have entities corresponding to schema texts at the level of the concrete syntax.
>
> We use a grammar for inline schema texts which can have any schema in a declaring position.
>
> Notice that for an arbitrary schema expression `S` the inline schema text `[S]` is not allowed – instead, the user should write simply `S`.

```
DeclPart      : DeclElem {SEMICOLON DeclElem}

DeclElem      : BasicDecl
              | Expression

BasicDecl     : NameList1 COLON Expression

TextOrExpression
              : DeclPart [ VBAR Predicate ]
```

```
Text         : DeclPart VBAR Predicate
             | DeclPart SEMICOLON DeclElem
             | BasicDecl
```

B.6.4 Binding and renaming

```
Bind       = Name DEFINE_EQUAL Expression
Rename     = Name RENAME Name
```

B.7 Lists

```
ExpressionList    = [ExpressionList1]
ExpressionList1   = Expression {COMMA Expression}

NameList   = [NameList1]
NameList1  = Name {COMMA Name}

BindList   = [BindList1]
BindList1  = Bind {COMMA Bind}

RenameList = Rename {COMMA Rename}
```

B.8 Operator templates

Introduction. Z has a number of 'mixfix' operators such as relational image. To allow definition (and re-definition) of such operators, a general method of defining mixfix operators is introduced. The specifier may write 'fixity paragraphs' which declare mixfix operators. The specifier can then use these operators in expressions.

The fixity paragraph can be viewed as a picture of the operator, with the argument slots made explicit: to declare an operator, the specifier simply presents this picture.

Example. For example, in using Z to define the semantics of a simple programming language, the specifier could write:

```
fixity while_do_ ;
fixity let_be_in_ ;
fixity for_from_to_do_ ;
fixity _becomes_ ;
fixity _equals _
```

where the lexis definition of FIXITY is `fixity`, of SEP is `;` and of ARG is `_`.

This declares that there exist the names: `(while_do_)`, `(let_be_in_)`, `(for_from_to_do_)`, `(_becomes_)` and `(_equals_)`.

It enables the specifier to use the resulting templates in an expression, such as:

```
let x be 3 in
   while
      x equals 0
   do
      y becomes y - 1
```

which is equivalent to the function application:

```
(let_be_in_) (x,
              3,
              (while_do_) ((_equals_) (x,0),
                           (_becomes_) (y,(_-_) (y,1))))
```

(where `_-_` is as defined in the toolkit).

Sequence arguments. The specifier may require that some argument slots take a comma-separated list of expressions, rather than a single expression. For example, the simple programming language could be extended with a block statement:

```
fixity begin..(NullSequence,UnitSequence,ConcatSequence)end
```

(where the lexis definition for SEQUENCE is `..`).

This declares that there exists the name `begin..end`.

The triple `(NullSequence,UnitSequence,ConcatSequence)` is a specification of a parse-time transformation to be applied to the elements of the commalist. For example:

```
begin
   a,
   b,
   c
end
```

is equivalent to

```
begin..end
   (ConcatSequence
      (ConcatSequence (UnitSequence a,UnitSequence b),
       UnitSequence c))
```

Thus the commalist argument can be made to correspond to any collection, such as a set, sequence (as here) or bag.

Precedence and associativity declarations: In order to resolve potentially ambiguous adjacent operator instances, specifiers can give a precedence and associativity to a template. For example, the `_*_` and `_-_` operators could have been declared thus:

```
fixity leftfun 10 _-_ ;
fixity leftfun 11 _*_
```

(where the lexis for LEFTFUN is `leftfun`). Then the instance:

```
1 - 2 * 3
```

would be equivalent to:

```
(_-_) (1,(_*_) (2,3))
```

In general, the meaning of chains of operator instances is resolved using the algorithm of Lalonde and Des Rivieres (see Annex G, reference [17]), taking the leftmost derivation.

Generic operators: Traditionally in Z it is possible to declare a 'generic' operator where the arguments to the operator are used both as arguments to the function application and as generic instantiations of the operator (for example, function arrows).

For example, the paragraph:

```
fixity #arrow#
```

(where the lexis for TYPE is `#`) would declare to the reader the existence of an operator `_arrow_`, and the expression

```
X arrow Y
```

would be equivalent to the function application

```
(_arrow_)[X,Y] (X,Y)
```

Relational operators It is also possible to declare relational operators, such as:

```
fixity rel _lessThan_
```

(where the lexis for REL is `rel`) which would declare to the reader the existence of an operator `_lessThan_`, and the predicate

```
x lessThan y
```

would be equivalent to the membership statement

```
(x,y) in (_lessThan_)
```

(where the lexis for MEMBER is `in`).

Restrictions on template declarations: Formally, the effect of the template declarations is achieved by changing the lexical value of the tokens which occur in those declarations.

The effect of a template declaration is:

- to assign suitable lexical status to the tokens which occur in the template.
- to assert the existence of a 'compound symbol' which represents the template.
- to assign an appropriate precedence and associativity to this compound symbol.
- to retain information about this symbol sufficient to identify it as a relation or function; to cope with its generic instantiation (if it has TYPE argument slots); and to build up sequence arguments at parse-time (if it has SEQUENCE argument slots).

The following token categories are defined (where names postfixed with `P` represent the relational version of the token used in `Predicate`)

`I, IP`	`infix token`
`PRE, PREP`	`prefix token`
`POST, POSTP`	`postfix token`
`L, LP`	`initial token`
`EL, ELP`	`initial token preceded by expression`
`ES`	`separator token preceded by expression`
`SS`	`separator token preceded by expression commalist`
`ER, ERP`	`final token preceded by expression`
`SR, SRP`	`final token preceded by expression commalist`
`ERE, EREP`	`final token preceded by expression and followed by expression`
`SRE, SREP`	`final token preceded by expression commalist and followed by expression`

Any attempt to redefine the lexical status of a token is an error.

Informative examples: Here are some example templates with descriptions of their effects on tokens (where the lexis for FIXITY is `fixity`, REL is `rel`, LEFTFUN is `leftfun`, RIGHTFUN is `rightfun`, ARG is `_`, TYPE is `#`):

```
fixity rel small_
```

The lexical value of `small` is PREP. `(small_)` is a relational operator.

```
fixity rel _isodd
```

The lexical value of `isodd` is POSTP. `(_isodd)` is a relational operator.

```
fixity rightfun 100 add_to#
```

The lexical value of `add` is L; the lexical value of `to` is ERE. `(add_to_)` is a generic function operator with one generic parameter instantiated by the second element of its argument; it is right-associative with a precedence of 100.

```
fixity rightfun 50 add..(f,g,h)to#
```

The lexical value of `add` is L; the lexical value of `to` is SRE. `(add_to_)` is a generic function operator with one generic parameter instantiated by the second element of its argument; its first argument is built out of a commalist at parse time using the zero, unit and union functions `f`, `g` and `h`; it is right-associative with a precedence of 50.

```
fixity rightfun 900 _anInfix_
```

The lexical value of `anInfix` is I. `(_anInfix_)` is a generic function operator with left-associativity and a precedence of 900.

B.8.1 Template declarations

```
Fixity        : FIXITY [Category] Template

Category      : REL
              | LEFTFUN [Prec]
              | RIGHTFUN [Prec]

Prec          : NUMBER

Template      : [Arg] NAME {SeqArg NAME} [Arg]

Arg           : NORMAL
              | TYPE

SeqArg        : Arg
              | SEQUENCE BRA Expression COMMA Expression COMMA Expression KET
```

B.8.2 Template instances as expressions

Note: It is not necesarily helpful to look at the following formal definitions in detail; they merely capture in a formal notation the informal idea that to declare a template, the specifier writes down a picture of it, and to apply a template, the specifier fills in the argument slots in the obvious way.

To help in reading the following grammars, it should be borne in mind that all templates fall into one of four categories: *prefix*, having an argument slot after the last token in the template, but no argument slot before the first token in the template; *postfix*, having an argument slot before the first token in the template, but no argument slot after the last token in the template; *infix*, having an argument slot before the first token in the template and after the last token in the template; and *nofix*, having no argument slot before the first token in the template or after the last token in the template. The grammars are organised to reflect this fact.

```
Prefix        : PreLeft {Expression ES | ExpressionList SS} PreRight
              | PRE

Postfix       : PostLeft {Expression ES | ExpressionList SS} PostRight
              | POST

Infix         : PostLeft {Expression ES | ExpressionList SS} PreRight
              | I

Nofix         : PreLeft {Expression ES | ExpressionList SS} PostRight
              | Name

PreLeft       : L
```

```
PostLeft     : EL

PreRight     : Expression ERE
             | ExpressionList SRE

PostRight    : Expression ER
             | ExpressionList SR
```

B.8.3 Template instances as predicates

```
PrefixRel    : PreRelLeft {Expression ES | ExpressionList SS} PreRelRight
             | PREP

PostfixRel   : PostRelLeft {Expression ES | ExpressionList SS} PostRelRight
             | POSTP

InfixRel     : PostRelLeft {Expression ES | ExpressionList SS} PreRelRight
             | IP
             | MEMBER
             | EQUALS

NofixRel     : PreRelLeft {Expression ES | ExpressionList SS} PostRelRight

PreRelLeft   : LP

PostRelLeft  : ELP

PreRelRight  : Expression EREP
             | ExpressionList SREP

PostRelRight : Expression ERP
             | ExpressionList SRP
```

B.8.4 Template instances as names

```
PrefixName   : PRE
             | PREP
             | PreNameLeft {Arg ES | SEQUENCE SS} PreNameRight

PostfixName  : POST
             | POSTP
             | PostNameLeft {Arg ES | SEQUENCE SS} PostNameRight

InfixName    : I
```

```
             | IP
             | PostNameLeft {Arg ES | SEQUENCE SS} PreNameRight

NofixName    : NAME
             | PreNameLeft {Arg ES | SEQUENCE SS} PostNameRight

PreNameLeft  : L
             | LP

PostNameLeft : EL
             | ELP

PreNameRight : Arg ERE
             | Arg EREP
             | SEQUENCE SRE
             | SEQUENCE SREP

PostNameRight: Arg ER
             | Arg ERP
             | SEQUENCE SR
             | SEQUENCE SRP
```

□

C Lexis – Normative Annex

Notes on this section of the Z Standard

Section title: Lexis
Note: Based on D-167 v2 and D-177, with comments from syntax subcommittee meeting 18 May 1995; also comments from Meeting 27 of the Z Standards Panel.
Section editor: Susan Stepney.
Contributions by: Rob Arthan, Randolph Johnson, Trevor King, William Milam, Mark Saaltink, Chris Sennett, ... (more to be added)
Source file: lexis.tex
Most recent update: 13th September 1995
Formatted: 14th September 1995

C.1 Introduction

The Concrete Syntax (Annex B), uses named tokens to define its terminal symbols. This section defines the lexical grammar of those tokens in terms of Z *glyphs* (defined in section C.4). This section describes a typical rendering of the tokens, showing how they might be displayed on a printed page or a graphics screen. The detailed appearance of the tokens is device-dependent. A character-based, machine-representable format, and the Interchange Format representation based on SGML are defined in sections C.4.3 and Annex D.

C.2 Soft newlines

Most white space is not recognized by a lexical analyzer, but is used as a *separator* when recognising tokens. In two special contexts some white space (called a 'hard new line') is recognized: as a `SEMICOLON` in a `DeclPart`, as a `CONJ` in a `Pred`. The rule for distinguishing 'soft' (white space) and 'hard' (recognised) newlines in these contexts is given in the following rules.

1. Tokens that can appear in these contexts are assigned to a 'soft newline category', based on whether the token could start or end a declaration or predicate.
    - BOTH: new lines are soft before and after the token, because it can neither start nor end a declaration or predicate. (It is 'infix', for example, ':')
    - AFTER: new lines are soft after the token, because it cannot end a declaration or predicate. (It is 'prefix', for example, '[')
    - BEFORE: new lines are soft before the token, because it cannot start a declaration or predicate. (It is 'postfix', for example, ']')
    - NEITHER: no new lines are soft, because such a token could start or end a declaration or predicate. (It is 'nofix', for example, '*true*')

2. When a new line appears between two tokens in the relevant context, the newline categories of both are examined. If either allows the newline to be soft in that position, it is soft, otherwise it is hard (and hence recognised).
3. All newlines are soft outside a `DeclPart` or a `Pred`. So tokens that cannot appear in these contexts can be considered to be in category BOTH.

The FIXITY paragraph allows the definition of various mixfix names, which are placed in the appropriate newline category (see section C.3.3). Other (ordinary) user declared names are 'nofix', and so are placed in NEITHER.

C.3 Tokens

```
ZToken = SIMPLE
       | BOX
       | MIXFIX
       | DECORATION
       | NAME
       | NUMBER
       ;
```

C.3.1 Simple tokens

```
SIMPLE = AND | ... | VBAR;
```

A typical rendering of these literal tokens is given below.

The third column defines the soft newline category of those tokens that can appear in the context of a DeclPart or Pred. The fourth column notes the representation syntax productions where they occur.

This representation lexis uses the typographical convention that 'keywords' are in **bold font** (except where this conflicts with previously existing conventions).

Token	**Representation**	**Newline**	**Production**
AND	$\wedge$	BOTH	Exp, Pred
BRA	(	AFTER	Exp, Pred, SeqArg
COLON	:	BOTH	BasicDecl
COMMA	,	BOTH	Exp, SeqArg, XList1
COMPOSE	$\circ\!\!\!\!\!\;\,{\scriptstyle 9}$	BOTH	Exp
CONJ	(newline)		Pred
CROSS	$\times$	BOTH	Product
DEFINEEQUAL	$:=$	BOTH	Bind, Paragraph
DOT	$\bullet$	BOTH	Exp
EQUALS	$=$	BOTH	Relation
ELSE	**else**	BOTH	Exp
EXISTS	$\exists$	AFTER	Exp, Pred
EXISTS1	$\exists_1$	AFTER	Exp, Pred

`FALSE`	*false*	NEITHER	Pred
`FIXITY`	**fixity**		Fixity
`FORALL`	∀	AFTER	Exp, Pred
`FREEBRA`	⟪		Branch
`FREEEQUALS`	::=		FreeType
`FREEKET`	⟫		Branch
`HIDING`	\	BOTH	Exp
`IF`	**if**	AFTER	Exp
`IFF`	⇔	BOTH	Exp, Pred
`IMPLIES`	⇒	BOTH	Exp, Pred
`KET`	)	BEFORE	Exp, Pred, SeqArg
`LAMBDA`	λ	AFTER	Exp
`LEFTFUN`	**leftfun**		Category
`LET`	**let**		Exp, Pred
`MEMBER`	∈	BOTH	Relation
`MU`	μ	AFTER	Exp
`NORMAL`	_		Arg
`NOT`	¬	AFTER	Exp, Pred
`OR`	∨	BOTH	Exp, Pred
`PRESCH`	pre	AFTER	Pred
`PROJECTION`	↾	BOTH	Exp
`PSET`	ℙ	AFTER	Exp
`REL`	**rel**		Category
`RENAME`	/	BOTH	Rename
`RIGHTFUN`	**rightfun**		Category
`SEMICOLON`	;	BOTH	DeclElem, DeclPart
`SEQUENCE`	...		SeqArg
`SELECT`	.	BOTH	Exp
`SETBRA`	{	AFTER	Exp
`SETKET`	}	BEFORE	Exp
`SQBRA`	[	AFTER	Exp, Formals, Paragraph
`SQKET`	]	BEFORE	Exp, Formals, Paragraph
`THEN`	**then**	BOTH	Exp
`THETA`	θ	AFTER	Exp
`TRUE`	*true*	NEITHER	Pred
`TURNSTILE`	⊢		Paragraph
`TYPE`	$		Arg
`VBAR`	|	BOTH	Paragraph, Text

C.3.2 Box tokens

```
BOX = AX | SCH | GEN     AFTER
    | IS | BAR           BOTH
    | ST                 BOTH
    | END | ENDAX        BEFORE
    ;
```

A typical rendering of these `BOX` tokens is lines drawn around the Z text.

Examples:

$$[X]$$

$$\_ \cup \_ : \mathbb{P}\, X \times \mathbb{P}\, X \longrightarrow \mathbb{P}\, X$$

$$\forall\, a, b : \mathbb{P}\, X \bullet a \cup b = \{\, x : X \mid x \in a \lor x \in b \,\}$$

$$AddName$$

$$\Delta System$$

$$n? : NAME$$

$$name' = name \cup n?$$

$$succ : \mathbb{N} \longrightarrow \mathbb{N}$$

$$\forall\, n : \mathbb{N} \bullet succ\ n = n + 1$$

The SGML Interchange Format (Annex E) defines a textual form.

C.3.3 Mixfix token categories

`MIXFIX` = `I` | ... | `SREP`

For the base language, these token categories are empty. They are populated by definitions using the template structure of the fixity paragraph, such as toolkit definitions.

The second column defines the soft newline category of names declared in these token categories. The third column notes the representation syntax productions where these tokens occur.

Token	**Newline**	**Production**
`I, IP`	BOTH	Infix
`POST, POSTP`	BEFORE	Postfix
`PRE, PREP`	AFTER	Prefix
`EL, ELP`	BOTH	Postfix, Infix
`ER, ERP`	BEFORE	Postfix, Nofix
`ERE, EREP`	BOTH	Prefix, Infix
`ES`	BOTH	Prefix, Postfix, Infix, Nofix
`L, LP`	AFTER	Prefix, Nofix
`SR, SRP`	BEFORE	Postfix, Nofix
`SS`	BOTH	Prefix, Postfix Infix
`SRE, SREP`	BOTH	Prefix, Infix

C.3.4 Decoration, name and number tokens

```
DECORATION = STROKE, {STROKE};
NAME       = WORD, {STROKE};
NUMBER     = '0' | DIGIT1, {DIGIT};
```

All these tokens are in the soft newline category NEITHER.

Notice that the lexis allows a `NAME` to include `STROKE`s, and that the concrete syntax allows an expression to be decorated with `STROKE`s. When the expression is a `NAME`, the two cases are disambiguated by white space.

Examples:

$x!$ is the `NAME` consisting of the `WORD` 'x' followed by the `STROKE` '!'

x ! is the decorated expression consisting of the `NAME` 'x' decorated with the `STROKE` '!'

$x!$! is the decorated expression consisting of the `NAME` '$x!$' decorated with the `STROKE` '!'

```
STROKE    = '′' | '!' | '?' | SUBSCRIPT;
SUBSCRIPT = DOWN, GLYPH, {DOWN, GLYPH}, UP
```

The `DOWN` and `UP` subscript delimiter tokens could be presented as in-line literals, or they could indicate a lowering/raising of the text, and possible size change. Such rendering details are not defined here.

```
ALPHASTRING  = {LETTER | DIGIT}
SYMBOLSTRING = {SYMBOL}
WORDPART     = '_', (ALPHASTRING | SYMBOLSTRING)

WORD = WORDPART, {WORDPART}
     | LETTER, ALPHASTRING, WORDPART
     | SYMBOL, SYMBOLSTRING, {WORDPART}
```

C.4 Glyphs

```
GLYPH = DIGIT | LETTER | SYMBOL | SPECIAL
```

The glyph sets forming `GLYPH` are disjoint. `SYMBOL` is a user-extensible glyph set for making new symbolic identifiers: this is where characters from other alphabets (such as Japanese or Russian) can be added.

```
DIGIT1  = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
DIGIT   = '0' | DIGIT1;
LETTER  = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm'
        | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'
        | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M'
        | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
        | 'Γ' | 'Δ' | 'Θ' | 'Λ' | 'Ξ' | 'Π' | 'Σ' | 'Υ' | 'Φ' | 'Ψ' | 'Ω'
        | 'α' | 'β' | 'γ' | 'δ' | 'ϵ' | 'ζ' | 'η' | 'θ' | 'ι' | 'κ' | 'λ' | 'μ'
        | 'ν' | 'ξ' | 'π' | 'ρ' | 'σ' | 'τ' | 'υ' | 'φ' | 'χ' | 'ψ' | 'ω'
        ;
```

`SYMBOL` = ‘\$’ | ‘,’ | ‘.’ | ‘/’ | ‘:’ | ‘;’ | ‘=’ | ‘[’ | ‘]’ | ‘{’ | ‘|’ | ‘}’
| ‘∧’ | ‘∨’ | ‘¬’ | ‘⇒’ | ‘⇔’ | ‘∀’ | ‘∃’ | ‘∈’
| ‘×’ | ‘ℙ’ | ‘•’ | ‘⟪’ | ‘⟫’ | ‘⊢’
| ‘⨾’ | ‘↾’ | ‘\’
| any toolkit glyphs, including MIXFIX (as supported)
| any user glyphs, including MIXFIX
;

SPECIAL = *BOX*
| ‘′’ | ‘!’ | ‘?’
| ‘(’ | ‘)’ | ‘_’
;

C.4.1 Examples

A glyph in the `LETTER` or `DIGIT` class may be rendered differently for reasons of emphasis or aesthetics, but it still represents the same glyph. For example, ‘*d*’, ‘d’, ‘**d**’ and ‘**d**’ are all the same glyph. This explains why those Greek characters that are identical in appearance to Roman characters do not appear twice in the list of letters.

A glyph in the `SYMBOL` class, which includes user-defined glyphs, must appear the same wherever it occurs in a specification. In particular,

schema composition ⨾ and the toolkit glyph relational composition ⨾ are different glyphs

schema projection ↾ and the toolkit glyph filter ↾ are different glyphs

schema hiding \ and the toolkit glyph set minus \ are different glyphs

Once a `NAME` has been recognised, if it consists of the glyph string corresponding to a *SIMPLE* token, it is recognised as that token instead. For example,

ℙ is recognised as *PSET*; $\mathbb{P}_1$ (‘ℙ’, `DOWN`, ‘1’, `UP`) and $\mathbb{P}_{45}$ (‘ℙ’, `DOWN`, ‘4’, `UP`, `DOWN`, ‘5’, `UP`) are recognised as `NAME`s

$\exists_1$ (‘∃’, `DOWN`, ‘1’, `UP`) is recognised as *EXISTS*1; $\exists_0$ (‘∃’, `DOWN`, ‘0’, `UP`) is recognised as a `NAME`

Underscore, ‘_’, is used in words to separate strings of alphanumerics from strings of symbols. For example,

‘ℕ⋄⋄=’ is a single word; ‘x_⋄⋄_y’ is a single word; ‘x⋄⋄y’ consists of the three words ‘x’, ‘⋄⋄’ and ‘y’

‘ℙ×’ is a single word; ‘ℙ_X’ is a single word; ‘ℙX’ is the token *PSET* followed by the word ‘X’

‘×:ℤ’ is a single word; ‘a_:_b’ is a single word; ‘a:b’ is the word ‘a’, followed by the token *COLON*, followed by the word ‘b’

C.4.2 Glyph representations

The following shows the glyphs of the Base language in a variety of formats designed for different purposes.

Spoken name A suggested form for reading the glyph out loud, designed for use in reviews, or for discussing specifications over the telephone. In the following, an English language form is given; for other natural languages, other formsmay be defined.

Interchange The format for use with the Z interchange format defined in Annex E.

Email The format for rendering the glyph on a low resolution device, such as a character-based terminal, or e-mail conversation. (The email form for digits and the Roman alphabet is the obvious one, and is not given explicitly.)

This lexis is designed primarily as a human-readable lightweight interchange format, rather than for processing by tools (the SGML Interchange Format is provided for the latter purpose). The character `%` is used to flag a special string, for example × as `%x`, and disambiguate it from, for example, the name `x`. This flag character may be omitted to reduce clutter, if there is no danger of ambiguity (for the human reader).

Mathematical The format for rendering the glyph on a high resolution device, such as a bit-mapped screen, or on paper (either hand-written, or printed).

C.4.3 Base language glyphs

Spoken name	**Interchange**	**Email**	**Mathematical**
and	&and	`/\`	$\wedge$
left [parenthesis]	(	`(`	(
colon	:	`:`	:
comma	,	`,`	,
schema compose	&scomp	`%%;`	$\fatsemi$
cross	&times	`%x`	$\times$
define equal	==	`==`	==
fat dot | spot	&bull	`@`	$\bullet$
equals	=	`=`	=
else	else	`else`	**else**
exists	&exist	`%E`	$\exists$
unique exists	&exist1	`%E1`	$\exists_1$
false	false	`false`	*false*
fixity	fixity	`fixity`	**fixity**
for all	&forall	`%A`	$\forall$
left chevron [bracket]	&lchev	`<<`	$\langle\!\langle$
free equals	::=	`::=`	::=
right chevron [bracket]	&rchev	`>>`	$\rangle\!\rangle$
hide	&hide	`%\`	$\setminus$
if	if	`if`	**if**
equivalent | if and only if	&iff	`<=>`	$\Leftrightarrow$
implies	&rArr	`==>`	$\Rightarrow$
right [parenthesis]	)	`)`	)
left function	leftfun	`leftfun`	**leftfun**
let	let	`let`	**let**
member | in	&isin	`%e`	$\in$
not	&not	`%not`	$\neg$
argument	_	`_`	_ (underscore)
or	&or	`\/`	$\vee$
coerce predicate	&pred	`pred`	**pred**
pre[condition]	&pre	`pre`	pre
project	&proj	`%|\`	$\upharpoonright$
power [set]	&pset	`%P`	$\mathbb{P}$
relation	rel	`rel`	**rel**
rename	/	`/`	/
right function	rightfun	`rightfun`	**rightfun**
semi[colon]	;	`;`	;
sequence argument	...	`...`	...
select | dot	.	`.`	.
left set [bracket]	&lcub	`{`	{
right set [bracket]	&rcub	`}`	}
left square [bracket]	&lsqb	`[`	[
right square [bracket]	&rsqb	`]`	]

then	then	`then`	**then**
true	true	`true`	*true*
turnstile	&vdash	`|-`	$\vdash$
type argument	&num	`$`	\$
bar	&verbar	`|`	|

C.4.4 Greek alphabet glyphs

Spoken name	**Interchange**	**Email**	**Mathematical**
alpha	&alpha	`%alpha`	α
beta	&beta	`%beta`	β
gamma	&gamma	`%gamma`	γ
delta	&delta	`%delta`	δ
epsilon	&epsi	`%epsilon`	ϵ
zeta	&zeta	`%zeta`	ζ
eta	&eta	`%eta`	η
theta	&thetas	`%theta`	θ
iota	&iota	`%iota`	ι
kappa	&kappa	`%kappa`	κ
lambda	&lambda	`%lambda`	λ
mu	&mu	`%mu`	μ
nu	&nu	`%nu`	ν
xi	&xi	`%xi`	ξ
pi	&pi	`%pi`	π
rho	&rho	`%rho`	ρ
sigma	&sigma	`%sigma`	σ
tau	&tau	`%tau`	τ
upsilon	&upsi	`%upsilon`	υ
phi	&phis	`%phi`	ϕ
chi	&chi	`%chi`	χ
psi	&psi	`%psi`	ψ
omega	&omega	`%omega`	ω
big delta	&Delta	`%Delta`	Δ
big gamma	&Gamma	`%Gamma`	Γ
big theta	&Theta	`%Theta`	Θ
big lambda	&Lambda	`%Lambda`	Λ
big xi	&Xi	`%Xi`	Ξ
big pi	&Pi	`%Pi`	Π
big sigma	&Sigma	`%Sigma`	Σ
big upsilon	&Upsi	`%Upsilon`	Υ
big phi	&Phi	`%Phi`	Φ
big psi	&Psi	`%Psi`	Ψ
big omega	&Omega	`%Omega`	Ω

C.4.5 Box and Decoration glyphs

Token	**Email**
`AX`	`+..`
`SCH`	`+--`
`GEN`	`+==`
`IS`	`---`
`BAR`	`===`
`ST`	`|--`
`END`	`---`
`ENDAX`	`-..`
`DOWN`	`%_`
`UP`	`%^`

Examples:

```
+== [X] ===
  _ %u _ : %P X %x %P X --> %P X
|--
  %A a,b: %P X @ a %u b = { x:X | x %e a \/ x %e b }
-==
```

```
+-- AddName ---
  %Delta System
  n? : NAME
|--
  name´ = name %u n?
---
```

```
+..
  succ : %N --> %N
|--
  %A n : %N @ succ n = n + 1
-..
```

C.5 Toolkit glyphs

Spoken name	**Interchange**	**Email**	**Mathematical**
not equal	&ne	`/=`	$\neq$
not in	&notin	`%/e`	$\notin$
empty [set]	&empty	`(/)`	$\emptyset$
subset	&sube	`%c_`	$\subseteq$
proper subset	&sub	`%c`	$\subset$
[set] union	&cup	`%u`	$\cup$
[set] intersection	&cap	`%n`	$\cap$
set difference	&sdiff	`\`	$\setminus$
generalised union	&Bigcup	`%uu`	$\bigcup$
generalised intersection	&Bigcap	`%nn`	$\bigcap$
finite sets	&fset	`%F`	$\mathbb{F}$
relation	&rel	`<-->`	$\leftrightarrow$
maplet	&map	`|-->`	$\mapsto$
[relational] compose	&rcomp	`%;`	⨾
functional compose	&compfn	`%o`	$\circ$
domain restrict	&dres	`<:`	$\lhd$
range restrict	&rres	`:>`	$\rhd$
domain subtract	&dsub	`<-:`	⩤
range subtract	&rsub	`:->`	⩥
inverse	&tilde	`~`	$\sim$
left relational image bracket	&limg	`(|`	⦇
right relational image bracket	&rimg	`|)`	⦈
transitive closure | plus	&tcl	`%+`	$^{+}$
reflexive transitive closure | star	&rtcl	`%*`	*
[relational] override	&oplus	`(+)`	$\oplus$
[partial] function	&pfun	`-|->`	⇸
total function	&rarr	`-->`	$\longrightarrow$
[partial] injection	&pinj	`>-|->`	⤔
total injection	&rarrtl	`>-->`	$\rightarrowtail$
[partial] surjection	&psur	`-|->>`	⤀
total surjection	&Rarr	`-->>`	$\twoheadrightarrow$
bijection	&bij	`>-->>`	⤖
finite function	&fpfun	`-||->`	⇻
finite injection	&fpinj	`>-||->`	⤕
nat[ural number]	&nat	`%N`	$\mathbb{N}$
integer	&int	`%Z`	$\mathbb{Z}$
less than	&lt	`<`	$<$
less than or equal to	&le	`<=`	$\leq$
greater than	&gt	`>`	$>$
greater than or equal to	&ge	`>=`	$\geq$

up to	&nldr	`..`	$..$
plus	+	`+`	$+$
[unary] minus	&uminus	`-`	$-$
[binary] minus	&bminus	`-`	$-$
times	*	`*`	$*$
cardinality | hash	&num	`#`	$\#$
left seq[uence bracket]	&lang	`%<`	$\langle$
right seq[uence bracket]	&rang	`%>`	$\rangle$
filter	&filter	`|\`	$\upharpoonright$
extract	&extract	`/|`	$\upharpoonleft$
concatenate	&frown	`^`	$\frown$
left bag [bracket]	&lbag	`[|`	$\llbracket$
right bag [bracket]	&rbag	`|]`	$\rrbracket$

□

D Mathematical toolkit – Normative Annex

Notes on this section of the Z Standard

Section title: Mathematical toolkit
Section editor: John Wordsworth
Contributions by: John Wordsworth, Susan Stepney, ...*(others to be added)*
Source file: toolkit.tex
Most recent update: 13th September 1995
Formatted: 14th September 1995

Introduction

This section defines a Mathematical Toolkit or Library for use with the Z notation. The principle is that those constructions that can be defined in terms of others are included in the Toolkit rather than in the core notation—this simplifies the core notation.

Most users will want to make use of the constructions defined in this section. This can therefore be regarded as a *basic* Toolkit, which users may augment with their own definitions, or replace if these definitions are not suitable for their use.

In this version of the Z Standard, the list of defined items follows the customary list of Toolkit items. Later versions of the Standard may include further definitions and explanations, and will link the Toolkit to related work on the semantics and proof system for Z.

Definitions of the Mathematical Toolkit are informally explained and illustrated. In some cases an illustration for one part of the Toolkit may rely on terms defined earlier in the toolkit. Many of the definitions given here are generic with respect to one or more sets.

Editor's note: The following note appeared an an earlier version. It is retained for possible use.

> Instantiation of a generic definition can be performed with any appropriate sets, not necessarily the maximal sets of their types. However the informal descriptions of these definitions are often here expressed as if the sets used for instantiation were in fact types, since that is the way in which these definitions are commonly instantiated in Z specifications.
>
> Reviewers of the draft standard are invited to comment on this approach.

D.1 Sets

Name

$\neq$	inequality
$\notin$	non-membership

Definition

fixity rel $\_ \neq \_$
fixity rel $\_ \notin \_$

$$[X]$$

$$\_ \neq \_ : X \leftrightarrow X$$
$$\_ \notin \_ : X \leftrightarrow \mathbb{P}\, X$$

$$\forall\, x, y : X \bullet x \neq y \Leftrightarrow \neg\, (x = y)$$

$$\forall\, x : X ; S : \mathbb{P}\, X \bullet x \notin S \Leftrightarrow \neg\, (x \in S)$$

Description

Inequality is a relation between values of the same type. The predicate $x \neq y$ denotes true when $x = y$ denotes false.

Non-membership is a relation between values of a certain type and sets of values of that type. The predicate $x \notin S$ denotes true when $x \in S$ denotes false.

Name

$\varnothing$	empty set
$\subseteq$	subset relation
$\subset$	proper subset relation
$\mathbb{P}_1$	non-empty subsets

Definition

$$\varnothing[X] == \{\, x : X \mid \mathit{false} \,\}$$

fixity rel $\_ \subseteq \_$
fixity rel $\_ \subset \_$

$$\begin{array}{|l}
[X] \\
\hline
\_ \subseteq \_, \_ \subset \_ : \mathbb{P}\, X \leftrightarrow \mathbb{P}\, X \\
\hline
\forall S, T : \mathbb{P}\, X \bullet \\
\qquad (S \subseteq T \Leftrightarrow (\forall x : X \bullet x \in S \Rightarrow x \in T)) \land \\
\qquad (S \subset T \Leftrightarrow S \subseteq T \land S \neq T) \\
\hline
\end{array}$$

fixity leftfun 100 $\mathbb{P}_1\ \$$

$$\mathbb{P}_1\, X == \{\, S : \mathbb{P}\, X \mid S \neq \varnothing \,\}$$

Description

The empty set of values of a certain type is the set of values of that type that has no members.

If S and T are sets of values of the same type, then $S \subseteq T$ is a predicate denoting true if and only if every member of S is a member of T. The empty set of values of a certain type is a subset of every set of values of that type.

If S and T are sets of values of the same type, then $S \subset T$ is a predicate denoting true if and only if every member of S is a member of T and S and T are not equal. If S is a proper subset of T, then it is also a subset of T. The empty set of values of a certain type is a proper subset of every non-empty set of values of that type.

If X is a set, then $\mathbb{P}_1\, X$ is the set of all non-empty subsets of X. $\mathbb{P}_1\, X$ is a proper subset of $\mathbb{P}\, X$.

Name

$\cup$	set union
$\cap$	set intersection
$\setminus$	set difference

Definition

fixity leftfun 30 $\_ \cup \_$
fixity leftfun 40 $\_ \cap \_$
fixity leftfun 50 $\_ \setminus \_$

$$[X]$$

$$\_ \cup \_, \_ \cap \_, \_ \setminus \_ : \mathbb{P}\, X \times \mathbb{P}\, X \longrightarrow \mathbb{P}\, X$$

$$\begin{aligned}
&\forall\, S, T : \mathbb{P}\, X \bullet \\
&\qquad S \cup T = \{\, x : X \mid x \in S \lor x \in T \,\} \land \\
&\qquad S \cap T = \{\, x : X \mid x \in S \land x \in T \,\} \land \\
&\qquad S \setminus T = \{\, x : X \mid x \in S \land x \notin T \,\}
\end{aligned}$$

Description

The union of two sets of values of the same type is the set of values that are members of either set.

The intersection of two sets of values of the same type is the set of values that are members of both sets.

The difference of two sets of values of the same types is the set of values that are members of the first set but not members of the second.

Name

$\bigcup$	generalized union
$\bigcap$	generalized intersection

Definition

$[X]$

$$\bigcup, \bigcap : \mathbb{P}(\mathbb{P}\,X) \longrightarrow \mathbb{P}\,X$$

$$\begin{array}{l} \forall\, A : \mathbb{P}(\mathbb{P}\,X) \bullet \\ \qquad \bigcup A = \{\, x : X \mid (\exists\, S : A \bullet x \in S) \,\} \land \\ \qquad \bigcap A = \{\, x : X \mid (\forall\, S : A \bullet x \in S) \,\} \end{array}$$

Description

The generalised union of a set of sets of values of the same type is the set of values of that type that are members of at least one of the sets.

The generalised intersection of a set of sets of values of the same type is the set of values of that type that are members of every one of the sets.

Name

first, *second*	projection functions for ordered pairs

Definition

$$
\begin{array}{|l}
\hline\hline
[X, Y] \\
\hline\hline
\mathit{first} : X \times Y \longrightarrow X \\
\mathit{second} : X \times Y \longrightarrow Y \\
\hline
\forall x : X; y : Y \bullet \\
\qquad \mathit{first}(x, y) = x \land \\
\qquad \mathit{second}(x, y) = y \\
\hline
\end{array}
$$

Description

For any ordered pair (x, y), $\mathit{first}(x, y)$ is x and $\mathit{second}(x, y)$ is y.

If p is of type $X \times Y$, then $p = (\mathit{first}\, p, \mathit{second}\, p)$.

D.2 Relations

Name

$\longleftrightarrow$	binary relations
$\mapsto$	maplet

Definition

fixity rightfun $5\ \$\longleftrightarrow\$$

$$X \longleftrightarrow Y == \mathbb{P}(X \times Y)$$

fixity leftfun $10\ \_\mapsto\_$

$$\begin{array}{|l}
\hline\hline [X, Y] \\
\_\mapsto\_ : X \times Y \longrightarrow X \times Y \\
\hline
\forall\, x : X;\ y : Y \bullet \\
\qquad x \mapsto y = (x, y) \\
\hline
\end{array}$$

Description

$X \longleftrightarrow Y$ is the set of all sets of ordered pairs whose first members are members of X and whose second members are members of Y. To declare $R : X \longleftrightarrow Y$ is to say that R is such a set of ordered pairs.

The maplet forms an ordered pair from two values, so if x is of type X and y is of type Y, then $x \mapsto y$ is of type $X \times Y$. $x \mapsto y$ is thus just another notation for (x, y).

Name

dom, ran	domain and range of a relation

Definition

$$
\begin{array}{|l}
[X, Y] \\
\hline
\mathrm{dom} : (X \leftrightarrow Y) \longrightarrow \mathbb{P}\, X \\
\mathrm{ran} : (X \leftrightarrow Y) \longrightarrow \mathbb{P}\, Y \\
\hline
\forall R : X \leftrightarrow Y \bullet \\
\qquad \mathrm{dom}\, R = \{\, x : X;\ y : Y \mid (x \mapsto y) \in R \bullet x \,\} \land \\
\qquad \mathrm{ran}\, R = \{\, x : X;\ y : Y \mid (x \mapsto y) \in R \bullet y \,\} \\
\hline
\end{array}
$$

Description

The domain of a relation R is the set of first members of the ordered pairs in R. If R is of type $X \leftrightarrow Y$, the domain of R is of type $\mathbb{P}\, X$. If R is an empty relation, then its domain is an empty set.

The range of a relation R is the set of second members of the ordered pairs in R. If R is of type $X \leftrightarrow Y$, the domain of R is of type $\mathbb{P}\, Y$. If R is an empty relation, then its range is an empty set.

Name

id	identity relation
$\fatsemi$	relational composition
$\circ$	backward relational composition

Definition

fixity leftfun id \$

$$\mathrm{id}\, X == \{\, x : X \bullet x \mapsto x \,\}$$

fixity leftfun 40 $\_ \fatsemi \_$
fixity leftfun 40 $\_ \circ \_$

$$
\begin{array}{l}
[X, Y, X] \\
\hline
\_ \fatsemi \_ : (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \longrightarrow (X \leftrightarrow Z) \\
\_ \circ \_ : (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \longrightarrow (X \leftrightarrow Z) \\
\hline
\forall R : X \leftrightarrow Y; S : Y \leftrightarrow Z \bullet \\
\qquad R \fatsemi S = S \circ R = \{\, x : X; y : Y; z : Z \mid \\
\qquad\qquad\qquad\qquad (x \mapsto y) \in R \land (y \mapsto z) \in S \bullet x \mapsto z \,\}
\end{array}
$$

Description

The identity relation on a set X is the relation that relates every member of X to itself. Its type is $X \leftrightarrow X$. The identity relation on an empty set is an empty relation.

The relational composition of a relation $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$ is a relation of type $X \leftrightarrow Z$ formed by taking all the pairs (x, y) of R whose second members are in the domain of S, and relating x to every member of Z that y is related to by S.

The backward composition of S and R is the same as the composition of R and S.

Name

$\lhd$ domain restriction

$\rhd$ range restriction

Definition

fixity leftfun 60 $\_ \lhd \_$
fixity leftfun 60 $\_ \rhd \_$

$$[X, Y]$$

$$\begin{array}{l}
\_ \lhd \_ : \mathbb{P}\, X \times (X \leftrightarrow Y) \longrightarrow (X \leftrightarrow Y) \\
\_ \rhd \_ : (X \leftrightarrow Y) \times \mathbb{P}\, Y \longrightarrow (X \leftrightarrow Y)'
\end{array}$$

$$\begin{array}{l}
\forall\, S : \mathbb{P}\, X;\ R : X \leftrightarrow Y \bullet \\
\qquad S \lhd R = \{\, x : X;\ y : Y \mid x \in S \land (X \mapsto y) \in R \bullet x \mapsto y \,\} \\
\forall\, R : X \leftrightarrow Y;\ T : \mathbb{P}\, Y \bullet \\
\qquad R \rhd T = \{\, x : X;\ y : Y \mid (x \mapsto y) \in R \land y \in T \bullet x \mapsto y \,\}
\end{array}$$

Description

The domain restriction of a relation $R : X \leftrightarrow Y$ by a set $S : \mathbb{P}\, X$ is the set of pairs in R whose first members are in S. $S \lhd R$ is a subset of R, and its domain is a subset of S.

The range restriction of a relation $R : X \leftrightarrow Y$ by a set $T : \mathbb{P}\, Y$ is the set of pairs in R whose second members are in T. $R \rhd T$ is a subset of R, and its range is a subset of T.

Name

$⩤$	domain anti-restriction
$⩥$	range anti-restriction

Definition

fixity leftfun 60 $\_ ⩤ \_$
fixity leftfun 60 $\_ ⩥ \_$

$$
\begin{array}{|l}
[X, Y] \\
\hline
\_ ⩤ \_ : \mathbb{P}\, X \times (X \leftrightarrow Y) \longrightarrow (X \leftrightarrow Y) \\
\_ ⩥ \_ : (X \leftrightarrow Y) \times \mathbb{P}\, Y \longrightarrow (X \leftrightarrow Y) \\
\hline
\forall S : \mathbb{P}\, X; R : X \leftrightarrow Y \bullet \\
\qquad S ⩤ R = \{\, x : X; y : Y \mid x \notin S \land (x \mapsto y) \in R \bullet x \mapsto y \,\} \\
\forall R : X \leftrightarrow Y; T : \mathbb{P}\, Y \bullet \\
\qquad R ⩥ T = \{\, x : X; y : Y \mid (x \mapsto y) \in R \land y \notin T \bullet x \mapsto y \,\} \\
\hline
\end{array}
$$

Description

The domain anti-restriction of a relation $R : X \leftrightarrow Y$ by a set $S : \mathbb{P}\, X$ is the set of pairs in R whose first members are not in S. $S ⩤ R$ is a subset of R, and its domain contains no members of S.

The range anti-restriction of a relation $R : X \leftrightarrow Y$ by a set $T : \mathbb{P}\, Y$ is the set of pairs in R whose second members are not in T. $R ⩥ T$ is a subset of R, and its range contains no members of T.

Name

$\_^{\sim}$ relational inversion

Definition

fixity leftfun 100 $\_^{\sim}$

$$
\begin{array}{|l}
[X, Y] \\
\_^{\sim} : (X \leftrightarrow Y) \longrightarrow (Y \leftrightarrow X) \\
\hline
\forall R : X \leftrightarrow Y \bullet \\
\qquad R^{\sim} = \{ x : X; y : Y \mid (x \mapsto y) \in R \bullet y \mapsto x \} \\
\hline
\end{array}
$$

Description

The inverse of a relation is the relation obtained by reversing every ordered pair in the relation.

Name

$\_\llparenthesis\_\rrparenthesis$ relational image

Definition

fixity leftfun 100 $\_\llparenthesis\_\rrparenthesis$

$$
\begin{array}{|l}
[X, Y] \\
\hline
\_\llparenthesis\_\rrparenthesis : (X \leftrightarrow Y) \times \mathbb{P}\, X \rightarrow \mathbb{P}\, Y \\
\hline
\forall R : X \leftrightarrow Y; S : \mathbb{P}\, X \bullet \\
\qquad R\llparenthesis S\rrparenthesis = \{\, x : X; y : Y \mid x \in S \land (x \mapsto y) \in R \bullet y \,\} \\
\hline
\end{array}
$$

Description

The relational image of a set $S : \mathbb{P}\, X$ under a relation $R : X \leftrightarrow Y$ is the set of values of type Y that are related under R to a value in S.

Name

$\_^{+}$	transitive closure
$\_^{*}$	reflexive-transitive closure

Definition

fixity leftfun 100 $\_^{+}$
fixity leftfun 100 $\_^{*}$

$$
\begin{array}{|l}
\hline
[X] \\
\_^{+}, \_^{*} : (X \leftrightarrow X) \longrightarrow (X \leftrightarrow X) \\
\hline
\forall R : X \leftrightarrow X \bullet \\
\qquad R^{+} = \bigcap \{\, Q : X \leftrightarrow X \mid R \subseteq Q \land Q \mathbin{;} Q \subseteq Q \,\} \land \\
\qquad R^{*} = \bigcap \{\, Q : X \leftrightarrow X \mid \mathrm{id}\, X \subseteq Q \land R \subseteq Q \land Q \mathbin{;} Q \subseteq Q \,\} \\
\hline
\end{array}
$$

Description

The transitive closure of a relation $R : X \leftrightarrow X$ is the relation obtained by relating each member of the domain of R to its images under R, and to anything related to any of its images under R by any number of steps of application of R.

The reflexive transitive closure of a relation $R : X \leftrightarrow X$ is the relation formed by extending the transitive closure of R by the identity relation on X.

D.3 Functions

Name

$\pfun$	partial functions
$\longrightarrow$	total functions

Definition

fixity rightfun 5 \$ $\pfun$ \$
fixity rightfun 5 \$ $\longrightarrow$ \$

$$
\begin{aligned}
&X \pfun Y == \\
&\quad \{ f : X \longleftrightarrow Y \mid (\forall x : X; y_1, y_2 : Y \bullet \\
&\qquad\qquad (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \} \\
&X \longrightarrow Y == \{ f : X \pfun Y \mid \operatorname{dom} f = X \}
\end{aligned}
$$

Description

The partial functions from X to Y are a subset of the relations $X \longleftrightarrow Y$. They are distinguished by the property that each x in X is related to at most one y in Y. $X \pfun Y$ is the set of all partial functions from X to Y, and to declare $f : X \pfun Y$ is to say that f is one such partial function.

The total functions from X to Y are a subset of the partial functions $X \pfun Y$. They are distinguished by the property that each x in X is related to exactly one y in Y. $X \longrightarrow Y$ is the set of all total functions from X to Y, and to declare $f : X \longrightarrow Y$ is to say that f is one such total function. The domain of $f : X \longrightarrow Y$ is X.

Name

$\rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow$	partial injections
$\rightarrowtail$	total injections

Definition

fixity rightfun 5 \$ $\rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow$ \$
fixity rightfun 5 \$ $\rightarrowtail$ \$

$$X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y ==$$
$$\{ f : X \mathrel{+\!\!\!\!\!\!\rightarrow} Y \mid (\forall x_1, x_2 : \operatorname{dom} f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$$

$$X \rightarrowtail Y == (X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y) \cap (X \longrightarrow Y)$$

Description

The partial injections from X to Y are a subset of the partial functions $X \mathrel{+\!\!\!\!\!\!\rightarrow} Y$. They are distinguished by the property that each y in Y is related to at most one x in X. Thus the inverse of a partial injection is also a partial injection. $X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y$ is the set of all partial injections from X to Y, and to declare $f : X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y$ is to say that f is one such partial injection.

The total injections from X to Y are a subset of the partial injections $X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y$. They are distinguished by the property that each x in X is related to exactly one y in Y. $X \rightarrowtail Y$ is the set of all total injections from X to Y, and to declare $f : X \rightarrowtail Y$ is to say that f is one such total injection.

Name

$\psurj$	partial surjections
$\surj$	total surjections
$\bij$	bijections

Definition

fixity rightfun 5 \$ $\psurj$ \$
fixity rightfun 5 \$ $\surj$ \$
fixity rightfun 5 \$ $\bij$ \$

$$X \psurj Y == \{ f : X \pfun Y \mid \operatorname{ran} f = Y \}$$
$$X \surj Y == (X \psurj Y) \cap (X \fun Y)$$
$$X \bij Y == (X \surj Y) \cap (X \inj Y)$$

Description

The partial surjections from X to Y are a subset of the partial functions $X \pfun Y$. They are distinguished by the property that each y in Y is related to at least one x in X. $X \psurj Y$ is the set of all partial surjections from X to Y, and to declare $f : X \psurj Y$ is to say that f is one such partial surjection.

The total surjections from X to Y are a subset of the partial surjections $X \psurj Y$. They are distinguished by the property that each x in X is related to exactly one y in Y. $X \surj Y$ is the set of all total surjections from X to Y, and to declare $f : X \surj Y$ is to say that f is one such total surjection.

The bijections from X to Y are a subset of the total surjections $X \surj Y$. They are distinguished by the property that each y in Y is related to exactly one x in X. $X \bij Y$ is the set of all bijections from X to Y, and to declare $f : X \bij Y$ is to say that f is one such total bijection.

D.3.1 Name

$\oplus$ functional over-riding

D.3.2 Definition

fixity leftfun 50 $\_ \oplus \_$

$$
\begin{array}{|l}
[X, Y] \\
\hline
\_ \oplus \_ : (X \pfun Y) \times (X \pfun Y) \longrightarrow (X \pfun Y) \\
\hline
\forall f, g : X \pfun Y \bullet \\
\qquad f \oplus g = ((\mathrm{dom}\, g) \lhd f) \cup g \\
\hline
\end{array}
$$

Description

If f and g are both functions from X to Y, then the functional overriding of f by g is the function g together with such pairs of f as have first elements different from the first element of any pair in g.

D.4 Numbers and finiteness

Name

$\mathbb{N}$	natural numbers
$\mathbb{Z}$	integers
$-$	prefix minus
$+, -, *, \mathsf{div}, \mathsf{mod}$	arithmetic operations
$<, \leq, \geq, >$	numerical comparison

Definition

fixity leftfun 30 $\_ + \_$
fixity leftfun 30 $\_ - \_$
fixity leftfun 40 $\_ * \_$
fixity leftfun 40 $\_ \mathsf{div} \_$
fixity leftfun 40 $\_ \mathsf{mod} \_$
fixity leftfun 50 $- \_$
fixity rel $\_ < \_$
fixity rel $\_ \leq \_$
fixity rel $\_ \geq \_$
fixity rel $\_ > \_$

$[number]$

$$
\begin{array}{|l}
\mathbb{Z}\mathbb{N} : \mathbb{P}\, number \\
\\
\_ + \_, \_ - \_, \_ * \_ : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z} \\
\_ \mathsf{div} \_, \_ \mathsf{mod} \_ : \mathbb{Z} \times (\mathbb{Z} \setminus \{0\}) \longrightarrow \mathbb{Z} \\
-\_ : \mathbb{Z} \longrightarrow \mathbb{Z} \\
\\
\_ < \_, \_ \leq \_, \_ \geq \_, \_ > \_ : \mathbb{Z} \leftrightarrow \mathbb{Z} \\
\hline
\mathbb{N} = \{\, n : \mathbb{Z} \mid n \geq 0 \,\} \\
\\
\text{(other definitions to be added)}
\end{array}
$$

Description

The natural numbers are the integers from zero upwards.

The type of $\mathbb{N}$ is $\mathbb{P}\,\mathbb{Z}$, since $\mathbb{N}$ is a set of integers.

The declaration $n : \mathbb{N}$ makes $\mathbb{Z}$ the type of n, and entails the property $n \geq 0$.

Name

$\mathbb{N}_1$	strictly positive integers
succ	successor function

Definition

$$\mathbb{N}_1 == \mathbb{N} \setminus \{0\}$$

$$\begin{array}{|l}
succ : \mathbb{N} \longrightarrow \mathbb{N} \\
\hline
\forall\, n : \mathbb{N} \bullet succ(n) = n + 1
\end{array}$$

Description

The strictly positive numbers $\mathbb{N}$ are the natural numbers except zero.

The successor of any natural number is the next natural number in ascending order.

Name

iter k R	iteration

Definition

$$
\begin{array}{|l}
=[X]= \\
\mathit{iter} : \mathbb{Z} \longrightarrow (X \leftrightarrow X) \longrightarrow (X \leftrightarrow X) \\
\hline
\forall R : X \leftrightarrow X \bullet \\
\quad \mathit{iter}\ 0\ R = \mathrm{id}\ X \land \\
\quad (\forall k : \mathbb{N} \bullet \mathit{iter}\ (k+1) R = R \mathbin{;} (\mathit{iter}\ k\ R)) \land \\
\quad (\forall k : \mathbb{N} \bullet \mathit{iter}\ (-k) R = \mathit{iter}\ k\ (R^{\sim}))
\end{array}
$$

Description

The iteration of a relation $R : X \leftrightarrow X$ by zero is the identity relation on the set X.

The iteration of a relation $R : X \leftrightarrow X$ by one is the relation R.

The iteration of a relation $R : X \leftrightarrow X$ by an integer greater than one is the composition of R with its iteration by the next lower integer.

The iteration of a relation $R : X \leftrightarrow X$ by an integer less that zero is the iteration of the inverse of R by the corresponding positive integer. Thus the iteration of R by -1 is the inverse of R.

Name

$..$ number range

Definition

fixity leftfun 20 $\_\,..\,\_$

$$\begin{array}{|l}
\_\,..\,\_ : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{P}\,\mathbb{Z} \\
\hline
\forall\, a, b : \mathbb{Z} \bullet \\
\qquad a\,..\,b = \{\, k : \mathbb{Z} \mid a \leq k \leq b \,\}
\end{array}$$

Description

If a and b are integers, and a is less than b, the number range $a..b$ contains a, b and any integers between.

If a is equal to b, the number range $a..b$ is a singleton set containing a only.

If a is greater than b, the number range $a..b$ is an empty set of integers.

The number range $a..b$ is always finite, and if $b \geq a$ its size is $b - a + 1$.

Name

$\mathbb{F}$	finite sets
$\mathbb{F}_1$	non-empty finite sets
$\#$	number of members of a set

Definition

fixity leftfun 100 $\mathbb{F}\ \$$
fixity leftfun 100 $\mathbb{F}_1\ \$$

$$\begin{aligned} \mathbb{F}\, X &== \{\, S : \mathbb{P}\, X \mid \exists\, n : \mathbb{N} \bullet \exists\, f : 1 \mathinner{..} n \longrightarrow S \bullet \operatorname{ran} f = S \,\} \\ \mathbb{F}_1\, X &== \mathbb{F}\, X \setminus \{\varnothing\} \end{aligned}$$

$$\begin{array}{|l} [X] \\ \hline \# : \mathbb{F}\, X \longrightarrow \mathbb{N} \\ \hline \forall\, S : \mathbb{F}\, X \bullet \\ \qquad \# S = (\mu\, n : \mathbb{N} \mid (\exists\, f : 1 \mathinner{..} n \rightarrowtail S \bullet \operatorname{ran} f = S)) \\ \hline \end{array}$$

Description

A set is finite if its members can be put into one-to-one correspondence with the natural numbers from 1 up to some limit. $\mathbb{F}\, X$ is the set of all finite subsets of X. $\mathbb{F}\, X$ is a subset of $\mathbb{P}\, X$. If X is finite, then it is a member of $\mathbb{F}\, X$.

The non-empty finite subsets of X are the finite subsets of X except the empty set.

The number of members of a finite set is the upper limit of the number range starting with 1 that can be put into one-to-one correspondence with the members of the set.

Name

$\ffun$	finite partial functions
$\finj$	finite partial injections

Definition

fixity rightfun 5 \$ $\ffun$ \$
fixity rightfun 5 \$ $\finj$ \$

$$X \ffun Y == \{ f : X \pfun Y \mid \dom f \in \mathbb{F} X \}$$
$$X \finj Y == (X \ffun Y) \cap (X \pinj Y)$$

Description

The finite partial functions from X to Y are the partial functions from X to Y whose domains are finite sets.

The finite partial injections from X to Y are the partial injections from X to Y whose domains are finite sets.

Name

min, *max*	minimum and maximum of a set of numbers

Definition

$$
\begin{array}{|l}
min : \mathbb{P}_1\,\mathbb{Z} \pfun \mathbb{Z} \\
max : \mathbb{P}_1\,\mathbb{Z} \pfun \mathbb{Z} \\
\hline
min \;=\; \{\, S : \mathbb{P}_1\,\mathbb{Z};\ m : \mathbb{Z} \mid \\
\qquad\qquad m \in S \land (\forall\, n : S \bullet m \leq n) \bullet S \mapsto m \,\} \\[1ex]
max \;=\; \{\, S : \mathbb{P}_1\,\mathbb{Z};\ m : \mathbb{Z} \mid \\
\qquad\qquad m \in S \land (\forall\, n : S \bullet m \geq n) \bullet S \mapsto m \,\}
\end{array}
$$

Description

The minimum of a non-empty set of integers that has a least member is the least member. Sets of integers that have no least member are not in the domain of min. If $a \leq b$, $\min a..b = a$.

The maximum of a non-empty set of integers that has a greatest member is the greatest member. Sets of integers that have no greatest member are not in the domain of max. If $a \leq b$, $\max a..b = b$.

D.5 Sequences

Name

seq	finite sequences
seq_1	non-empty finite sequences
iseq	injective sequences

Definition

$$
\begin{array}{l}
\textbf{fixity leftfun}\ \ 100\ \ \$\,\mathrm{seq}\ \$ \\
\textbf{fixity leftfun}\ \ 100\ \ \$\,\mathrm{seq}_1\ \$ \\
\textbf{fixity leftfun}\ \ 100\ \ \$\,\mathrm{iseq}\ \$
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{seq}\,X & == \{\, f : \mathbb{N} ⇻ X \mid \mathrm{dom}\, f = 1 \mathrel{..} \# f \,\} \\
\mathrm{seq}_1\,X & == \{\, f : \mathrm{seq}\,X \mid \# f > 0 \,\} \\
\mathrm{iseq}\,X & == \mathrm{seq}\,X \cap (\mathbb{N} ⤔ X)
\end{array}
$$

Description

A sequence is a finite aggregate of values of the same type in which each value can be identified by its position in the sequence. The formal definition establishes a sequence as a partial function relating the numbers from the set $1..n$ for some n (the domain of the sequence) to the values (the range of the sequence). $\mathrm{seq}\,X$ is the set of all finite sequences of values of type X. The declaration $S : \mathrm{seq}\,X$ says that S is one such finite sequence. Since a sequence is a *function* (i.e. a set of ordered pairs), a sequence might be empty, and the function application notation $S\ i$ can be used to denote the element at position i, provided that i is in the domain of the sequence.

$\mathrm{seq}_1\,X$ is the set of all non-empty finite sequences of values of type X. The declaration $s : \mathrm{seq}_1\,X$ says that s is such a non-empty finite sequence. $\mathrm{seq}_1\,X$ is a subset of $\mathrm{seq}\,X$.

$\mathrm{iseq}\,X$ is the set of all injective finite sequences of values of type X. A sequence is injective if no value appears more than once in the sequence. The declaration $S : \mathrm{iseq}\,X$ says that S is such an injective finite sequence. $\mathrm{iseq}\,X$ is a subset of $\mathrm{seq}\,X$.

Name

$\langle\ \rangle$	sequence brackets
$\frown$	concatenation

Definition

fixity leftfun $10\ \_ \frown \_$

$$
\begin{array}{|l}
\hline\hline [X] \\
\_ \frown \_ : \mathrm{seq}\, X \times \mathrm{seq}\, X \longrightarrow \mathrm{seq}\, X \\
\hline
\forall\, s, t : \mathrm{seq}\, X \bullet \\
\qquad s \frown t = s \cup \{\, n : \mathrm{dom}\, t \bullet n + \# s \mapsto t(n) \,\} \\
\hline
\end{array}
$$

Description

The brackets $\langle$ and $\rangle$ can be used for enumerated sequences. The empty enumeration is the empty function. A singleton enumeration is the function that maps 1 to the element in the enumeration. The function that extends an enumeration is the concatenation function.

Concatenation is a function of a pair of sequences of values of the same type that denotes a sequence that begins with the first sequence and continues with the second. Either or both of the sequences might be empty. If either sequence is empty, the result is the other sequence.

Name

head, *last*, *tail*, *front*   sequence decomposition

Definition

$$
\begin{array}{l}
[X] \\
\hline
\mathit{head}, \mathit{last} : \mathrm{seq}_1\, X \longrightarrow X \\
\mathit{tail}, \mathit{front} : \mathrm{seq}_1\, X \longrightarrow \mathrm{seq}\, X \\
\hline
\forall\, s : \mathrm{seq}_1\, X \bullet \\
\quad \mathit{head}\ s = s(1) \land \\
\quad \mathit{last}\ s = s(\#s) \land \\
\quad \mathit{tail}\ s = (\lambda\, n : 1 \mathinner{\ldotp\ldotp} \#s - 1 \bullet s(n+1)) \land \\
\quad \mathit{front}\ s = (1 \mathinner{\ldotp\ldotp} \#s - 1) \lhd s \\
\hline
\end{array}
$$

Description

If S is a non-empty sequence of values of type X, then *head* S is the value of type X that is first in the sequence. Empty sequences are not in the domain of *head*.

If S is a non-empty sequence of values of type X, then *last* S is the value of type X that is last in the sequence. Empty sequences are not in the domain of *last*.

If S is a non-empty sequence of values of type X, then *tail* S is the sequence of values of type X obtained from S by discarding the first member. Empty sequences are not in the domain of *tail*.

If S is a non-empty sequence of values of type X, then *front* S is the sequence of values of type X obtained from S by discarding the last member. Empty sequences are not in the domain of *front*.

Name

rev	reverse

Definition

$$
\begin{array}{|l}
\hline\hline [X] \\
rev : \mathrm{seq}\, X \longrightarrow \mathrm{seq}\, X \\
\hline
\forall\, s : \mathrm{seq}\, X \bullet \\
\qquad rev\; s = (\lambda\, n : \mathrm{dom}\, s \bullet s(\# s - n + 1)) \\
\hline
\end{array}
$$

Description

The reverse of a sequence is the sequence obtained by taking its members in the opposite order.

Name

$\upharpoonright$ filtering

Definition

fixity leftfun 40 $\_ \upharpoonright \_$

$$
\begin{array}{|l}
\hline\hline [X] \\
\_ \upharpoonright \_ : \mathrm{seq}\, X \times \mathbb{P}\, X \longrightarrow \mathrm{seq}\, X \\
\hline
\forall V : \mathbb{P}\, X \bullet \\
\quad \langle\rangle \upharpoonright V = \langle\rangle \land \\
\quad (\forall x : X \bullet \\
\qquad (x \in V \Rightarrow \langle x\rangle \upharpoonright V = \langle x\rangle) \land \\
\qquad (x \notin V \Rightarrow \langle x\rangle \upharpoonright V = \langle\rangle)) \land \\
\quad (\forall s, t : \mathrm{seq}\, X \bullet \\
\qquad ((s \mathbin{^\frown} t) \upharpoonright V = (s \upharpoonright V) \mathbin{^\frown} (t \upharpoonright V)) \\
\hline
\end{array}
$$

Description

The filter of a sequence of values of type X by a set of values of type X is the sequence obtained from the original by discarding any members that are not in the set.

Name

$\frown/$ distributed concatenation

Definition

$$
\begin{array}{|l}
[X] \\
\hline
\frown/ : \mathrm{seq}(\mathrm{seq}\, X) \longrightarrow \mathrm{seq}\, X \\
\hline
\frown/\langle\rangle = \langle\rangle \\
\forall\, s : \mathrm{seq}\, X \bullet \frown/\langle s \rangle = s \\
\forall\, q, r : \mathrm{seq}(\mathrm{seq}\, X) \bullet \\
\qquad \frown/(q \frown r) = (\frown/\, q) \frown (\frown/\, r) \\
\hline
\end{array}
$$

Description

The distributed concatenation of a sequence of sequences of values of type X is a sequence of values of type X obtained by concatenating the lesser sequences in the order in which they appear in the greater sequence.

Name

disjoint	disjointness
partition	partitions

Definition

fixity rel disjoint _
fixity rel _ partition _

$$
\begin{array}{l}
[I, X] \\
\hline
\mathsf{disjoint}\ \_ : \mathbb{P}(I \nrightarrow \mathbb{P}\, X) \\
\_\ \mathsf{partition}\ \_ : (I \nrightarrow \mathbb{P}\, X) \leftrightarrow \mathbb{P}\, X \\
\hline
\forall\, S : I \nrightarrow \mathbb{P}\, X;\ T : \mathbb{P}\, X \bullet \\
\quad (\mathsf{disjoint}\ S \Leftrightarrow \\
\qquad (\forall\, i, j : \mathrm{dom}\, S \mid i \neq j \bullet S(i) \cap S(j) = \varnothing)) \land \\
\quad (S\ \mathsf{partition}\ T \Leftrightarrow \\
\qquad \mathsf{disjoint}\ S \land \bigcup\{\, i : \mathrm{dom}\, S \bullet S(i)\,\} = T) \\
\hline
\end{array}
$$

Description

An indexed family of sets is disjoint if no two members having distinct indexes have any members in common.

An indexed family S of sets partition a set T if S is disjoint and the union of all the members of S is T.

D.6 Bags

Name

bag	bags
count	multiplicity
⋿	bag membership

Definition

fixity leftfun 100 \$ bag \$
fixity rel _ ⋿ _

$$\mathrm{bag}\, X == X \mathrel{+\!\!\!\rightarrow} \mathbb{N}_1$$

$$\begin{array}{|l}
[X] \\
\hline
\mathit{count} : \mathrm{bag}\, X \rightarrowtail\!\!\!\!\twoheadrightarrow (X \longrightarrow \mathbb{N}) \\
\_ ⋿ \_ : X \longleftrightarrow \mathrm{bag}\, X \\
\hline
\forall x : X;\ B : \mathrm{bag}\, X \bullet \\
\qquad \mathit{count}\, B = (\lambda\, x : X \bullet 0) \oplus B \wedge \\
\qquad x ⋿ B \Leftrightarrow x \in \mathrm{dom}\, B \\
\hline
\end{array}$$

Description

A bag represents an aggregate in which order is not important, but in which a given value can occur several times. A bag of values of type X is a function whose domain is a subset of X and whose range is a set of strictly positive natural numbers.

The count of a bag of values of type X is a function that extends the bag function by relating every member of X that is not is the domain of the bag to zero.

A value $x : X$ is said to be in B : *bag* X if and only if x is in the domain of B.

Name

$\uplus$ bag union

Definition

$\textbf{fixity}\llbracket\ ..\ \textbf{fixity}\ \ \textbf{leftfun}\ 30\ \ \_ \uplus \_$

$$
\begin{array}{|l}
\hline\hline
[X] \\
\_ \uplus \_ : \mathrm{bag}\,X \times \mathrm{bag}\,X \longrightarrow \mathrm{bag}\,X \\
\hline
\forall B, C : \mathrm{bag}\,X;\ x : X \bullet \\
\qquad \mathit{count}\,(B \uplus C)x = \mathit{count}\,B\,x + \mathit{count}\,C\,x \\
\hline
\end{array}
$$

Description

The bag brackets ⟦ and ⟧ can be used for enumerated bags. The empty bag is the empty function. A singleton enumeration is the bag in which the element has a count of 1. The function that extends a bag is the bag union.

The bag union of two bags is the bag that relates every member of the domain of either bag to the sum of its occurrences in the two bags.

Name

items	bag of elements of a sequence

Definition

$$
\begin{array}{l}
[X] \\
\hline
\mathit{items} : \operatorname{seq} X \longrightarrow \operatorname{bag} X \\
\hline
\forall\, s : \operatorname{seq} X;\ x : X \bullet \\
\qquad \mathit{count}\ (\mathit{items}\ s) x = \#\{\, i : \operatorname{dom}\ s \mid s(i) = x \,\} \\
\hline
\end{array}
$$

Description

The items of a sequence of values of type X is a bag such that the range of the sequence and the domain of the bag are the same, and the each value in the domain of the bag is related to the number of indexes in the sequence at which that value occurred.

□

E Interchange format – Normative Annex

Notes on this section of the Z Standard

Section title: Interchange format
Section editor: Jonathan Hammond (Trevor King)
Contributions by: Trevor King, ... *(others to be added)*
Notes: Conforms to new syntax, lexis and toolkit.
Source file: icformat.tex
Most recent update: 9th August 1995
Formatted: 14th September 1995

E.1 Introduction

The **Interchange Format** defines a portable representation of Z, allowing Z documents to be transmitted between different products or machines. The most suitable means of communication is the use of text files in which the character set is limited for portability reasons. The Interchange Format defines a syntax for such text files.

The basis for the Interchange Format is the ISO Standard Generalized Markup Language (SGML). SGML permits the structure of texts to be represented and encoded in a standard form, convenient for storage, editing, retrieval and processing. The SGML Standard is defined in [13]. A general description of the aims and principles of SGML, together with an annotated version of the standard, is included in *The SGML Handbook* by C. F. Goldfarb [10]. Case studies and applications in SGML are described in the work of the Text Encoding Initiative reported in [23].

The structure of this Appendix is as follows:

- section E.2 describes the scope of the Interchange Format — i.e. the facilities offered by the Format;
- section E.3 contains an informal description of SGML;
- section E.4 defines the Interchange Format;
- section E.5 presents explanatory material and examples of the use of the Interchange Format.

E.2 Scope of the Interchange Format

The Interchange Format allows a distinction to be made between formal text and other text included in a Z document. The Interchange Format does not prescribe the structure of all parts of a Z document; in particular it does not define the internal structure of *informal* text.

As one possible application of the Interchange Format is to transmit a Z document for Z syntax checking, the format is sufficiently liberal to permit syntactically-incorrect Z to be written. The format thus prescribes markup only for the higher levels of the Z syntax hierarchy. No structure is prescribed by the Interchange Format for individual declarations, predicates and expressions.

For a Z document to be syntactically correct when written in the Interchange Format, it must conform at the higher levels to the markup defined in this Appendix, and at the lower levels (e.g. predicate or expression level) to the Z Concrete Syntax, with all mathematical symbols replaced by the alphanumeric representations discussed in Section E.4.3.

The Interchange Format also provides markup for requirements which are additional to the prime requirement for encoding the *structure* of the Z in a document. The following additional requirements are accommodated:

- identification of informal Z fragments, i.e. Z fragments which do not belong to the *formal* part of a Z document;
- indication of particular presentation formats, e.g. whether a vertical or horizontal style should be employed for a schema definition;
- allocation of identifiers to Z paragraphs, e.g. so that associations between Z operation schemas and data-flow diagrams can be made, or so that Z definitions can be indexed;
- logical grouping of Z paragraphs independently of the positions they occupy in the document, e.g. so that the group can be considered as a unit for type-checking purposes, or that 'units of conservative extension' can be identified for subsequent processing by a proof tool;
- labelling of 'stacked' predicates in an axiomatic or 'boxed' definition.

E.3 Introduction to SGML

This section provides an introduction to SGML, sufficient for the understanding of the definition of the Interchange Format in Section E.4. More comprehensive descriptions of SGML are given in [13] and [10].

Examples of text written in SGML are printed with a fixed-width font (the `tt` font in LaTeX) as follows:

```
<tag> text </tag>
```

E.3.1 SGML Element Definitions

Structures are described in the Interchange Format by means of SGML **elements**. Elements are delimited by start-tags and end-tags. A start-tag is of the form `<name>`, where `name` is the generic identifier of the delimited element. The end-tag is of the form `</name>`. For example, a particular Z given set definition may be written in the Interchange Format as:

```
<givendef> NAME, DATE </givendef>
```

The internal structure of a general SGML element is itself defined in SGML by means of a formal SGML **element declaration**. The components of an element declaration are:

1. the name of the element;

2. two characters (separated by a space) which specify the **minimisation rules** for the element;
3. the **content model** of the element.

The minimisation rules indicate whether the start-tags or end-tags may be omitted in instances of the element. The first character in the pair corresponds to the start-tag and the second to the end-tag. The character '-' or 'o' indicates that the corresponding tag respectively must be present or may be omitted.

The content model specifies what any occurrences of the element may legitimately contain. Contents may be specified in terms of other elements and special reserved words. Ultimately all elements consist of 'parsed character data' (represented in element declarations by the reserved word `#PCDATA`), which contains any valid character data but *not* further elements. Further structural information concerning elements which are constituents of the declared element is provided by the use of **occurrence indicators** and **group connectors**.

Occurrence indicators define how many times a constituent element may occur in instances of the defined element and are placed at the end of the constituent element. The following occurrence indicators are used in this Appendix:

- a question mark (?) indicates that the preceding element occurs at most once;
- an asterisk (*) indicates that the preceding element may be absent or occurs one or more times;
- a plus sign (+) indicates that the preceding element occurs one or more times.

Group connectors specify the ordering of constituent elements. The following connectors are used in this Appendix:

- a vertical bar (|) indicates that only one of the components it connects may appear;
- a comma (,) indicates that the components must appear in that order.

For example the element definition for a Z schema declaration is given as:

```
<!ELEMENT schemadef    - -
          ((#PCDATA | sub | mixedname)+, formals?, decpart?, axpart?) >
```

Occurrences of this element thus consist of a sequence of parsed character data, subscript and mixed name elements (representing the name of the schema), followed by an optional element which holds the formal parameters of the definition, followed by elements representing the optional declaration and axiomatic parts of the schema definition. The start-tag and end-tag of the schema definition must both be present.

E.3.2 SGML attribute declarations

In SGML, **attributes** are used to provide information associated with elements. The Interchange Format employs attributes to encode layout information and other information which is not considered to be part of the *structure* of a Z specification. For example, the Interchange Format defines a 'style' attribute for schema definitions which permits an indication of whether the definition should be in vertical or horizontal form. An occurrence of a 'schemadef' element may thus contain an **attribute-value pair** inside the element's start-tag; for example:

```
<schemadef style=vert> S ...  </schemadef>
```

An SGML **attribute declaration** specifies the name(s) of the element(s) to which the attributes are attached, followed by a list of rows, each of which consists of the name of the attribute being declared, its type, and an optional default value. A type may be given as a collection of explicit values, or as one of the following special keywords:

`CDATA`	the attribute value may contain any valid character data and must be delimited by double quotation marks;
`ID`	indicates that a unique identifying value will be supplied for each instance of the element;
`NMTOKEN`	the attribute value is a name token (i.e. any alphanumeric string);
`NUMBER`	the attribute value is a number.

The default value for an attribute may be denoted as one of the set of explicit values defined for an attribute; alternatively it may be one of the following special values:

`#IMPLIED`	a value need not be supplied;
`#REQUIRED`	a value must be supplied.

E.3.3 SGML entities

An **SGML entity** is a named part of a marked-up document. An example of an entity declaration is:

```
<!ENTITY  ZBS     'Z Standard, version 1.2' >
```

References to entities are contructed by prefixing the name of the entity with an ampersand character (&) and delimiting the end of the name with a semicolon, space or end-of-file. Here is an example of an entity reference:

```
We are now in a position to issue the &ZBS;.
```

The entity reference in this document fragment would be expanded by an SGML parser as:

> We are now in a position to issue the Z Standard, version 1.2.

In the Interchange Format, SGML entities are used to represent certain Z symbols or words. The associations between the alphanumeric representation of mathematical symbols or words and their local codes should be defined in SGML entity declarations. However – since local word processor codes may differ – section E.4.3 presents a scheme in which the entity names used in the Interchange Format are listed against the usual presentation format of the corresponding Z symbols or words.

E.4 Definition of the Interchange Format

This section presents the definition of the Interchange Format as a collection of SGML declarations. Explanatory material and examples of the use of the Interchange Format are also given in Section E.5.

An SGML Document Type Definition (DTD) defines the syntax of SGML-conformant documents in a style which is readable by SGML parsers. The Interchange Format does not warrant a full DTD for two reasons:

- the format does not specify the structure of the *informal* text in a Z document;
- the entity declarations are implementation-dependent.

A DTD consists of a header, followed by a body containing the element declarations, attribute declarations and entity declarations. The definition of the Interchange Format presented in this Section may be considered as the partial body of a DTD (*partial* because the entity declarations are not given explicitly); it is also equivalent to a definition in BNF of the structure of the Interchange Format. Newlines in a Z document are not significant in the translation to the Interchange Format except where they serve to separate predicates or declarations.

It is unlikely that this Interchange Format could ever accommodate every function required by its users. However, any collection of SGML declarations (such as those which define this Interchange Format) may be replaced or enhanced by the pre-insertion of additional SGML declarations. Such a 'customisation' of the Interchange Format would be acceptable by SGML parsers.

E.4.1 Element declarations

These declarations define the higher-level structure of the Z paragraphs in a Z document written in the Interchange Format. They correspond closely to the appropriate constructs of the Z Concrete Syntax. Note that no element minimisation options are offered; the start and end tags must both be present for each element instance.

There are two top-level elements:

- the `Z` element represents a sequence of Z paragraphs;
- the `informalZ` element represents a fragment of mathematics which does not belong to the *formal* part of the specification document.

The `Z` element contains a (possibly empty) sequence of individual Z paragraph elements which constitute (part of) a formal specification.

```
<!ELEMENT Z    - -
      (fixity | givendef | axdef | constraint
          | schemadef | gendef | abbrevdef
          | goal | freetypedef)* >
```

The `informalZ` element defines a fragment of mathematics which is not to be considered as a *formal* part of the specification; these fragments may be `Z` elements, `declaration` elements, or a sequence of parsed character data, subscript elements and mixed name elements (corresponding to unstructured fragments of mathematics).

```
<!ELEMENT informalZ    - -
      (Z | declaration | (#PCDATA | sub | mixedname)+) >
```

Elements representing goals, top-level constraints, declarations, abbreviation definition bodies, predicates, fixity template expressions, given set definitions, declarations of formal parameters, subscripts and operator names in fixity template definitions all consist of an unstructured sequence of parsed character data, subscript elements and mixed name elements.

```
<!ELEMENT (goal | constraint | declaration
| body | predicate |exp | givendef
| formals | sub | namearg)    - -
                                    (#PCDATA | sub | mixedname)+ >
```

Elements representing axiomatic definitions, schema definitions and generic definitions contain `decpart` and `axpart` elements; the latter element is optional, as is also the `decpart` element for schema definitions. Schema definitions and generic definitions may contain a `formals` element (representing the declaration of formal parameters). The name introduced by a schema definition is modelled as an unstructured sequence of parsed character data, subscript elements and mixed name elements.

```
<!ELEMENT axdef    - -    (decpart, axpart?) >

<!ELEMENT schemadef    - -
        ((#PCDATA | sub | mixedname)+, formals?, decpart?, axpart?) >

<!ELEMENT gendef    - -    (formals?, decpart, axpart?) >
```

The element representing an abbreviation definition consists of the name introduced by the definition (which is modelled as an unstructured sequence of parsed character data, subscript elements and mixed name elements), an optional `formals` element which represents the declaration of formal parameters, and a `body` element which represents the right-hand side of the abbreviation definition.

```
<!ELEMENT abbrevdef    - -
    ((#PCDATA | sub | mixedname)+, formals?, body) >
```

The element representing a free type definition consists of the name introduced by the definition (which is modelled as an unstructured sequence of parsed character data, subscript elements and mixed name elements) and a non-empty sequence of `branch` elements.

```
<!ELEMENT freetypedef    - -
     ((#PCDATA | sub | mixedname)+, branch+) >
```

The element representing a branch of a free type definition consists of a name (which is modelled as an unstructured sequence of parsed character data, subscript elements and mixed name elements) and an optional `exp` element. If the branch consists of just a name then the name is a member of the free type. If the branch consists of a name and `exp` element then the name is a constructor function whose domain is given by `exp`.

```
<!ELEMENT branch    - -   ((#PCDATA | sub | mixedname)+, exp?) >
```

The element representing the axiomatic part of a 'boxed' definition consists of a sequence of `predicate` elements, representing the predicates which are intended to be separated by the weakly-binding conjunction denoted by significant newlines.

```
<!ELEMENT axpart    - -    (predicate+) >
```

The element representing the declaration part of a 'boxed' definition consists of a sequence of `declaration` elements, each representing a collection of declarations which is separated from other such collections by significant newlines.

```
<!ELEMENT decpart    - -    (declaration+) >
```

The element which represents a fixity statement consists of an unstructured sequence of parsed character data, subscript elements and mixed name elements (modelling the first part of the operator name introduced by the statement) and a (possibly empty) sequence of `namearg` and `exparg` elements which represents the rest of the fixity template. Each `exparg` element consists of three `exp` elements followed by an unstructured sequence of parsed character data, subscript elements and mixed name elements which models part of the operator name.

```
<!ELEMENT fixity    - -
          ((#PCDATA | sub | mixedname)+, (namearg | exparg)*) >

<!ELEMENT exparg    - -
    (exp, exp, exp, (#PCDATA | sub | mixedname)+) >
```

The element which represents a mixed name consists of parsed character data. A mixed name element must be employed in cases where a Z name consists of more than one entity reference or of a mixture of entity references and normal characters (see section E.4.3 for an example).

```
<!ELEMENT mixedname    - -          #PCDATA >
```

E.4.2 Attribute declarations

The attribute declarations permit the association of additional information with occurrences of elements in a Z document written in the Interchange Format.

The attributes `id` and `group` permit identification and logical grouping of Z paragraphs respectively.

```
<!ATTLIST
    (givendef | axdef | constraint | schemadef | gendef
      | goal | abbrevdef | freetypedef)
      id    ID    #IMPLIED
      group    NMTOKEN    #IMPLIED >
```

The attributes `style` and `purpose` define the layout and intended use of a schema definition respectively.

```
<!ATTLIST  schemadef
      style    (vert | horiz)    vert
      purpose    (state | operation | datatype)    #IMPLIED >
```

The attribute `label` permits informal annotation of each member of the 'stack' of predicates which constitutes the axiomatic part of a boxed definition.

```
<!ATTLIST  predicate      label    CDATA    #IMPLIED >
```

The following attributes are used with the `fixity` element:

- the attribute `category` indicates whether the defined operator is a relation, left-associative function or right-associative function;
- the attribute `prec` indicates the binding priority of the defined function (this attribute is not required if the value of `category` is `rel`);
- the attributes `firstarg` and `lastarg` indicate the kind of first and last argument (if any) respectively in the fixity statement.

```
<!ATTLIST  fixity
      category    (leftfun | rightfun | rel)    #REQUIRED
      prec    #NUMBER    #IMPLIED
      (firstarg | lastarg)    (normal | type)    #IMPLIED >
```

The attribute `midarg` of the `namearg` element indicates the kind of argument which appears just before the part of the operator name presented in the `namearg` element.

```
<!ATTLIST  namearg
      midarg    (normal | type)    #REQUIRED >
```

E.4.3 Entity declarations

Entity declarations are used to define alphanumeric denotations for certain Z symbols and words. The entity declarations for the Interchange Format are not presented in the conventional SGML format because of the dependence of the internal representation of mathematical symbols or words on the implementation of each user's Z document processor. The mode of declaration used in this Standard is to present tables which record the association of each entity name with the corresponding mathematical symbol or word. These tables are presented in the Lexis Section of this Standard.

Many of the entity names used in the tables have already been defined as standard in Appendix D of [13].

In order to encode names which consist of more than one entity reference, or of a mixture of entity references and normal text, "mixed name" elements must be employed. For example, the application of the function Δ to the variable *State* (ie Δ *State*) is encoded as `Δ State` (or `ΔState` or `&Delta State`), but the schema name $\Delta State$ must be encoded as any of these alternatives enclosed within `mixedname` element tags. The use of the `mixedname` element indicates that the contents are to be considered as a single name.

Z users may create additional entity declarations to cater for new symbols introduced in their specification documents. For example, assume that the new symbol ⊘ is to be used in a Z document. The author of the document must create an entity declaration of the form

```
<!ENTITY   oslash   SDATA  ¨ oslash¨
  --o enclosing a solidus-->
```

This declaration gives the name `oslash` to the entity which represents the symbol and identifies the local code which generates the presentation format ⊘ of the symbol. As it is unlikely that *other* systems possess this code, a description of the presentation format of the symbol is given as a comment in the entity declaration. Any receivers of the Z document may then establish a suitable local code for this symbol in their respective systems.

Entity definitions for the toolkit

Editor's note: This subsection will be updated when a final version of the toolkit has been developed.

Entity definitions are provided only for the following classes of toolkit member:

- non-alphanumeric members (apart from the addition (+) and multiplication (*) symbols, as it is assumed that these symbols are reasonably portable);
- relations with alphanumeric names (as these are customarily presented in sans-serif font);
- 'type constructors' (ie generic constants with set values) with alphanumeric names (as these are customarily presented in Roman font);
- the functions dom and ran (as these are customarily presented in Roman font).

E.5 Examples

This section presents examples of the use of the Interchange Format. Thses examples are carefully chosen to cover the more obscure aspects of the Format. The areas covered are indicated in the subsection headings.

E.5.1 Declaring infix identifiers

Consider the following axiomatic definition, which declares a relation *isTwice* which is intended to be used in an infix manner:

SYNTAX REL NORMAL *isTwice* NORMAL

$$\begin{array}{|l} \_isTwice\_ : \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline \forall i, j : \mathbb{N} \bullet i \mathrel{isTwice} j \Leftrightarrow i = 2 * j \end{array}$$

This can be encoded in the Interchange Format as

```
<Z>
<fixity  category=rel  firstarg=normal  lastarg=normal>
isTwice </fixity>

<axdef>
<decpart>
<declaration>
  _isTwice_: &Nat; &rel; &Nat;
</declaration> </decpart>
<axpart>
  <predicate>
  &forall; i, j: &Nat; &bull; i isTwice j &iff; i = 2*j
</predicate> </axpart>
</axdef>
</Z>
```

E.5.2 Subscripts

The axiomatic definition

$$\begin{array}{|l} a_1,\ a_3 : \mathbb{N} \\ \hline a_3 \mathrel{isTwice} a_1 \end{array}$$

is encoded in the Interchange Format as:

```
<Z> <axdef>
<decpart>
  <declaration>
    a<sub> 1 </sub>, a<sub> 3 </sub>: &Nat;
</declaration> </decpart>
<axpart>
  <predicate>
    a<sub> 3 </sub> isTwice a<sub> 1 </sub>
</predicate> </axpart>
</axdef> </Z>
```

E.5.3 Schema definitions and predicate labelling

Consider the following definitions:

$[PERSON, HOUSE]$

Street

$$inhabits : PERSON \ffun HOUSE$$
$$houses : \mathbb{P}\, HOUSE$$

$$houses = \mathrm{ran}\, inhabits$$
$$\forall h : houses \bullet \#inhabits^{\sim}(\!|\{h\}|\!) \leq 4$$

/∗ *No house may be occupied by more than* 4 *persons* ∗/

The author of this specification intends to accomplish the following objectives:

- to attach a label to the second predicate in the schema definition;
- to indicate that the schema definition should be displayed in vertical form;
- to indicate (to a specification checker, for example) that the schema *Street* defines the *state* of a system.

These objectives can be attained in the Interchange Format with the following encoding:

```
<Z>
<givendef> PERSON, HOUSE </givendef>

<schemadef style=vert purpose=state> Street
<decpart>
<declaration>
   inhabits: PERSON &fpfun; HOUSE </declaration>
<declaration>
   houses: &pset; HOUSE </declaration> </decpart>
```

```
<axpart>
  <predicate>
    houses = &ran; inhabits  </predicate>
  <predicate
     label='No house may be occupied by more than 4 persons'>
    &forall; h: houses &bull;
       &num; inhabits &tilde; &limg; &lcub; h &rcub; &rimg;  &le; 4
</predicate> </axpart>
</schemadef>
</Z>
```

E.5.4 Symbols in top-level definitions

Note that in the Interchange Format there are no *entity* representations of the symbols immediately associated with top-level definitions such as free type definitions and abbreviation definitions. These symbols are subsumed by the element tags for those definitions. For example, consider the following abbreviation definition:

$$n == 5 + x$$

This definition is encoded in the Interchange Format as:

```
<Z> <abbrevdef> n
<body> 5 + x </body> </abbrevdef> </Z>
```

i.e. there is no need in the Interchange Format for an SGML entity which represents the == symbol (although this symbol would of course be employed in the *presentation format* for instances of the `abbrevdef` element).

□

F The logical theory of Z – Normative Annex

Notes on this section of the Z Standard

Section title: The Logical Theory of Z
Section editor: Andrew Martin
Original text by: Stephen Brien
Contributions by: Peter Lupton, Jim Woodcock . . . *(to be added)*
Source file: logical.tex
Most recent update: 22nd August 1995
Formatted: 14th September 1995

F.1 Preamble

Editor's note: *To be supplied by Peter Lupton.* This chapter defines what the theorems of Z are, discusses the relationship between logic, deductive system, semantics; and soundness. It also describes how other deductive systems (and logics?) are to be derived; what it means for them to be conformant/sound and complete.

The remaining sections of this annex define the meta-language used, and then present rules for free variable, alphabet, and substitution calculation, type inference rules, and value (logical) inference rules.

This inference system is not intended to be (necessarily) *the* inference system for Z, rather one against which other systems (such as an equational reasoning framework) can be proven sound.

F.2 Meta-language

The deductive system will be expressed using the abstract syntax. Some simplifications will be used. In particular, the paragraph keywords **given**, **gendef**, **abbr** etc. will be omitted.

F.2.1 Meta-Variables

The meta-variables used in what follows are arbitrary members of the following (abstract) syntactic categories:

$$
\begin{aligned}
\boldsymbol{\Pi} &:: \mathsf{PAR}\\
\boldsymbol{\Gamma} &:: \mathsf{PAR} \ddagger \ldots \ddagger \mathsf{PAR}\\
\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R} &:: \mathsf{PRED}\\
\boldsymbol{x}, \boldsymbol{y} &:: \mathsf{NAME}\\
\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v} &:: \mathsf{EXP}\\
\boldsymbol{S}, \boldsymbol{T} &:: \mathsf{SCHEMA}\\
{}^{q} &:: \mathsf{DECOR}\\
\boldsymbol{i} &:: \mathsf{NUMBERL}\\
\boldsymbol{z} &:: \mathsf{STRINGL}
\end{aligned}
$$

F.2.2 Sequent

The abstract syntax has a paragraph form CONJECTURE, which is a simple sequent:

$$\vdash \boldsymbol{P}$$

The deductive system will be presented using a richer language of sequents, in which a sequent has a list of Z paragraphs on its left-hand side.

$$\boldsymbol{\Pi}_1 \ddagger \cdots \boldsymbol{\Pi}_n \vdash \boldsymbol{P}$$

These paragraphs provide a context for the predicate on the right-hand side, (for the sequent to be well-formed, the free variables of $\boldsymbol{P}$ must be declared in $\boldsymbol{\Pi}_1 \ddagger \cdots \ddagger \boldsymbol{\Pi}_n$) and may be considered to be (part of) the specification in which the conjecture is evaluated.

The preamble above describes what it means for a sequent to be a theorem of Z; the deductive system defines which sequents may be deduced from which other sequents.

This chapter also presents *type judgements* of the form

$$\boldsymbol{\Gamma} \vdash \boldsymbol{e} \mathrel{⦂} \boldsymbol{\tau}$$

This sequent means that, in the specification $\boldsymbol{\Gamma}$, the expression $\boldsymbol{e}$ is well-typed with type $\boldsymbol{\tau}$. The following proof rules assume that $\boldsymbol{\tau}$ is arbitrary, it should be established by pattern matching. For predicates (which do not have a type) we write $\vdash \boldsymbol{P} \ \surd$ to mean that the predicate $\boldsymbol{P}$ is well typed. For paragraphs we write $\vdash \boldsymbol{P} ::$ to indicate that they are well-typed. Though the turnstile is the same as for the deduction rules, it is used to represent different kinds of relations. These assertions can be distinguished by the syntax of the consequent (the antecedent in both cases being a specification).

A third form of judgement is mentioned briefly below. Certain provisos must be evaluated *within a context* (in particular, schema alphabets can be calculated only when suitable schema definitions are present).

$$\boldsymbol{\Gamma} \vdash \mathit{proviso}$$

The paragraphs $\boldsymbol{\Gamma}$ provide the context in which to evaluate the proviso. Some meta-rules in Section F.2.5 suggest how the different presentations of inference rules can be related. Again, though this form of sequent appears very like that for logical (value) judgements, it is distinguished by the use of meta-functions ϕ and α in the antecedent.

F.2.3 Rules

The deductive systems consist of a number of rules for manipulating sequents. Inference rules will be written

$$\mathit{Rule} \mathrel{\widehat{=}} \frac{\mathit{Premises}}{\mathit{Conclusion}}\ \mathit{Name}\ (\mathit{Proviso})$$

The premises are a (possibly empty) list of sequents:

$$\mathit{Premises} \mathrel{\widehat{=}} \mathit{Sequent} \ldots \mathit{Sequent}$$

The conclusion is always a single sequent:

$$\mathit{Conclusion} \mathrel{\widehat{=}} \mathit{Sequent}$$

The Proviso is a decidable condition on the free variables of the expressions and predicates in the rule. The annotation ↑↓ indicates that the rule can be applied in both directions—that is, the rule

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{Q}}{\boldsymbol{\Gamma}' \vdash \boldsymbol{P}}\ \uparrow\downarrow$$

denotes both of the following inference rules

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{Q}}{\boldsymbol{\Gamma}' \vdash \boldsymbol{P}} \qquad \text{and} \qquad \frac{\boldsymbol{\Gamma}' \vdash \boldsymbol{P}}{\boldsymbol{\Gamma} \vdash \boldsymbol{Q}}$$

A rule is *sound* if whenever it is applied to valid premisses, a valid conclusion results. This is defined in the semantics by saying that the set of environments supporting the premisses is a subset of those supporting the conclusion. The rule

$$\frac{S_1 \quad \ldots \quad S_m}{\mathit{Seq}}\ N(P)$$

is sound if and only if

$$P \quad \Rightarrow \quad \{\!\![\ \boldsymbol{S}_1\ ]\!\!\}^{\mathcal{M}} \cap \ldots \cap \{\!\![\ \boldsymbol{S}_m\ ]\!\!\}^{\mathcal{M}} \subseteq \{\!\![\ \boldsymbol{Seq}\ ]\!\!\}^{\mathcal{M}}$$

The following meta-theorem holds for rules in the deductive system:

Theorem F.1 (Sequent-lifting)

The rule $\dfrac{}{\boldsymbol{\Gamma} \vdash \boldsymbol{P}}$ *is sound if and only if the sequent* $\boldsymbol{\Gamma} \vdash \boldsymbol{P}$ *is a theorem.*

This meta-theorem states that a theorem can be deduced from no premisses.

The semantic equivalences for substitution are given in tables in later sections. These tables state the semantic equality of various expressions. A theorem which permits the use of semantic-equivalences in proofs is the following.

Theorem F.2 (Semantic-equivalence-lifting) *Given the semantic-equivalence for any predicates or expressions*

$$A \equiv B$$

the following inference rule is sound:

$$\frac{\Delta(A)}{\Delta(B)}$$

F.2.4 wf

The proviso $\mathbf{wf}(\boldsymbol{\Pi})$ is an abbreviation stating that $(\boldsymbol{\Pi})$ is well-formed in that

$$\phi(\boldsymbol{\Pi}) \cap \alpha(\boldsymbol{\Pi}) = \varnothing$$

F.2.5 Provisos as judgements

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{P}}{\boldsymbol{\Gamma}' \vdash \boldsymbol{P}'}\;(\alpha\boldsymbol{S} = \boldsymbol{s}) \equiv \frac{\boldsymbol{\Gamma} \vdash \boldsymbol{P} \qquad \boldsymbol{\Gamma}' \vdash \alpha\boldsymbol{S} = \boldsymbol{s}}{\boldsymbol{\Gamma}' \vdash \boldsymbol{P}'}$$

$$\frac{\boldsymbol{\Gamma} \vdash \phi\boldsymbol{S} = \boldsymbol{x} \qquad \boldsymbol{\Gamma} \vdash \alpha\boldsymbol{S} = \boldsymbol{z} \qquad \boldsymbol{\Gamma}\ddagger\boldsymbol{S} \vdash \phi\boldsymbol{P} = \boldsymbol{y}}{\boldsymbol{\Gamma} \vdash \phi(\forall \boldsymbol{S} \bullet \boldsymbol{P}) = \boldsymbol{x} \cup (\boldsymbol{y} \setminus \boldsymbol{z})}$$

F.2.6 Proofs

Proofs in the deductive system proceed in the way that is usual for sequent calculi: proofs are developed *backwards*, starting from the sequent which is to be proved. A rule is applied, resulting in fresh sequents which must be proved. This process continues until there are no more sequents requiring proof, in which case the original sequent is now proved.

A completed proof may thus be represented as a tree, with the proved sequent as the root node, and every leaf node containing an empty list of sequents. However, if some of these lists in the leaves are non-empty, then the derivation tree is still useful, although it does not represent a proof, it represents a partial proof.

Theorem F.3 (Tree-squashing) *Suppose that we have the derivation tree:*

$$\frac{S_1 \quad \ldots \quad \dfrac{S_{i1} \quad \ldots \quad S_{im}}{S_i}\;[R_i](P_i) \quad \ldots \quad S_n}{\mathit{Seq}}\;[R](P)$$

where each of the rules R and R_i are sound rules, then the derived rule

$$\frac{S_1 \quad \ldots \quad S_{i1} \quad \ldots \quad S_{im} \quad \ldots \quad S_n}{Seq} \; [R'](P, P_i)$$

is also sound.

F.3 Free variables and alphabets

F.3.1 Paragraphs

$$
\begin{aligned}
\phi[\boldsymbol{x}] &= \varnothing \\
\phi\boldsymbol{P} &= \Phi\boldsymbol{P} \\
\phi\boldsymbol{S} &= \text{See below} \\
\phi(\boldsymbol{x} : \boldsymbol{s}) &= \phi\boldsymbol{s} \\
\phi(\boldsymbol{x} := \boldsymbol{e}) &= \phi\boldsymbol{e} \\
\phi([\boldsymbol{x}]\boldsymbol{S}) &= \phi\boldsymbol{S} \setminus \{\boldsymbol{x}\} \\
\phi(\boldsymbol{x}[\boldsymbol{y}] := \boldsymbol{e}) &= \phi\boldsymbol{e} \\
\phi(\boldsymbol{\Pi}_1 \ddagger \boldsymbol{\Pi}_2) &= \phi\boldsymbol{\Pi}_1 \cup (\phi\boldsymbol{\Pi}_2 \setminus \alpha\boldsymbol{\Pi}_1)
\end{aligned}
$$

$$
\begin{aligned}
\alpha[\boldsymbol{x}] &= \{\boldsymbol{x}\} \\
\alpha\boldsymbol{P} &= \varnothing \\
\alpha\boldsymbol{S} &= \text{See below} \\
\alpha(\boldsymbol{x} : \boldsymbol{s}) &= \{\boldsymbol{x}\} \\
\alpha(\boldsymbol{x} := \boldsymbol{e}) &= \{\boldsymbol{x}\} \\
\alpha([\boldsymbol{x}]\boldsymbol{S}) &= \alpha\boldsymbol{S} \\
\alpha(\boldsymbol{x}[\boldsymbol{y}] := \boldsymbol{e}) &= \{\boldsymbol{x}\} \\
\alpha(\boldsymbol{\Pi}_1 \ddagger \boldsymbol{\Pi}_2) &= \alpha\boldsymbol{\Pi}_1 \cup \alpha\boldsymbol{\Pi}_2
\end{aligned}
$$

F.3.2 Predicates

$$
\begin{aligned}
\Phi(\boldsymbol{e} \in \boldsymbol{s}) &= \phi\boldsymbol{e} \cup \phi\boldsymbol{s} \\
\Phi(\boldsymbol{e} = \boldsymbol{v}) &= \phi\boldsymbol{e} \cup \phi\boldsymbol{v} \\
\Phi\mathbf{true} &= \varnothing \\
\Phi\mathbf{false} &= \varnothing \\
\Phi(\neg\boldsymbol{P}) &= \Phi\boldsymbol{P} \\
\Phi(\boldsymbol{P} \wedge \boldsymbol{Q}) &= \Phi\boldsymbol{P} \cup \Phi\boldsymbol{Q} \\
\Phi(\boldsymbol{P} \vee \boldsymbol{Q}) &= \Phi\boldsymbol{P} \cup \Phi\boldsymbol{Q} \\
\Phi(\boldsymbol{P} \Rightarrow \boldsymbol{Q}) &= \Phi\boldsymbol{P} \cup \Phi\boldsymbol{Q} \\
\Phi(\boldsymbol{P} \Leftrightarrow \boldsymbol{Q}) &= \Phi\boldsymbol{P} \cup \Phi\boldsymbol{Q} \\
\Phi\,\forall\,\boldsymbol{S} \bullet \boldsymbol{P} &= \phi\boldsymbol{S} \cup (\Phi\boldsymbol{P} \setminus \alpha\boldsymbol{S}) \\
\Phi\,\exists\,\boldsymbol{S} \bullet \boldsymbol{P} &= \phi\boldsymbol{S} \cup (\Phi\boldsymbol{P} \setminus \alpha\boldsymbol{S}) \\
\Phi\,\exists_1\,\boldsymbol{S} \bullet \boldsymbol{P} &= \phi\boldsymbol{S} \cup (\Phi\boldsymbol{P} \setminus \alpha\boldsymbol{S}) \\
\Phi(\boldsymbol{S}) &= \alpha\boldsymbol{S} \cup \phi\boldsymbol{S} \\
\Phi(\boldsymbol{b} \odot P) &= \phi\boldsymbol{b} \cup (\Phi\boldsymbol{P} \setminus \alpha\boldsymbol{b})
\end{aligned}
$$

F.3.3 Schemas

$$
\begin{aligned}
\phi[\boldsymbol{x}_1 : \boldsymbol{s}_1;\ \cdots;\boldsymbol{x}_n : \boldsymbol{s}_n] &= \phi(\boldsymbol{s}_1) \cup \cdots \cup \phi(\boldsymbol{s}_n)\\
\phi[\boldsymbol{S} \mid \boldsymbol{P}] &= \phi\boldsymbol{S} \cup (\Phi\boldsymbol{P} \setminus \alpha\boldsymbol{S})\\
\phi(\neg\boldsymbol{S}) &= \phi\boldsymbol{S}\\
\phi(\boldsymbol{S} \land \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S} \lor \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S} \Rightarrow \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S} \Leftrightarrow \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S} \upharpoonright \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S}\setminus[\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n]) &= \phi\boldsymbol{S}\\
\phi(\forall\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\exists\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\exists_1\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S}[\boldsymbol{x}_1/\boldsymbol{y}_1,\ldots,\boldsymbol{x}_n/\boldsymbol{y}_n]) &= \phi\boldsymbol{S}\\
\phi(\boldsymbol{S} \fatsemi \boldsymbol{T}) &= \phi\boldsymbol{S}\cup\phi\boldsymbol{T}\\
\phi(\boldsymbol{S}^{q}) &= \phi\boldsymbol{S}
\end{aligned}
$$

$$
\begin{aligned}
\alpha[\boldsymbol{x}_1 : \boldsymbol{s}_1,\ldots,\boldsymbol{x}_n : \boldsymbol{s}_n] &= \{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}\\
\alpha[\boldsymbol{S} \mid \boldsymbol{P}] &= \alpha\boldsymbol{S}\\
\alpha(\neg\boldsymbol{S}) &= \alpha\boldsymbol{S}\\
\alpha(\boldsymbol{S} \land \boldsymbol{T}) &= \alpha\boldsymbol{S}\cup\alpha\boldsymbol{T}\\
\alpha(\boldsymbol{S} \lor \boldsymbol{T}) &= \alpha\boldsymbol{S}\cup\alpha\boldsymbol{T}\\
\alpha(\boldsymbol{S} \Rightarrow \boldsymbol{T}) &= \alpha\boldsymbol{S}\cup\alpha\boldsymbol{T}\\
\alpha(\boldsymbol{S} \Leftrightarrow \boldsymbol{T}) &= \alpha\boldsymbol{S}\cup\alpha\boldsymbol{T}\\
\alpha(\boldsymbol{S} \upharpoonright \boldsymbol{T}) &= \alpha\boldsymbol{S}\cap\alpha\boldsymbol{T}\\
\alpha(\boldsymbol{S}\setminus[\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n]) &= \alpha\boldsymbol{S}\setminus\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}\\
\alpha(\forall\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \alpha\boldsymbol{T}\setminus\alpha\boldsymbol{S}\\
\alpha(\exists\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \alpha\boldsymbol{T}\setminus\alpha\boldsymbol{S}\\
\alpha(\exists_1\, \boldsymbol{S} \bullet \boldsymbol{T}) &= \alpha\boldsymbol{T}\setminus\alpha\boldsymbol{S}\\
\alpha(\boldsymbol{S}[\boldsymbol{x}_1/\boldsymbol{y}_1,\ldots,\boldsymbol{x}_n/\boldsymbol{y}_n]) &= (\alpha\boldsymbol{S})[\boldsymbol{x}_1/\boldsymbol{y}_1,\ldots,\boldsymbol{x}_n/\boldsymbol{y}_n]\\
\alpha(\boldsymbol{S} \fatsemi \boldsymbol{T}) &=\\
\alpha(\boldsymbol{S}^{q}) &= (\alpha\boldsymbol{S})^{q}\\
\alpha\langle\!\vert\ \boldsymbol{x}_1 := \boldsymbol{e}_1,\ldots,\boldsymbol{x}_n := \boldsymbol{e}_n \vert\!\rangle &= \{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}
\end{aligned}
$$

F.3.4 Expressions

$$
\begin{aligned}
\phi(\boldsymbol{x}) &= \{\boldsymbol{x}\} \\
\phi(\boldsymbol{x}[\boldsymbol{e}]) &= \{\boldsymbol{x}\} \cup \phi\boldsymbol{e} \\
\phi(\boldsymbol{i}) &= \varnothing \\
\phi(\boldsymbol{z}) &= \varnothing \\
\phi\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\} &= \phi(\boldsymbol{e}_1) \cup \cdots \cup \phi(\boldsymbol{e}_n) \\
\phi\{\boldsymbol{S} \bullet \boldsymbol{e}\} &= \phi\boldsymbol{S} \cup (\phi\boldsymbol{e} \setminus \alpha\boldsymbol{S}) \\
\phi(\mathbf{Pow}\, \boldsymbol{s}) &= \phi(\boldsymbol{s}) \\
\phi(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n) &= \phi(\boldsymbol{e}_1) \cup \cdots \cup \phi(\boldsymbol{e}_n) \\
\phi(\boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_n) &= \phi(\boldsymbol{s}_1) \cup \cdots \cup \phi(\boldsymbol{s}_n) \\
\phi(\boldsymbol{e}.\boldsymbol{i}) &= \phi(\boldsymbol{e}) \\
\phi\langle\!|\ \boldsymbol{x}_1 := \boldsymbol{e}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{e}_n\ |\!\rangle &= \phi(\boldsymbol{e}_1) \cup \cdots \cup \phi(\boldsymbol{e}_n) \\
\phi(\theta\boldsymbol{S}) &= \Phi\boldsymbol{S} \\
\phi(\boldsymbol{b}.\boldsymbol{x}) &= \phi(\boldsymbol{b}) \\
\phi(\boldsymbol{f}\, \boldsymbol{e}) &= \phi\boldsymbol{f} \cup \phi\boldsymbol{e} \\
\phi(\mu\, \boldsymbol{S} \bullet \boldsymbol{e}) &= \phi\boldsymbol{S} \cup \Phi(\boldsymbol{e}_1 \setminus \alpha\boldsymbol{S}) \\
\phi(\mathbf{if}\ \boldsymbol{P}\ \mathbf{then}\ \boldsymbol{e}_1\ \mathbf{else}\ \boldsymbol{e}_2\ \mathbf{fi}) &= \Phi\boldsymbol{P} \cup \phi\boldsymbol{e}_1 \cup \phi\boldsymbol{e}_2 \\
\phi(\boldsymbol{b} \odot \boldsymbol{e}) &= \phi\boldsymbol{b} \cup (\phi\boldsymbol{e} \setminus \alpha\boldsymbol{b})
\end{aligned}
$$

F.4 Substitution

F.4.1 Predicates

$$
\begin{aligned}
\boldsymbol{b} \odot (\boldsymbol{e} = \boldsymbol{u}) &\equiv \boldsymbol{b}{\odot}\boldsymbol{e} = \boldsymbol{b}{\odot}\boldsymbol{u} \\
\boldsymbol{b} \odot (\boldsymbol{e} \in \boldsymbol{s}) &\equiv \boldsymbol{b}{\odot}\boldsymbol{e} \in \boldsymbol{b}{\odot}\boldsymbol{s} \\
\boldsymbol{b} \odot \mathbf{true} &\equiv \mathbf{true} \\
\boldsymbol{b} \odot \mathbf{false} &\equiv \mathbf{false} \\
\boldsymbol{b} \odot \neg \boldsymbol{P} &\equiv \neg\, \boldsymbol{b} \odot \boldsymbol{P} \\
\boldsymbol{b} \odot (\boldsymbol{P} \wedge \boldsymbol{Q}) &\equiv \boldsymbol{b} \odot \boldsymbol{P} \wedge \boldsymbol{b} \odot \boldsymbol{Q} \\
\boldsymbol{b} \odot (\boldsymbol{P} \vee \boldsymbol{Q}) &\equiv \boldsymbol{b} \odot \boldsymbol{P} \vee \boldsymbol{b} \odot \boldsymbol{Q} \\
\boldsymbol{b} \odot (\boldsymbol{P} \Rightarrow \boldsymbol{Q}) &\equiv \boldsymbol{b} \odot \boldsymbol{P} \Rightarrow \boldsymbol{b} \odot \boldsymbol{Q} \\
\boldsymbol{b} \odot (\boldsymbol{P} \Leftrightarrow \boldsymbol{Q}) &\equiv \boldsymbol{b} \odot \boldsymbol{P} \Leftrightarrow \boldsymbol{b} \odot \boldsymbol{Q} \\
\text{When } \alpha\boldsymbol{b} \cap \Phi\boldsymbol{P} \subseteq \alpha\boldsymbol{S}\text{:} & \\
\boldsymbol{b} \odot \forall\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \forall\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{P} \\
\boldsymbol{b} \odot \exists\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \exists\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{P} \\
\boldsymbol{b} \odot \exists_1\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \exists_1\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{P} \\
\text{When } \alpha\boldsymbol{b} \cap \alpha\boldsymbol{S} \cap \Phi\boldsymbol{P} = \varnothing & \text{ and } \alpha\boldsymbol{S} \cap \phi\boldsymbol{b} = \varnothing\text{:} \\
\boldsymbol{b} \odot \forall\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \forall\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{P} \\
\boldsymbol{b} \odot \exists\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \exists\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{P} \\
\boldsymbol{b} \odot \exists_1\, \boldsymbol{S} \bullet \boldsymbol{P} &\equiv \exists_1\, \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{P} \\
\text{When } \alpha\boldsymbol{b} \cap \alpha\boldsymbol{S} = \varnothing\text{:} & \\
\boldsymbol{b} \odot \boldsymbol{S} &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S}] \\
\text{When } \mathbf{wf}\,\boldsymbol{b}\text{:} & \\
\boldsymbol{b} \odot \boldsymbol{S} &\equiv \boldsymbol{b} \odot [\boldsymbol{b}{\odot}\boldsymbol{S}]
\end{aligned}
$$

$$
\begin{aligned}
\llparenthesis\, \boldsymbol{y} := \boldsymbol{v} \,\rrparenthesis \odot (\llparenthesis\, \boldsymbol{y} := \boldsymbol{u} \,\rrparenthesis \odot \boldsymbol{P}) &\equiv \llparenthesis\, \boldsymbol{y} := \llparenthesis\, \boldsymbol{y} := \boldsymbol{v} \,\rrparenthesis {\odot}\boldsymbol{u} \,\rrparenthesis \odot \boldsymbol{P} \\
\llparenthesis\, \boldsymbol{x} := \boldsymbol{v} \,\rrparenthesis \odot (\llparenthesis\, \boldsymbol{y} := \boldsymbol{u} \,\rrparenthesis \odot \boldsymbol{P}) &\equiv \llparenthesis\, \boldsymbol{y} := \llparenthesis\, \boldsymbol{x} := \boldsymbol{v} \,\rrparenthesis {\odot}\boldsymbol{u} \,\rrparenthesis \odot (\llparenthesis\, \boldsymbol{x} := \boldsymbol{v} \,\rrparenthesis \odot \boldsymbol{P}) \\
& \text{ where } \boldsymbol{y} \notin \phi\boldsymbol{v}
\end{aligned}
$$

F.4.2 Schema predicates

$$
\begin{array}{rcl}
\boldsymbol{b} \odot [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n : \boldsymbol{s}] & \equiv & \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{x}_1 \in \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{s} \wedge \ldots \wedge \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{x}_n \in \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{s} \\
\boldsymbol{b} \odot [\boldsymbol{S} \mid \boldsymbol{P}] & \equiv & \boldsymbol{b} \odot \boldsymbol{S} \wedge \boldsymbol{b} \odot \boldsymbol{P} \\
\boldsymbol{b} \odot [\neg \boldsymbol{S}] & \equiv & \neg \boldsymbol{b} \odot \boldsymbol{S} \\
\boldsymbol{b} \odot [\boldsymbol{S} \wedge \boldsymbol{T}] & \equiv & \boldsymbol{b} \odot \boldsymbol{S} \wedge \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\boldsymbol{S} \vee \boldsymbol{T}] & \equiv & \boldsymbol{b} \odot \boldsymbol{S} \vee \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\boldsymbol{S} \Rightarrow \boldsymbol{T}] & \equiv & \boldsymbol{b} \odot \boldsymbol{S} \Rightarrow \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\boldsymbol{S} \Leftrightarrow \boldsymbol{T}] & \equiv & \boldsymbol{b} \odot \boldsymbol{S} \Leftrightarrow \boldsymbol{b} \odot \boldsymbol{T} \\
\text{When } \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} = \alpha \boldsymbol{S} \setminus \alpha \boldsymbol{T} & & \\
\boldsymbol{b} \odot [\boldsymbol{S} \upharpoonright \boldsymbol{T}] & \equiv & \exists\, \boldsymbol{b} {\scriptstyle\odot} [\boldsymbol{x}_1 : \boldsymbol{e}_1;\ \cdots;\ \boldsymbol{x}_n : \boldsymbol{e}_n] \bullet b \odot (\boldsymbol{S} \wedge \boldsymbol{T}) \\
\boldsymbol{b} \odot \boldsymbol{S} \setminus [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] & \equiv & \exists\, \boldsymbol{b} {\scriptstyle\odot} [\boldsymbol{x}_1 : \boldsymbol{e}_1;\ \cdots;\ \boldsymbol{x}_n : \boldsymbol{e}_n] \bullet b \odot \boldsymbol{S} \\
\text{When } \alpha \boldsymbol{S} \cap (\phi \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{T} \cup \alpha \boldsymbol{b}) = \varnothing\text{:} & & \\
\boldsymbol{b} \odot [\forall\, \boldsymbol{S} \bullet \boldsymbol{T}] & \equiv & \forall\, \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\exists\, \boldsymbol{S} \bullet \boldsymbol{T}] & \equiv & \exists\, \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\exists_1\, \boldsymbol{S} \bullet \boldsymbol{T}] & \equiv & \exists_1\, \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{T} \\
\boldsymbol{b} \odot [\boldsymbol{S}[\boldsymbol{x}_1 / \boldsymbol{y}_1, \ldots, \boldsymbol{x}_n / \boldsymbol{y}_n]] & \equiv & \\
\boldsymbol{b} \odot [\boldsymbol{S} \fatsemi \boldsymbol{T}] & \equiv & \\
\boldsymbol{b} \odot [\boldsymbol{S}^q] & \equiv &
\end{array}
$$

F.4.3 Expressions

$$\begin{aligned}
b \odot x &\equiv x \quad \text{when } x \notin \alpha b \\
\langle\!| \, b, x := e \,|\!\rangle \odot x &\equiv e \\
\langle\!| \, b, x := e \,|\!\rangle \odot y &\equiv \langle\!| \, b \,|\!\rangle \odot e \\
b \odot x[y] &\equiv \\
b \odot i &\equiv i \\
b \odot z &\equiv z \\
b \odot \{e_1, \ldots, e_n\} &\equiv \{b \odot e_1, \ldots, b \odot e_n\}
\end{aligned}$$

When $\alpha b \cap \phi e \subseteq \alpha S$:

$$\begin{aligned}
b \odot \{S \bullet e\} &\equiv \{\, b \odot S \bullet e \,\} \\
b \odot (\mu\, S \bullet e) &\equiv \mu\, b \odot S \bullet e
\end{aligned}$$

When $\alpha b \cap \alpha S \cap \phi e = \varnothing$ and $\alpha S \cap \phi b = \varnothing$:

$$\begin{aligned}
b \odot \{S \bullet e\} &\equiv \{b \odot S \bullet b \odot e\} \\
b \odot (\mu\, S \bullet e) &\equiv \mu\, b \odot S \bullet b \odot e
\end{aligned}$$

$$\begin{aligned}
b \odot \mathrm{Pow}\, s &\equiv \mathrm{Pow}\, b \odot s \\
b \odot ((e_1, \ldots, e_n)) &\equiv (b \odot e_1, \ldots, b \odot e_n) \\
b \odot (s_1 \times \cdots \times s_n) &\equiv b \odot s_1 \times \cdots \times b \odot s_n \\
b \odot (e.i) &\equiv (b \odot e).i \\
b \odot \langle\!| \, x_1 := e_1, \ldots, x_n := e_n \,|\!\rangle &\equiv \langle\!| \, x_1 := b \odot e_1, \ldots, x_n := b \odot e_n \,|\!\rangle
\end{aligned}$$

When $\alpha b \cap \alpha S = \varnothing$

$$b \odot \theta S \equiv \theta b \odot S$$

When $\alpha b = \alpha S$

$$\begin{aligned}
b \odot \theta S &\equiv b \\
b_1 \odot b.x &\equiv (b_1 \odot b).x \\
b \odot (f\, e) &\equiv (b \odot f)(b \odot e) \\
b \odot (\mathbf{if}\ P\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2\ \mathbf{fi}) &\equiv \mathbf{if}\, b \odot P\, \mathbf{then}\, b \odot e_1\, \mathbf{else}\, b \odot e_2 \\
b_1 \odot (b \odot e) &\equiv \\
\langle\!| \, x := v \,|\!\rangle \odot (\langle\!| \, x := u \,|\!\rangle \odot e) &\equiv \langle\!| \, x := \langle\!| \, x := v \,|\!\rangle \odot u \,|\!\rangle \odot e \\
\langle\!| \, y := v \,|\!\rangle \odot (\langle\!| \, x := u \,|\!\rangle \odot e) &\equiv \langle\!| \, x := \langle\!| \, y := v \,|\!\rangle \odot u \,|\!\rangle \odot \\
&\qquad\qquad (\langle\!| \, y := v \,|\!\rangle \odot e) \\
&\quad \text{when } x \notin \phi v.
\end{aligned}$$

F.4.4 Schema expressions

$$
\begin{aligned}
\boldsymbol{b}{\odot}[\boldsymbol{x}_1, \ldots \boldsymbol{x}_n : \boldsymbol{s}] &\equiv [\boldsymbol{x}_1, \ldots \boldsymbol{x}_n : \boldsymbol{b}{\odot}\boldsymbol{s}] \\
\text{When } \alpha\boldsymbol{b} \cap \alpha\boldsymbol{S} = \varnothing & \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \mid \boldsymbol{P}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \mid \boldsymbol{b} \odot \boldsymbol{P}] \\
\text{When } \alpha\boldsymbol{b} \cap \Phi\boldsymbol{P} \subseteq \alpha\boldsymbol{S} & \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \mid \boldsymbol{P}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \mid \boldsymbol{P}] \\
\boldsymbol{b}{\odot}[\neg \boldsymbol{S}] &\equiv [\neg \boldsymbol{b}{\odot}\boldsymbol{S}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \land \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \land \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \lor \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \lor \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \Rightarrow \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \Rightarrow \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \Leftrightarrow \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \Leftrightarrow \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \upharpoonright \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \upharpoonright \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}\boldsymbol{S} \backslash [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] &\equiv (\boldsymbol{b}{\odot}\boldsymbol{S}) \backslash [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \\
\boldsymbol{b}{\odot}[\forall \boldsymbol{S} \bullet \boldsymbol{T}] &\equiv [\forall \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\exists \boldsymbol{S} \bullet \boldsymbol{T}] &\equiv [\exists \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\exists_1 \boldsymbol{S} \bullet \boldsymbol{T}] &\equiv [\exists_1 \boldsymbol{b}{\odot}\boldsymbol{S} \bullet \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S}[\boldsymbol{x}_1/\boldsymbol{y}_1, \ldots, \boldsymbol{x}_n/\boldsymbol{y}_n]] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S}[\boldsymbol{x}_1/\boldsymbol{y}_1, \ldots, \boldsymbol{x}_n/\boldsymbol{y}_n]] \\
\boldsymbol{b}{\odot}[\boldsymbol{S} \fatsemi \boldsymbol{T}] &\equiv [\boldsymbol{b}{\odot}\boldsymbol{S} \fatsemi \boldsymbol{b}{\odot}\boldsymbol{T}] \\
\boldsymbol{b}{\odot}[\boldsymbol{S}^{q}] &\equiv [(\boldsymbol{b}{\odot}\boldsymbol{S})^{q}]
\end{aligned}
$$

F.5 Type inference

F.5.1 Structural rules

$$\frac{\Gamma \vdash \Theta}{\Pi \ddagger \Gamma \vdash \Theta} \quad \text{T1}$$

$$\frac{\Gamma \vdash \Theta}{\Gamma \ddagger \Pi \vdash \Theta} \quad \text{T2}$$

$$\frac{\Gamma \vdash x \fatsemi \tau}{\Gamma \ddagger [y] \vdash x \fatsemi \tau} \quad \text{T3}$$

$$\frac{\Gamma \vdash x \fatsemi \tau}{\Gamma \ddagger y := e \vdash x \fatsemi \tau} \quad \text{T4}$$

$$\frac{\Gamma \vdash x \fatsemi \tau}{\Gamma \ddagger y : s \vdash x \fatsemi \tau} \quad \text{T5}$$

$$\frac{\Gamma \vdash y \fatsemi \tau}{\Gamma \ddagger [x]T \vdash y \fatsemi \tau} \quad \text{T6}\,(y \notin \alpha[x]T)$$

F.5.2 Paragraphs

$$\frac{}{\Gamma \vdash [x] ::} \quad \text{T7}$$

$$\frac{\Gamma \vdash s \fatsemi \mathcal{P}\tau}{\Gamma \vdash x : s ::} \quad \text{T8}$$

$$\frac{\Gamma \vdash e \fatsemi \tau}{\Gamma \vdash x := e ::} \quad \text{T9}$$

$$\frac{\Gamma \vdash P \ \surd}{\Gamma \vdash P ::} \quad \text{T10}$$

$$\frac{\vdash \Gamma :: \qquad \Gamma \vdash \Pi ::}{\vdash \Gamma \ddagger \Pi \ ::} \quad \text{T11}$$

$$\frac{\Gamma \ddagger [x] \vdash T ::}{\Gamma \vdash [x]T ::} \quad \text{T12}\,(x \text{ not given set of } \Gamma)$$

$$\frac{\Gamma \vdash b \fatsemi \Sigma(x_1 := \tau_1, \ldots, x_n := \tau_n)}{\Gamma \vdash b ::} \quad \text{T13}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma(x_1 := \tau_1, \ldots, x_n := \tau_n)}{\Gamma \vdash S ::} \quad \text{T14}$$

F.5.3 Predicates

$$\frac{\Gamma \vdash e_1 \mathrel{\overset{\circ}{\circ}} \tau \qquad \Gamma \vdash e_2 \mathrel{\overset{\circ}{\circ}} \tau}{\Gamma \vdash e_1 = e_2\ \surd} \quad \text{T15}$$

$$\frac{\Gamma \vdash e \mathrel{\overset{\circ}{\circ}} \tau \qquad \Gamma \vdash s \mathrel{\overset{\circ}{\circ}} \mathcal{P}\tau}{\Gamma \vdash e \in s\ \surd} \quad \text{T16}$$

$$\frac{}{\Gamma \vdash \mathbf{true}\ \surd} \quad \text{T17}$$

$$\frac{}{\Gamma \vdash \mathbf{false}\ \surd} \quad \text{T18}$$

$$\frac{\Gamma \vdash P\ \surd}{\Gamma \vdash \neg\, P\ \surd} \quad \text{T19}$$

$$\frac{\Gamma \vdash P\ \surd \qquad \Gamma \vdash Q\ \surd}{\Gamma \vdash P \land Q\ \surd} \quad \text{T20}$$

$$\frac{\Gamma \vdash P\ \surd \qquad \Gamma \vdash Q\ \surd}{\Gamma \vdash P \lor Q\ \surd} \quad \text{T21}$$

$$\frac{\Gamma \vdash P\ \surd \qquad \Gamma \vdash Q\ \surd}{\Gamma \vdash P \Rightarrow Q\ \surd} \quad \text{T22}$$

$$\frac{\Gamma \vdash P\ \surd \qquad \Gamma \vdash Q\ \surd}{\Gamma \vdash P \Leftrightarrow Q\ \surd} \quad \text{T23}$$

$$\frac{\Gamma \vdash e \mathrel{\overset{\circ}{\circ}} \tau \qquad \Gamma \ddagger x := e \vdash P}{\Gamma \vdash \llparenthesis\, x := e \rrparenthesis \odot P\ \surd} \quad \text{T24}$$

$$\frac{\Gamma \vdash S :: \qquad \Gamma \ddagger S \vdash P\ \surd}{\Gamma \vdash \forall\, S \bullet P\ \surd} \quad \text{T25}$$

$$\frac{\Gamma \vdash S :: \qquad \Gamma \ddagger S \vdash P\ \surd}{\Gamma \vdash \exists\, S \bullet P\ \surd} \quad \text{T26}$$

$$\frac{\Gamma \vdash S :: \qquad \Gamma \ddagger S \vdash P\ \surd}{\Gamma \vdash \exists_1\, S \bullet P\ \surd} \quad \text{T27}$$

$$\frac{\Gamma \vdash S \mathrel{\overset{\circ}{\circ}} \mathcal{P}\Sigma(x_1 := \tau_1, \ldots, x_n := \tau_n) \qquad \Gamma \vdash x_1 \mathrel{\overset{\circ}{\circ}} \tau_1 \quad \ldots \quad \Gamma \vdash x_n \mathrel{\overset{\circ}{\circ}} \tau_n}{\Gamma \vdash S\ \surd} \quad \text{T28}$$

$$\frac{\Gamma \ddagger b \vdash P\ \surd}{\Gamma \vdash b \odot P\ \surd} \quad \text{T29}$$

F.5.4 Expressions

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{s} \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \ddagger \boldsymbol{x} : s \vdash \boldsymbol{x} \mathrel{⦂} \boldsymbol{\tau}} \quad \text{T30}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e} \mathrel{⦂} \boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \ddagger \boldsymbol{x} := \boldsymbol{e} \vdash \boldsymbol{x} \mathrel{⦂} \boldsymbol{\tau}} \quad \text{T31}$$

$$\frac{}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{i} \mathrel{⦂} \mathbb{Z}} \quad \text{T32}$$

$$\frac{}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{z} \mathrel{⦂} \mathbb{S}} \quad \text{T33}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \ddagger [\boldsymbol{x}] \boldsymbol{T} \vdash \boldsymbol{s} \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}' \qquad \boldsymbol{\mathit{\Gamma}} \ddagger [\boldsymbol{x}] \ddagger \boldsymbol{T} \vdash \boldsymbol{y} \mathrel{⦂} \boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \ddagger [\boldsymbol{x}] \boldsymbol{T} \vdash y_{[s]} \mathrel{⦂} \{\mho \boldsymbol{x} \mapsto \boldsymbol{\tau}'\}^{\boldsymbol{T}} \boldsymbol{\tau}} \quad \text{T34}$$

$$\frac{}{\boldsymbol{\mathit{\Gamma}} \ddagger [\boldsymbol{x}] \vdash \boldsymbol{x} \mathrel{⦂} \mathcal{P}(\mho \boldsymbol{x})} \quad \text{T35}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}_1 \mathrel{⦂} \boldsymbol{\tau} \quad \ldots \quad \boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}_n \mathrel{⦂} \boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \vdash \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\} \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}} \quad \text{T36}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \ddagger \boldsymbol{S} \vdash \boldsymbol{e} \mathrel{⦂} \boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \vdash \{\, \boldsymbol{S} \bullet \boldsymbol{e} \,\} \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}} \quad \text{T37}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{s} \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}}{\boldsymbol{\mathit{\Gamma}} \vdash \mathbf{Pow}\boldsymbol{s} \mathrel{⦂} \mathcal{P}(\mathcal{P}\boldsymbol{\tau})} \quad \text{T38}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}_1 \mathrel{⦂} \boldsymbol{\tau}_1 \quad \ldots \quad \boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}_n \mathrel{⦂} \boldsymbol{\tau}_n}{\boldsymbol{\mathit{\Gamma}} \vdash (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n) \mathrel{⦂} \chi(\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n)} \quad \text{T39}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{s}_1 \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}_1 \quad \ldots \quad \boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{s}_n \mathrel{⦂} \mathcal{P}\boldsymbol{\tau}_n}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_n \mathrel{⦂} \mathcal{P}\chi(\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n)} \quad \text{T40}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e} \mathrel{⦂} \chi(\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_i, \ldots, \boldsymbol{\tau}_n)}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}.\boldsymbol{i} \mathrel{⦂} \boldsymbol{\tau}_i} \quad \text{T41}\,(1 \leq i \leq n)$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e}_1 \mathrel{⦂} \boldsymbol{\tau}_1 \quad \cdots \quad \vdash \boldsymbol{e}_n \mathrel{⦂} \boldsymbol{\tau}_n}{\boldsymbol{\mathit{\Gamma}} \vdash \llparenthesis\, \boldsymbol{x}_1 := \boldsymbol{e}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{e}_n \,\rrparenthesis \mathrel{⦂} \Sigma(\boldsymbol{x}_1 := \boldsymbol{\tau}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{\tau}_n)} \quad \text{T42}$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{x}_1 \mathrel{⦂} \boldsymbol{\tau}_1 \quad \cdots \quad \boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{x}_n \mathrel{⦂} \boldsymbol{\tau}_n}{\boldsymbol{\mathit{\Gamma}} \vdash \theta\boldsymbol{S} \mathrel{⦂} \Sigma(\boldsymbol{x}_1 := \boldsymbol{\tau}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{\tau}_n)} \quad \text{T43}\,(\alpha\boldsymbol{S}=\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{b} \mathrel{⦂} \Sigma(\boldsymbol{x}_1 := \boldsymbol{\tau}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{\tau}_n)}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{b}.\boldsymbol{x}_i \mathrel{⦂} \boldsymbol{\tau}_i} \quad \text{T44}\,(1 \leq i \leq n)$$

$$\frac{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{f} \mathrel{⦂} \mathcal{P}\chi(\boldsymbol{\tau}', \boldsymbol{\tau}) \quad \boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{e} \mathrel{⦂} \boldsymbol{\tau}'}{\boldsymbol{\mathit{\Gamma}} \vdash \boldsymbol{f}\,\boldsymbol{e} \mathrel{⦂} \boldsymbol{\tau}} \quad \text{T45}$$

$$\frac{\Gamma \vdash s \fatsemi \mathcal{P}\tau \quad x : s \vdash P \ \surd}{\Gamma \vdash \mu\, x : s \mid P \fatsemi \tau} \quad \text{T46}$$

$$\frac{\Gamma \vdash P \ \surd \qquad \Gamma \vdash e_1 \fatsemi \tau \qquad \Gamma \vdash e_2 \fatsemi \tau}{\Gamma \vdash \textbf{if } P \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi} \fatsemi \tau} \quad \text{T47}$$

$$\frac{\Gamma \vdash b \fatsemi \Sigma(x_1 := \tau_1, \ldots, x_i := \tau_i, \ldots, x_n := \tau_n)}{\Gamma \ddagger b \vdash x_i \fatsemi \tau_i} \quad \text{T48}\,(1 \leq i \leq n)$$

$$\frac{\Gamma \ddagger b \vdash e \fatsemi \tau}{\Gamma \vdash b \odot e \fatsemi \tau} \quad \text{T49}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma(x_1 := \tau_1, x_i := \tau_i, \ldots, x_n := \tau_n)}{\Gamma \ddagger S \vdash x_i \fatsemi \tau_i} \quad \text{T50}\,(1 \leq i \leq n)$$

$$\frac{\Gamma \ddagger x := u \vdash e \fatsemi \tau}{\Gamma \vdash \langle\!\mid x := u \rangle \odot e \fatsemi \tau} \quad \text{T51}\,(1 \leq i \leq n)$$

F.5.5 Schema

$$\frac{\Gamma \vdash s_1 \fatsemi \mathcal{P}\tau_1 \quad \cdots \quad \Gamma \vdash s_n \fatsemi \mathcal{P}\tau_n}{\Gamma \vdash x_1 : s_1;\ \cdots;\ x_n : s_n \fatsemi \mathcal{P}\Sigma(x_1 := \tau_1, \ldots, x_n := \tau_n)} \quad \text{T52}$$

$$\frac{\Gamma \vdash S \fatsemi \tau \qquad S \vdash P \ \surd}{\Gamma \vdash S \mid P \fatsemi \tau} \quad \text{T53}$$

$$\frac{\Gamma \vdash S \fatsemi \tau}{\Gamma \vdash \neg S \fatsemi \tau} \quad \text{T54}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varrho \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varsigma}{\Gamma \vdash S \land T \fatsemi \mathcal{P}\Sigma(\varrho \sqcup \varsigma)} \quad \text{T55}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varrho \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varsigma}{\Gamma \vdash S \lor T \fatsemi \mathcal{P}\Sigma(\varrho \sqcup \varsigma)} \quad \text{T56}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varrho \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varsigma}{\Gamma \vdash S \Rightarrow T \fatsemi \mathcal{P}\Sigma(\varrho \sqcup \varsigma)} \quad \text{T57}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varrho \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varsigma}{\Gamma \vdash S \Leftrightarrow T \fatsemi \mathcal{P}\Sigma(\varrho \sqcup \varsigma)} \quad \text{T58}$$

$$\frac{\mathit{SOMETHING}}{\Gamma \vdash \mathsf{SProjection}} \quad \text{T59}$$

$$\frac{\mathit{SOMETHING}}{\Gamma \vdash S \backslash [x_1, \ldots, x_n]} \quad \text{T60}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varsigma \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varrho}{\Gamma \vdash \forall\, S \bullet T \fatsemi \mathcal{P}\Sigma(\varrho \mathbin{\backslash\backslash} \varsigma)} \quad \text{T61}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varsigma \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varrho}{\Gamma \vdash \exists\, S \bullet T \fatsemi \mathcal{P}\Sigma(\varrho \mathbin{\backslash\backslash} \varsigma)} \quad \text{T62}$$

$$\frac{\Gamma \vdash S \fatsemi \mathcal{P}\Sigma\varsigma \qquad \vdash T \fatsemi \mathcal{P}\Sigma\varrho}{\Gamma \vdash \exists_1\, S \bullet T \fatsemi \mathcal{P}\Sigma(\varrho \mathbin{\backslash\backslash} \varsigma)} \quad \text{T63}$$

$$\frac{\mathit{SOMETHING}}{\Gamma \vdash S \fatsemi T} \quad \text{T64}$$

$$\frac{\mathit{SOMETHING}}{\Gamma \vdash S^{q}} \quad \text{T65}$$

F.6 Inference rules

F.6.1 Structural rules

Assumption rules

$$\frac{}{\boldsymbol{\Gamma} \ddagger \boldsymbol{P} \vdash \boldsymbol{P}} \quad \mathit{AssumPred}$$

$$\frac{}{\boldsymbol{\Gamma} \ddagger \boldsymbol{x} := \boldsymbol{e} \vdash \boldsymbol{x} = \boldsymbol{e}} \quad \mathit{AssumDefin}\ (\mathbf{wf}(\boldsymbol{x} := \boldsymbol{e}))$$

$$\frac{}{\boldsymbol{\Gamma} \ddagger \boldsymbol{x} : \boldsymbol{s} \vdash \boldsymbol{x} \in \boldsymbol{s}} \quad \mathit{AssumDecl}\ (\mathbf{wf}(\boldsymbol{x} : \boldsymbol{s}))$$

$$\frac{}{\boldsymbol{\Gamma} \ddagger \boldsymbol{S} \vdash \boldsymbol{S}} \quad \mathit{SchemaAss}\ (\alpha \boldsymbol{S} \cap \phi \boldsymbol{S} = \varnothing)$$

Paragraph and thinning rules

$$\frac{\boldsymbol{P} \wedge \boldsymbol{R} \vdash \boldsymbol{Q}}{\boldsymbol{P} \ddagger \boldsymbol{R} \vdash \boldsymbol{Q}} \quad \uparrow\downarrow \mathit{PredConj}$$

$$\frac{\boldsymbol{\Gamma} \ddagger \boldsymbol{S} \ddagger \boldsymbol{P} \vdash \boldsymbol{Q}}{\boldsymbol{\Gamma} \ddagger \boldsymbol{S} \mid \boldsymbol{P} \vdash \boldsymbol{Q}} \quad \uparrow\downarrow \mathit{SchPred}$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{P}}{\boldsymbol{\Pi} \ddagger \boldsymbol{\Gamma} \vdash \boldsymbol{P}} \quad \mathit{Thinl}$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{P}}{\boldsymbol{\Gamma} \ddagger \boldsymbol{\Pi} \vdash \boldsymbol{P}} \quad \mathit{Thinr}\ (\alpha \boldsymbol{\Pi} \cap \phi \boldsymbol{P} = \varnothing)$$

$$\frac{\boldsymbol{\Gamma}_1 \ddagger \boldsymbol{\Gamma}_2 \ddagger \boldsymbol{\Pi} \vdash \boldsymbol{P}}{\boldsymbol{\Gamma}_1 \ddagger \boldsymbol{\Pi} \ddagger \boldsymbol{\Gamma}_2 \vdash \boldsymbol{P}} \quad \mathit{Shift} \left(\begin{array}{c} \alpha \boldsymbol{\Gamma}_2 \cap \phi \boldsymbol{\Pi} = \varnothing \\ \alpha \boldsymbol{\Pi} \cap \phi \boldsymbol{\Gamma}_2 = \varnothing \end{array} \right)$$

F.6.2 Equality and substitution

$$\frac{}{\boldsymbol{\Gamma} \vdash \boldsymbol{e} = \boldsymbol{e}} \quad \mathit{Refl}$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{u} = \boldsymbol{e}}{\boldsymbol{\Gamma} \vdash \boldsymbol{e} = \boldsymbol{u}} \quad \mathit{Symm}$$

$$\frac{\boldsymbol{\Gamma} \ddagger \boldsymbol{u} = \boldsymbol{e} \vdash \boldsymbol{v} = \boldsymbol{e}}{\boldsymbol{\Gamma} \ddagger \boldsymbol{u} = \boldsymbol{e} \vdash \boldsymbol{v} = \boldsymbol{u}} \quad \mathit{Trans}$$

$$\frac{\boldsymbol{\Gamma} \ddagger \boldsymbol{b} \vdash \boldsymbol{P}}{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \odot \boldsymbol{P}} \quad \uparrow\downarrow \mathit{UseBind}$$

$$\frac{\boldsymbol{\Gamma} \ddagger \boldsymbol{b} \vdash \boldsymbol{u} = \boldsymbol{e}}{\boldsymbol{\Gamma} \vdash \boldsymbol{u} = \boldsymbol{b} {\scriptstyle\odot} \boldsymbol{e}} \quad \uparrow\downarrow \mathit{EquBind}\ (\alpha \boldsymbol{b} \cap \phi \boldsymbol{u} = \varnothing)$$

F.6.3 Propositional calculus

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \land Q} \quad \mathit{AndI}$$

$$\frac{\Gamma \vdash P \land Q}{\Gamma \vdash P} \quad \mathit{AndEr}$$

$$\frac{\Gamma \vdash P \land Q}{\Gamma \vdash Q} \quad \mathit{AndEl}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \lor Q} \quad \mathit{OrIr}$$

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \lor Q} \quad \mathit{OrIl}$$

$$\frac{\Gamma \vdash P \lor Q \qquad \Gamma \ddagger P \vdash R \qquad \Gamma \ddagger Q \vdash R}{\Gamma \vdash R} \quad \mathit{OrE}$$

$$\frac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \quad \mathit{impI}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash P \Rightarrow Q}{\Gamma \vdash Q} \quad \mathit{impE}$$

$$\frac{\Gamma \vdash \mathbf{false}}{\Gamma \vdash P} \quad \mathit{falseE}$$

$$\frac{\Gamma \ddagger \neg\, P \vdash \mathbf{false}}{\Gamma \vdash P} \quad \mathit{notE}$$

$$\frac{\Gamma \vdash P \Rightarrow \mathbf{false}}{\Gamma \vdash \neg\, P} \quad \uparrow\downarrow\ \mathit{notDef}$$

$$\frac{}{\Gamma \vdash \mathbf{false} \Rightarrow P} \quad \mathit{falseDef}$$

$$\frac{}{\Gamma \vdash P \Rightarrow \mathbf{true}} \quad \mathit{trueDef}$$

$$\frac{\Gamma \vdash P \Rightarrow Q \land Q \Rightarrow P}{\Gamma \vdash P \Leftrightarrow Q} \quad \uparrow\downarrow\ \mathit{iffDef}$$

F.6.4 Quantifier rules

$$\frac{\Gamma \ddagger S \vdash P}{\Gamma \vdash \forall\, S \bullet P} \quad \uparrow\downarrow\ \mathit{AllI}$$

$$\frac{\Gamma \vdash \forall\, S \bullet P \qquad \Gamma \vdash b \in S}{\Gamma \vdash b \odot P} \quad \mathit{AllE}$$

$$\frac{\Gamma \vdash b \odot P \qquad \Gamma \vdash b \in S}{\Gamma \vdash \exists\, S \bullet P} \quad \mathit{ExistsI}$$

$$\frac{\Gamma \vdash \exists\, S \bullet P \qquad \Gamma \ddagger S \ddagger P \vdash Q}{\Gamma \vdash Q} \quad \mathit{ExistsE}$$

$$\frac{\Gamma \vdash \exists\, S \bullet P \land \mathit{SOMETHING}}{\Gamma \vdash \exists_1\, S \bullet P} \quad \uparrow\downarrow\ \mathit{UniqExists}$$

F.6.5 Expression rules

Sets

$$\frac{\boldsymbol{\Gamma}\ddagger[\boldsymbol{x}]\boldsymbol{T} \vdash \{\, \langle\!|\ \boldsymbol{x} := \boldsymbol{s} |\!\rangle \odot \boldsymbol{T} \bullet \boldsymbol{y} \,\} \in \mathbf{Pow}\boldsymbol{e}}{\boldsymbol{\Gamma}\ddagger[\boldsymbol{x}]\boldsymbol{T} \vdash \boldsymbol{y}_{[\boldsymbol{s}]} \in \boldsymbol{e}} \quad \uparrow\downarrow \mathit{GenMem}\,(y \in \alpha\boldsymbol{T}, \mathbf{wf}\boldsymbol{T})$$

$$\frac{\boldsymbol{\Gamma} \vdash (\forall\, \boldsymbol{x} : \boldsymbol{s} \bullet \boldsymbol{x} \in \boldsymbol{t}) \land (\forall\, \boldsymbol{x} : \boldsymbol{t} \bullet \boldsymbol{x} \in \boldsymbol{s})}{\boldsymbol{\Gamma} \vdash \boldsymbol{s} = \boldsymbol{t}} \quad \uparrow\downarrow \mathit{Seteq}\,(\boldsymbol{x} \notin \phi s \cup \phi t)$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{v} = \boldsymbol{e}_1 \lor \ldots \lor \boldsymbol{v} = \boldsymbol{e}_n}{\boldsymbol{\Gamma} \vdash \boldsymbol{v} \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}} \quad \uparrow\downarrow \mathit{Extmem}$$

$$\frac{\boldsymbol{\Gamma} \vdash \exists\, \boldsymbol{S} \bullet \boldsymbol{e} = \boldsymbol{u}}{\boldsymbol{\Gamma} \vdash \boldsymbol{e} \in \{\, \boldsymbol{S} \bullet \boldsymbol{u} \,\}} \quad \uparrow\downarrow \mathit{Setcomp}\,(\phi\mathbf{e} \cap \alpha\boldsymbol{S} = \varnothing)$$

$$\frac{\boldsymbol{\Gamma} \vdash \forall\, \boldsymbol{x} : \boldsymbol{t} \bullet \boldsymbol{x} \in \boldsymbol{s}}{\boldsymbol{\Gamma} \vdash \boldsymbol{t} \in \mathbf{Pow}\boldsymbol{s}} \quad \uparrow\downarrow \mathit{Powerset}\,(\boldsymbol{x} \notin \phi s)$$

Cartesian products

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{v} = \boldsymbol{e}_i}{\boldsymbol{\Gamma} \vdash \boldsymbol{v} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n).\boldsymbol{i}} \quad \mathit{Tupleequ}\,(1 \leq i \leq n)$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{u}.1 \in \boldsymbol{s}_1 \land \ldots \land \boldsymbol{u}.n \in \boldsymbol{s}_n}{\boldsymbol{\Gamma} \vdash \boldsymbol{u} \in \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_n} \quad \uparrow\downarrow \mathit{Prodmem}$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{u} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)}{\boldsymbol{\Gamma} \vdash \boldsymbol{u}.1 = \boldsymbol{e}_1 \land \ldots \land \boldsymbol{u}.n = \boldsymbol{e}_n} \quad \uparrow\downarrow \mathit{Tuplesel}$$

Labelled products

$$\frac{}{\boldsymbol{\Gamma} \vdash \langle\!|\ \boldsymbol{x}_1 := \boldsymbol{e}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{e}_n |\!\rangle\ .\boldsymbol{x}_i = \boldsymbol{e}_i} \quad \mathit{BindEqu}\,(1 \leq i \leq n)$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{y}.\boldsymbol{x}_1 = \boldsymbol{e}_1 \land \ldots \land u.\boldsymbol{x}_n = \boldsymbol{e}_n}{\boldsymbol{\Gamma} \vdash \boldsymbol{y} = \langle\!|\ \boldsymbol{x}_1 := \boldsymbol{e}_1, \ldots, \boldsymbol{x}_n := \boldsymbol{e}_n |\!\rangle} \quad \uparrow\downarrow \mathit{BindSel}$$

$$\frac{}{\boldsymbol{\Gamma}\ddagger\boldsymbol{b} \vdash \boldsymbol{x} = \boldsymbol{b}.\boldsymbol{x}} \quad \mathit{BindMem}\,(\boldsymbol{x} \in \alpha\boldsymbol{b}, \mathbf{wf}\boldsymbol{b})$$

Schemas

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{e}.\boldsymbol{x}_1 = \boldsymbol{x}_1 \land \cdots \land \boldsymbol{e}.\boldsymbol{x}_n = \boldsymbol{x}_n \qquad \boldsymbol{\Gamma} \vdash \boldsymbol{S}}{\boldsymbol{\Gamma} \vdash \boldsymbol{e} = \theta\boldsymbol{S}} \quad \mathit{ThetaEqu}$$

$$\frac{\vdash \langle\!|\ \boldsymbol{x}_1{:=}\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n{:=}\boldsymbol{x}_n |\!\rangle\ \in \boldsymbol{S}}{\vdash \boldsymbol{S}} \quad \uparrow\downarrow \mathit{BindSch}\,(\alpha S{=}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$$

Description

$$\frac{\Gamma \vdash (e, u) \in f \land \forall\, y : f \bullet (y.1 = e) \Rightarrow (y.2 = e\,y)}{\Gamma \vdash u = f\,e} \quad \uparrow\downarrow\ \mathit{FunctApp}\ (y \notin \phi e \cup \phi u)$$

$$\frac{\begin{array}{c} \Gamma \vdash e \in s \\ \Gamma \vdash \langle\!| x := e |\!\rangle \odot P \\ \Gamma \ddagger y : s \vdash \langle\!| x := y |\!\rangle \odot P \Rightarrow y = e \end{array}}{\Gamma \vdash e = \mu\, x : s \mid P} \quad \mathit{DefnDescr}$$

$$\frac{\Gamma \vdash P \Rightarrow e_1 = e \land \neg\, P \Rightarrow e_2 = e}{\Gamma \vdash \mathbf{if}\ P\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2\ \mathbf{fi} = e} \quad \uparrow\downarrow\ \mathit{IfThenElse}$$

$$\frac{\Gamma \vdash v = \langle\!| x := u |\!\rangle \odot e}{\Gamma \ddagger x := u \vdash v = e} \quad \uparrow\downarrow\ \mathit{Usedef}\ (x \notin \phi v)$$

F.6.6 Schema calculus

$$\frac{\Gamma \vdash u \in [x_1 : s_1;\ \ldots;\ x_n : s_n]}{\Gamma \vdash u.x_1 \in s_1 \land \ldots \land u.x_n \in s_n} \quad \uparrow\downarrow\ \mathit{BindProd}$$

$$\frac{\Gamma \vdash b \in S \land b \odot P}{\Gamma \vdash b \in [S \mid P]} \quad \uparrow\downarrow\ \mathit{SchemaMem}$$

$$\frac{\Gamma \vdash b \odot S}{\Gamma \vdash b \in S} \quad \uparrow\downarrow\ \mathit{SchBindMem}\ (\mathbf{wf}\, S)$$

$$\frac{\Gamma \vdash b \odot [b \odot S]}{\Gamma \vdash b \in S} \quad \uparrow\downarrow\ \mathit{SchBindMem}'\ (\mathbf{wf}\, b)$$

$$\frac{\Gamma \vdash \neg\, b \odot S}{\Gamma \vdash b \in \neg\, S} \quad \uparrow\downarrow\ \mathit{SNot}\ (\mathbf{wf}\, S)$$

$$\frac{\Gamma \vdash b \odot S \land b \odot T}{\Gamma \vdash b \in (S \land T)} \quad \uparrow\downarrow\ \mathit{SAnd}\ (\mathbf{wf}\, S \land T)$$

$$\frac{\Gamma \vdash b \odot S \lor b \odot T}{\Gamma \vdash b \in (S \lor T)} \quad \uparrow\downarrow\ \mathit{Sor}\ (\mathbf{wf}\, S \lor T)$$

$$\frac{\Gamma \vdash b \odot S \Rightarrow b \odot T}{\Gamma \vdash b \in (S \Rightarrow T)} \quad \uparrow\downarrow\ \mathit{SImp}\ (\mathbf{wf}\, S \Rightarrow T)$$

$$\frac{\Gamma \vdash b \odot S \Leftrightarrow b \odot T}{\Gamma \vdash b \in (S \Leftrightarrow T)} \quad \uparrow\downarrow\ \mathit{SIff}\ (\mathbf{wf}\, S \Leftrightarrow T)$$

$$\frac{\Gamma \vdash \exists\, S \bullet b \odot T}{\Gamma \vdash b \in \exists\, S \bullet T} \quad \uparrow\downarrow\ \mathit{SExists}\ (\phi T \cap (\alpha b \cup \alpha S) = \varnothing)$$

$$\frac{\boldsymbol{\Gamma} \vdash \forall\, \boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{T}}{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in \forall\, \boldsymbol{S} \bullet \boldsymbol{T}} \quad \uparrow\downarrow\, \mathit{SAll}\,(\phi \boldsymbol{T} \cap (\alpha b \cup \alpha \boldsymbol{S}) = \varnothing)$$

$$\frac{\boldsymbol{\Gamma} \vdash \exists_1\, \boldsymbol{S} \bullet \boldsymbol{b} \odot \boldsymbol{T}}{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in \exists_1\, \boldsymbol{S} \bullet \boldsymbol{T}} \quad \uparrow\downarrow\, \mathit{SUniqExists}\,(\phi \boldsymbol{T} \cap (\alpha \boldsymbol{b} \cup \alpha \boldsymbol{S}) = \varnothing)$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in \exists\, \boldsymbol{x}_1 : \boldsymbol{s}_1;\ \cdots;\ \boldsymbol{x}_n : \boldsymbol{s}_n \bullet \boldsymbol{S}}{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in \boldsymbol{S} \backslash [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]} \quad \uparrow\downarrow\, \mathit{SHide}$$

$$\frac{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in (\boldsymbol{S} \wedge \boldsymbol{T}) \backslash [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]}{\boldsymbol{\Gamma} \vdash \boldsymbol{b} \in \boldsymbol{S} \upharpoonright \boldsymbol{T}} \quad \uparrow\downarrow\, \mathit{SProj}\,(\alpha \boldsymbol{S} \setminus \alpha \boldsymbol{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\})$$

$$\frac{\mathit{SOMETHING}}{\boldsymbol{\Gamma} \vdash \boldsymbol{S} \fatsemi \boldsymbol{T}} \quad \uparrow\downarrow\, \mathit{SComp}$$

$$\frac{\mathit{SOMETHING}}{\boldsymbol{\Gamma} \vdash \boldsymbol{S}^q} \quad \uparrow\downarrow\, \mathit{SDecor}$$

□

G References – Informative Annex

Notes on this section of the Z Standard

Section title: References
Section editor: John Nicholls
Source file: infref.tex
Most recent update: 13th September 1995
Formatted: 14th September 1995

References

[1] Abrial, J-R., "A Course on System Specification," Lecture Notes, Programming Research Group, University of Oxford, 1981.

[2] Arthan, R.D., "Modules for Z," ZSRC Document 181, 3 August 1995, available from `ftp.comlab.ox.ac.uk`.

[3] Bowen, J. P., "Select Z Bibliography," available from newsgroup `comp.specification.z`.

[4] Brien, S.M., Gardiner, P.H.B., Lupton, P.J., Woodcock, J.C.P., "A Semantics for Z," in preparation, 1992.

[5] Brien, S.M., Nicholls, J.E., *Z Base Standard* Version 1.0, Working Draft of the Z Standards Panel. Also published as Technical Monograph PRG-107, Oxford University Computing Laboratory, 1992.

[6] BSI Standard **BS 0** : Part 1 : 1981, *A standard for standards. Part 1. Guide to general principles of standardization*, British Standards Institution, 1991.

[7] BSI Standard **BS 6154**, *Method of defining syntactic metalanguage*,