EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

A formal semantics for Z and the link between Z and the relational algebra

Citation for published version (APA):
van Diepen, M. J., & Hee, van, K. M. (1989). *A formal semantics for Z and the link between Z and the relational algebra*. (Computing science notes; Vol. 8917). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1989

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

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

Link to publication

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

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

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 21. Oct. 2021

A formal semantics for Z and the link between Z and the relational algebra

by

M.J. van Diepen and K.M. van Hee

89/17

December, 1989

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
Editors: prof.dr.M.Rem
prof.dr.K.M. van Hee

A formal semantics for Z and the link between Z and the relational algebra

M.J. van Diepen and K.M. van Hee

Abstract

A formal semantics for Z based on naive settheory is presented. In this approach a so-called Zbase is postulated to be used in the definition of a Zscript (a specification). From a Zbase we may construct a more powerfull Zbase by a Zscript. This allows us to work in a modular way. In our approach the semantics of a schema is a table from the relation datamodel. This suggests a link with the relational algebra and this link is further explored.

1 Introduction

The Z notation is a language and style for expressing formal specifications of computing systems and is invented by J.R. Abrial. It is based on a typed set theory, and the notion of a 'schema' is one of its key features. A schema combines a collection of typed variables with a relationship specified by a predicate (or some axioms) and Z provides notations for defining schemas and later combining them in various ways so that large specifications can be built up in stages. We assume the reader to be familiar with the Z notation, see for instance Hayes ([Hay87]) and Spivey ([Spi88]).

The Z notation is very close to the notation of tables in the relational algebra. In fact Z allows to define finite or even infinite tables in an implicit way, while the relational algebra only manipulates explicitly defined finite tables. This feature of Z gives it its power to use tables for specification of operations over infinite domains. One of our goals is to analyse the relationship between Z and the relational algebra in more detail. Both Z and the relational algebra are notations and so they should have semantics. We will show that the relational algebra can be embedded into Z. We use the notation of a script: a set of definitions in which a name is given to expressions. In fact we construct a mapping ψ from Rscripts, i.e. scripts in the relational algebra, to Zscripts and a partial mapping ϕ the other way around such that ψ and ϕ are semantics preserving, $\phi \circ \psi$ is the identity on Rscript and $\psi \circ \phi$ is a subset of the identity on Zscripts. To this end we need a formal semantics of both Z and the relational algebra. We have chosen the naive, untyped settheory for our semantic domain. This allows us to interpret Z-schema expressions and expressions in the relational algebra as subsets of a generalised cartesian product, called tables.
Our semantics is more intuitive than Spivey's. Another difference with Spivey's approach is that we don't want to define well-known concepts, like the natural numbers and their operators (In Hayes these operators are not considered!). Spivey only assumes the set concept to be known. We introduce the concept of a Zbase. A Zbase contains names for already given functions, including constants, types and schemas, and in addition a function that assigns to these names their meaning. This, so-called interpretation function is not defined explicitly, it is assumed to be known in some form. On top of a Zbase one may define a Zscript using the Z notation. Each Zscript, can be used to define a new Zbase, which is an extension of the Zbase of the Zscript. Our second goal is to demonstrate this way of modular definition of semantics. It resembles a modular style of specifications too, because a designer will use a Zbase, appropriate to its application domain, and he will not use the formal definitions of the concepts of this Zbase.
Our Z notation differs on minor points from the notation proposed in [Hayes87] and [Spivey88].

We did not include several notations to keep our treatment concise. We made a strict distinction between types, functions and schemas. Only types can be used to define the domain of a variable in a schema definition, in a set definition or to bound a variable by a quantor in a logical expression. Schemas are not allowed in these places. This is not a strong restriction because we have a tuple type which is in fact a schema without a restricting logical expression. So if one wants to use a schema as a type, one defines a tuple type by deleting the logical expression of the schema and one adds this logical expression as a conjunct to the logical expression of the schema definition, the set definition or the quantified expression. Another point of difference is the definition of functions. One wants to define functions in a Zscript that can be used in several schema definitions. So it should be global definitions. Hayes is not very precise on this point. Spivey has a special construct to define functions globally, it may be considered as an unnamed schema. We reserve one schema, called Context, to define all the functions that should be defined globally. In fact we consider this schema to be included in all other schemas. With regard to recursion we adopt the same view as Spivey and Hayes do. Our notation allows recursive functions to be defined, but does not allow recursive type and schema definitions.

The paper is organised as follows. In section 2 we define the concept of a Zbase, we give a syntax for our restricted version of the Z notation, we sketch the construction of an interpretation function for a Zscript given a Zbase, and finally we give the construction of a new Zbase. The details of context conditions of the syntax and of the construction of the interpretation function can be found in appendices B and C. Appendix A gives a glossary of the used mathematical notation.
In section 3 we follow the same approoah for the relational algebra. Here we also make a clear distinction between syntax and semantics. Details regarding context conditions and the construction of the interpretation function can be found in appendices D and E. Finally in section 4 we consider the embedding of the relational algebra in Z. Many details are given in appendices F and G.
In [Bjørner82] also the relational algebra is considered but from a totally different point of view. There the VDM language is used to define the relational model while we try to compare the relational model as a specification language to the Z language.

2 From Zbase to Zbase

We start with the definition of a Zbase. In a Zbase the precise structure of the objects type set, function and schema is not important. For the semantics of functions and schemas we only have to know their signatures and an interpretation function.

The variables, functions, schemas etc. find their names in an alphabet A of names. TN, SN, FN, CN, and VN are mutually disjoint subsets of A and denote respectively the set of type set names, the set of schemanames, the set of functionnames, the set of constantnames, and the set of variablenames.

Definition1 Formally a Zbase is a 6–tuple $<$tn, gn, sgn_F, sn, sgn_S, I$>$ where

- tn is a subset of TN, gn is a subset of FN $\cup$ CN and sn is a subset of SN,
- sgn_F is a total function from gn to tn* (tn* denotes the set of all finite sequences of elements of tn). The function sgn_F gives the signature of every element of gn as a sequence of type set names. In this representation the sequence $t_0 \cdot t_1 \cdot \ldots \cdot t_n$, $n \geq 0$ denotes $t_0 \leftarrow t_1 \times \ldots \times t_n$. The arity of element f$\in$gn can now be defined as the length of sequence sgn_F(f) minus one (len(sgn_F(f))$-$1). A constant is an element of gn with arity nill (no arguments). The set of constants is denoted by CN. The set of functions is denoted by fn.

- sgn_S is a total function from sn to the set of partial functions from VN to tn.
- I is an interpretation function for tn, gn and sn. The language to define I is based on the naive, untyped settheory or Z itself. I is defined such that
  - for every type set name $t \in tn$, I(t) is a set.
  - for every functionname $f \in fn$ with $sgn_F(f)=t_0 \cdot t_1 \cdot \ldots \cdot t_n$ and $n \geq 1$, I(f) is a mapping from $I(t_1) \times \ldots \times I(t_n)$ to $I(t_0)$. Note that for a constantname $c \in CN$ with signature t, $I(c) \in I(t)$.
  - for every schemaname $s \in sn$, I(s) is a subset of $\prod(I \circ sgn_S(s))$ where for a setvalued function F, $\prod(F)$ is defined by $\prod(F)=\{f \mid f \text{ is a function over } dom(F) \text{ and } \forall_{x \in dom(f)} \cdot f(x) \in F(x)\}$. $\prod$ is called the generalised cartesian product.

For example, a possible Zbase would be

$$
\begin{array}{ll}
< & \{nat\} \\
, & \{+, *, 0, 1, \ldots\} \\
, & \{< +, nat^3 >, < *, nat^3 >, < 0, nat>, \ldots \} \\
, & \{S\} \\
, & \{<S, \{<x, nat>,<y, nat>\}\} \\
, & I \\
> &
\end{array}
$$

where I is defined such that

- nat denotes the set of naturals.
- + and * denote respectively the addition and multiplication of naturals.
- true, false, 0, 1, ... denote respectively true, false, 0, 1, ...
- $I(S)=\{t \in \prod(\{<x, I(nat)>, <y, I(nat)>\}) \mid t(x) = 2 \times t(y)\}$.

2.1 Syntax and Semantics of the Z notation

A Zbase forms the base of a Zscript. Types find their base in type set names, predicates in functionnames and schema expressions in schemanames and explicitly defined schemas.

2.1.1 Meta-syntax

We first define a meta-syntax. This meta-syntax is BNF with the following extensions :

1. Any underlined part of the syntax must be taken literally.
2. Any part of the syntax between '<>' triangular brackets may be repeated; each such repetition must be preceded with a comma ','.
   So a ::= < b > is shorthand for a ::= b | b,a.
3. Any part of the syntax between '[]' brackets may be omitted.
   So a ::= [b]c is shorthand for a ::= c|bc.
4. VARS denotes the set (FN ∪ VN ∪ CN)\gn.

2.1.2 Syntax

Each Z notation depends on a Zbase. Of course the syntax does not depend on the interpretation function of the base. Our syntax is closely related to Hayes'.
Given a Zbase $<$ tn, fn, sgn_F, sn, sgn_S, I $>$ the syntax is given by

Type expressions

$typex$ ::= tn
 | $\underline{\mathcal{P}(}typex\underline{)}$
 | $\underline{(}typex\underline{\times}...\underline{\times}typex\underline{)}$
 | $typex\underline{\rightarrow}typex$
 | $\underline{[}<\mathrm{VN}\underline{:}typex>\underline{]}$

The simplest types are the type set names of tn. More complex types can be build in combination with the type constructors $\mathcal{P}$ (powerset), $\rightarrow$ (function) and a special tuple type constructor $[x_1 : T_1, \ldots, x_n : T_n]$. The schema type constructor provides in our need of schemas to be used as types.

Logical expressions

$constant$ ::= CN
 | $\underline{\{}<constant>\underline{\}}$
 | $\underline{\leq}\ constant\underline{,}...\underline{,}constant\ \underline{\geq}$
 | $\underline{\leq}<\mathrm{VN}\underline{:}constant>\underline{\geq}$

$term$ ::= $constant$
 | $\mathrm{VN}[\ \underline{(}term\underline{)}]$
 | $\mathrm{FN}\underline{(}<term>\underline{)}$
 | $\underline{\{}<\mathrm{VN}\ \underline{:}typex>\underline{|}logex\underline{\}}$

$atom$ ::= $term\ \underline{\Theta}\ term$
 where $\Theta : \in, \subset, =, \leq, \geq, <, >$

$logex$ ::= $atom$
 | $logex\ \underline{\Theta}\ logex$ where $\Theta : \wedge, \vee, \Rightarrow, \Leftrightarrow$
 | $\underline{\neg}logex$
 | $\underline{\forall}\underline{[}\mathrm{VN}\underline{:}typex\underline{|}logex\underline{]}$
 | $\underline{\exists}\underline{[}\mathrm{VN}\underline{:}typex\underline{|}logex\underline{]}$

Note the difference between tuple types and set types.
For example,
[x:nat, y:nat | x = y] denotes a set of tuples $\{t \in \prod(\{<x, N>, <y, N>\}) \mid t(x) = t(y)\}$
and { x:nat, y:nat | x = y } denotes the set of pairs $\{ <x,y> \mid x \in N \wedge y \in N \wedge x = y \}$,
where N is the set of naturals denoted by nat.

Schemas

schema ::= [<VARS:(*typex* |TN) > |*logex*]

Schema-expressions

sexp ::=	*schema*	
	|sn	
	| *sexp*$\triangle$*sexp*	(conjunction)
	| $\neg$*sexp*	(negation)
	| *sexp*|*logex*	(restriction)
	| *sexp*$\lceil$(<VN>)	(projection)
	| *sexp*\(<VN>)	(hiding)
	| *sexp*[<VN\VN>]	(renaming)
	| *pre* *sexp*	(precondition)
	| *post* *sexp*	(postcondition)
	| *sexp*$\otimes$*sexp*	(composition)
	| *sexp*$\gg$*sexp*	(piping)

Schema definitions

sdef ::= SN := *sexp*

Type definitions

tdef ::= TN := *typex*

Function definitions

fdefs ::= Context := *schema*

Zscript

zscript ::= [[< *tdef* >],[*fdefs*],[< *sdef* >]]

Remark All new function (constant) definitions are encapsulated within one schema. The schema name Context is reserved for this purpose. The variables declared in the schema Context must of course denote functions or constants and are treated as global variables of the other schemas which are part of the Zscript. This semantics condition is further elaborated in the next section.

Remark The syntax takes care of the fact that we don't want (for simplicity) that a name given to a type expression can itself be part of another type expression defined in the same Zscript. The same applies to the use of schemanames. Note that this excludes recursive type- and schema- definitions.

Not every Zscript, satisfying the syntax rules, does qualify as a Zscript of a Zbase. There are several context conditions a Zscript must satisfy. For instance, each variable declared in the schema Context must have a type of the form t or $t_0 \leftarrow t_1 \times \ldots \times t_n$ where $t, t_0, \ldots t_n$ are typenames and each such typename must be an element of tn or there must be a type definition with the same name in the left-hand side (of the ':=' sign). Some important context conditions are given in appendix B, the obvious ones are omitted.

We conclude this section by given an example Zscript. Given the example Zbase in the forgoing section, a possible Zscript would be

Context := [faculty : nat → nat |
  ∀[n: nat |
    ((n = 0 ∨ n = 1) ⇒ faculty(n) = 1) ∧
    (n ≥ 2 ⇒ faculty(n) = n * faculty(n - 1))]],
T := [i? : nat, j! : nat | j! = faculty(i?)]

2.2 Semantics

The next step is to give the formal semantics for the Zscript. This implicitely defines the semantics of the Z notation. The semantics are given by the function J. The language to define J is based on the naive, untyped settheory.
A schema or schema expression describes a relation between the variables declared within the schema or schema expression, that only holds if the predicate of the schema or schema expression holds. So, the schema $[x : \text{nat}; y : \mathcal{P}\text{nat} \mid x \in y]$ describes the set of all tuples (x, y) where x is of type nat and y is of type $\mathcal{P}$nat that satisfy $x \in y$.
The semantics of a Z expression depends on the context in which the expression must be evaluated. The context of an expression is modelled by a value called environment. An environment is a partial function from the set of names into the semantic domain, resolving any ambiguities concerning the meaning of names. The meaning of an expression is determined once an environment establishes the context for the expression. The environment for an expression must be such that each name in the expression is given a semantic value. The environment is extended each time a name is introduced. Context condition 12 takes care of the fact that the extended environment remains a functions.
In the sequel we assume the semantics of expression E to be defined in the environment denoted by e, unless stated otherwise. $J_i(E)$ denotes the semantics of expression E to be evaluated in environment i. Only in those cases where the semantics are to be defined in an extended environment this is explicitly defined. $J_{n/v}(E)$ denotes the semantics of the expression E in the environment extended with the name, semantic value pair $< n, v >$. The meaning of a name n in the environment e is given by $e(n)$.
The environment of a Zscript of a Zbase equals the interpretation function of the Zbase. The semantics of the Zscript is defined as the the extension of the interpretation function of the Zbase with the semantic value assignments to the schema Context (if present) and every defined schemaname and type set name in the Zscript.
The variables declared in the schema Context must denote functions or constants. There are at least two solutions to define global functions in the schema Context. One solution is to require that the designer of a specification only makes specifications such that there is only one function that satisfies the logical expression. Hence in this case the semantics of the schema Context contains one tuple.
Another solution is to use the standard approach of denotational semantics [Schmidt88] to consider on all types in a Zbase the discrete partial ordering for lifted domains. Then we may define a function specified in Context as the smallest function that satisfies the logical expression. For a full definition of the semantics function J we refer to appendix C. The definition proceeds along the usual lines of denotational semantics ([Schmidt88]).

We will explain the workings of the function J by the example Zbase and Zscript introduced in the forgoing sections. In fact, we will only determine the semantics of the schema Context.
Let $l1 = ((n = 0 \vee n = 1) \Rightarrow faculty(n) = 1)$.
Let $l2 = (n \geq 2 \Rightarrow faculty(n) = n * faculty(n))$.
The interpretation function i of the Zbase assigns the names $nat, *, 0, 1, \ldots$ there semantic value, which respectively are $N, \times, 0, 1, \ldots$.

$$J_i(Context) = \{x \in \prod(\{< faculty, J(nat \to nat)\}) \mid J_{faculty/x.faculty}(\forall[n : nat \mid l1 \wedge l2])\}$$

$$= \{x \in \prod(\{< faculty, N \to N)\}) \mid \forall \tilde{n} \in J(nat) : J_{n/\tilde{n}}(l1 \wedge l2)\}$$

$$= \{x \in \prod(\{< faculty, N \to N)\}) \mid \forall \tilde{n} \in N : ((\tilde{n} = 0 \vee \tilde{n} = 1) \Rightarrow x.faculty(\tilde{n}) = 1) \wedge (\tilde{n} \geq 2 \Rightarrow x.faculty(\tilde{n}) = \tilde{n} \times x.faculty(\tilde{n})).$$

2.3 Construction of a new Zbase

The construction of a new Zbase out of an old Zbase and a Zscript for that old Zbase is straightforward. The old Zbase is copied directly in the new Zbase and because the constructs of the Zscript meet the requirements imposed on a Zbase, these constructs can be easily incorporated in the new Zbase. The new Zbase can be found by applying the rules given hereafter.

Let a Zbase < tn, gn, sgn$_F$, sn, sgn$_S$, I > be given.
Let ntn be the set of type set names defined in the Zscript Zs.
Let nfn be the set of functions(constants) defined in the Zscript Zs (V(Context)).
Let nsn be the set of schemas defined in the Zscript Zs.
Let j be the semantics of the Zscript Zs. j gives in addition to the semantic value assignments to type set names, function names and schema names in the Zbase (the function i), each name in ntn, nfn and nsn a semantic value.
The new Zbase is defined by < tn', gn', sgn'$_F$, sn', sgn'$_S$, I' > where

- tn' = tn ∪ ntn,
- fn' = gn ∪ nfn,
- sgn'$_F$(f) = sgn$_F$(f), f∈gn,
  sgn'$_F$(f) = VT(Context)(f), f∈nfn,
- sn' = sn ∪ nsn,
- sgn'$_S$(s) = sgn$_S$(s), s∈sn,
  sgn'$_S$(s) = VT(se), s∈nsn and se is the schema expression associated with s in Zs by $s := se$,
- I'(n) = j(n), n∈tn ∪ ntn ∪ gn ∪ sn ∪ nsn,
  I'(n) = j(Context)(n), ∈nfn.

Using this construction one may build up libraries of specifications. We only have to start with one or more primitive Zbases and then we form new ones. If an application designer understands a Zbase he can use it without going back to define all the schemas used. If an application designer understands a Zbase he can use it without going back to define all the schemas used.

3 Syntax and semantics of relational algebra

A similar iterative process as seen in the preceding sections can be applied to relational algebra. For an informal definition of the relational algebra we refer to Ullman ([Ull82]). The differences stem from the facts that

1. functions are not part of relational algebra,
2. only the relational operators $=, \leq, \geq, <, >$ are allowed,
3. new constants cannot be defined,
4. domain (type) constructs are not possible, i.e. we cannot define new domains,
5. an attribute can only have one domain,
6. infinite relations and implicit definitions of relations are not possible.

In our approach an attribute can be associated with several types and relations can be infinite.

3.1 Rbase

The names to denote constants, domains, tables and attributes come from an alphabet A of names. CN, DN, TABN, and AN are mutually disjoint subsets of A and denote respectively the set of constantnames, the set of domains, the set of tablenames, and the set of attributenames.

A Rbase is a 6-tuple $<$ dn, cn, sgn_C, tabn, sgn_T, I $>$ where

- dn is a subset of DN, cn is a subset of CN and tabn is a subset of TABN'
- $\mathrm{sgn}_C \in$ cn $\rightarrow$ dn, domain of each constant,
- $\mathrm{sgn}_T \in$tabn $\rightarrow$ (AN $\not\rightarrow$ dn),
- I is an interpretation function for dn, cn and tabn. The definition of I is based on the naive, untyped settheory
  I is defined such that
  - for every domainname d$\in$dn, I(d) is a set,
  - for every constantname c$\in$cn, I(c)$\in$I(sgn_C(c)),
  - for every tablename tab$\in$tabn, I(tab) $\subseteq \prod$(I $\circ$ sgn_T(tab)).

3.2 Syntax

A Rbase forms the base of a Rscript. Given the Rbase $<$ dn, cn, sgn_C, tabn, sgn_T, I $>$ the syntax is given by

Relational logical expressions

constant ::= cn

term ::= *constant*
 | AN

$atom ::= term\ \underline{\Theta}\ term$
where $\Theta : =, \leq, \geq, <, >$

$rlogex ::= atom$
$\mid rlogex\ \underline{\Theta}\ rlogex$ where $\Theta : \wedge, \vee, \Rightarrow, \Leftrightarrow$
$\mid \underline{\neg} rlogex$

Table expressions

$texp ::=$ tabn	
$\mid texp \underline{\cup} texp$	(union)
$\mid texp \underline{-} texp$	(difference)
$\mid texp \underline{\bowtie} texp$	(join)
$\mid texp \underline{\Uparrow\{} < \text{AN} > \underline{\}}$	(projection)
$\mid \underline{\sigma(} texp \underline{;} rlogex \underline{)}$	(selection)
$\mid \underline{\rho(} texp \underline{;[} < \text{AN} \underline{\backslash} \text{AN} > \underline{])}$	(renaming)

table definitions

$tabdef ::= \text{TABN}\ \underline{:=} texp$

Rscript

$rscript ::= [< tabdef >]$

For some important context conditions, we refer to appendix D.

3.3 Semantics

The semantics of a Rscript for a Rbase are given by the function J and proceeds along the same lines as for the Zscript. Only the extensions of the environment are explicated. The environment of the Rscript of a Rbase is given by the interpretation function of the Rbase. For a full definition of the semantics function J we refer to Appendix E.

Given a Rbase $<$ dn, cn, sgn_C, tabn, sgn_T, I $>$ and the set ntabn of table names defined in a Rscript Rs where j is the interpretation of Rs, i.e. the extension of the interpretation function I of the Rbase with the semantic value assignments to the tablenames of ntabn, the resulting Rbase is defined by $<$ dn', cn', sgn'_C, tabn', sgn'_T, J $>$ where

- tabn' = tabn $\cup$ ntabn, ntabn $\cap$ tabn = $\emptyset$
- sgn'_T(tab) = sgn_T(tab), tab$\in$tabn,
  sgn'_T(tab) = AV(te), tab$\in$ntabn and te is the table expression associated with tab by Rs.

4 Link between Z and relational algebra

In this section we formalise the link between Z and the relational algebra, by constructing two mappings, one from Zbases to Rbases and one the other way round. Given these two mappings we construct two other mappings, one semantics preserving mapping from Zscripts to Rscripts and one semantics preserving mapping from Rscripts to Zscripts. In fact we show the following:

- For every Zbase there is a Rbase such that there is a partial semantics preserving function from Zscripts to Rscripts. The domain of this function is the set of Zscripts satisfying

1. an empty type definition part,
2. an empty function definition part,
3. with respect to the .. | .. schema operator, restriction to relational logical operators only,
4. restriction to the schema names of the Zbase as the only base operands, i.e. schema is not allowed as part of a schema expression.
   For example the expression [x: nat; y: nat | x = y] may not be part of a schema expression. In the relational algebra tables cannot be defined implicitly.

- The other way around we can show that for every Rbase there is a Zbase such that there is a total semantics preserving function from Rscripts to Zscripts.

So, in the first case, we have to show the existence of a semantics preserving function from schema expressions obeying rules 3 and 4 to table expressions. Of course the schema expressions must satisfy the rules imposed on Zscripts. The same applies to the table expressions, in the second case, with regard to the Rscripts.

4.1 From Z to the Relational Algebra

We construct a trivial total function F from Zbases to Rbases and for every Zbase Zb a semantics preserving function $\phi(s)$ from schema expressions for Zbase Zb, satisfying rules 3 and 4 and the rules imposed on a Zscript, to table expressions for Rbase F(Zb), satisfying the rules imposed on Rbases. In the sequel we assume all schema expressions to satisfy rules 3 and 4 and the rules imposed on a Zscript.

Let a Zbase Zb = $<$ tn, gn, sgn_F, sn ,sgn_S, i $>$ be given.

The function F With every schema name in sn we associate a table name. Due to the negation operator there is a need for a table for each variable declaration in any of the schemas of the Zbase. To each variable declaration we assign a name, which will become a table name in the corresponding Rbase. The tablename corresponding with the declaration x : t is given the same semantics as trhe schema [x : t | true]. The set of variable declarations can be extracted from the signature function sgn_S. Let dtn be the set of these tablenames and let d be the bijection assigning to each variable declaration a tablename in dtn.

The Rbase Rb corresponding to Zb is given by $<$ dn, cn, sgn_C, tabn, sgn_T, j $>$ where

- dn equals the set of type set names tn.

- cn equals the set of constants in gn.
- $\mathrm{sgn}_C(\mathrm{c}) = \mathrm{sgn}_F(\mathrm{c})$, c∈cn.
- tabn = sn ∪ dtn, sn ∩ dtn = ∅.
- $\mathrm{sgn}_T(\mathrm{tab}) = \mathrm{sgn}_S(\mathrm{tab})$, tab∈sn,
  $\mathrm{sgn}_T(\mathrm{tab}) = \{<\mathrm{x,t}>\}$, tab∈dtn and tab is assigned to the declaration x : t, i.e. $\mathrm{d}^{-1}(\mathrm{tab}) = \mathrm{x : t}$.
- j(t) = i(t), t∈tn,
  j(c) = i(c), c∈cn,
  j(tab) = i(tab), tab∈sn,
  $\mathrm{j(tab)} = \prod(\{< x, \mathrm{i}(t) >\})$, tab∈dtn and tab is assigned to the declaration $x : t$, i.e. $\mathrm{d}^{-1}(\mathrm{tab}) = x : t$.

The function $\phi(s)$ The function $\phi(s)$ from schema expressions E for Zb to table expressions for Rb, satisfying the rules for Rscripts, is defined by

1. **If** E is a schemaname s ∈sn
   then $\phi(E) = s$.
2. **If** E is of the form $s_1 \wedge s_2$ where s_1, s_2 are schema expressions
   then $\phi(E) = \phi(s_1) \bowtie \phi(s_2)$.
3. **If** E is of the form $\neg s$ where s is a schema expression
   then $\phi(E) = \bowtie \{\mathrm{d}(v : VT(s)(v)) \mid v \in V(s)\} \setminus \phi(s)$.
4. **If** E is of the form $s \mid l$ where s is a schema expression and l is a logical expression
   then $\phi(E) = \sigma(\phi(s); l)$.
5. **If** E is of the form $s\lceil(\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
   then $\phi(E) = \phi(s) \Uparrow \vec{v}$
6. **If** E is of the form $s \setminus (\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
   then $\phi(E) = \phi(s) \Uparrow (V(s) \setminus \vec{v})$
7. **If** E is of the form $s[\vec{w} \setminus \vec{v}]$ where $v_1, \ldots, v_n, w_1, \ldots, w_n \in$ VN and s is a schema expression
   then $\phi(E) = \rho(\psi(s); [\vec{w} \setminus \vec{v}])$.
8. **If** E is of the form *pre* s where s is a schema expression
   then $\phi(E) = \psi(s) \Uparrow (V(s) \setminus (DH \cup OP))$.
9. **If** E is of the form *post* s where s is a schema expression
   then $\phi(E) = \psi(s) \Uparrow (V(s) \setminus (UD \cup IP))$.
10. **If** E is of the form $s_1 \otimes s_2$ where s_1, s_2 are schema expressions
    then $\psi(E) = (\rho(\phi(s_1); [h_1(\vec{v}) \setminus \vec{v}]) \bowtie \rho(\phi(s_2)); [h_2(\vec{z}) \setminus \vec{z}]) \Uparrow (V(s_1) - \setminus SA) \cup (V(s_2) \setminus SB)$.
    where
    - $SA = \{v \in V(s_1) \cap DH \mid \exists w \in V(s_2) \cap UD \bullet \mathit{basename}(v) = w\}$.
    - $SB = \{v \in V(s_2) \cap UD \mid \exists w \in V(s_1) \cap DH \bullet \mathit{basename}(w) = v\}$.
    - $\{v_1, \ldots, v_n\} = V(s_1)$ and h_1 is defined as in point 27 Appendix C.
    - $\{z_1, \ldots, z_n\} = V(s_2)$ and h_2 is defined as in point 27 Appendix C.

11. **If** E is of the form $s_1 \gg s_2$ where s_1, s_2 are schema expressions **then** analogous to point 10.

Let I' be the interpretation of the schema expressions and J' be the interpretation of the table expressions. The schema expressions are evaluated in the environment i (interpretation function Zbase) and the the table expressions in the environment j (interpretation function Rbase). The function $\phi(s)$ has the property that each variable declared in a schema expression becomes an attribute with the same name in the corresponding table expression. Moreover the variable and its atrribute counterpart have the same type (domain) and due to the semantic functions I' and J' the type and corresponding domain have the same semantics.

Lemma1 For each schema expression s the following holds

1. $V(s) = A(\phi(s))$.
2. $VT(s) = AV(\phi(s))$.
3. $I' \circ VT(s) = J' \circ AV(\phi(s))$.

(We omit the proof; it is trivial)

Theorem1 For every schema expression s, $I'(s) = J'(\phi(s))$ holds.

The proof is given in Appendix F and uses induction on the structure of schema expressions.

4.2 From the relational algebra to Z

We associate to each Rbase a Zbase, the total function G, and we construct for every Rbase Rb a semantics preserving function ψ from table expressions for Rb, satisfying the rules imposed on Rscripts, to schema expressions for Zbase G(Rb), satisfying the rules imposed on Zscripts.

Let a Rbase Rb = $<$ dn, cn, sgn$_C$, tabn, sgn$_T$, j $>$ be given.

The function G The corresponding Zbase Zb is defined by $<$ tn, gn, sgn$_F$, sn, sgn$_S$, i $>$where

- tn = dn.
- gn = cn.
- sgn$_F$(c) = sgn$_C$(c), c$\in$cn.
- sn = tabn.
- sgn$_S$(s) = sgn$_T$(s), s$\in$tabn.
- i(d) = j(d), d$\in$dn,
  i(c) = j(c), c$\in$fn,
  i(s) = j(s), s$\in$tabn.

The function ψ The function ψ is defined by

1. **If** E is a table name tab $\in$tabn
then $\psi(E) \;=\; tab.$

2. **If** E is of the form $te_1 \;\cup\; te_2$ where te_1, te_2 are table expressions
then $\psi(E) \;=\; \neg(\neg\psi(te_1) \;\wedge\; \neg\psi(te_2))$

3. **If** E is of the form $te_1 \;-\; te_2$ where te_1, te_2 are table expressions
then $\psi(E) \;=\; \psi(te_1) \;\wedge\; \neg\psi(te_2)$

4. **If** E is of the form $te_1 \;\bowtie\; te_2$ where te_1, te_2 are table expressions
then $\psi(E) \;=\; \psi(te_1) \;\wedge\; \psi(te_2)$

5. **If** E is of the form $te \Uparrow \vec{a}$ where $a_1, \ldots, a_n \in$ AN and te is a table expression
then $\psi(E) \;=\; \psi(te)\lceil(\vec{a})$

6. **If** E is of the form $\sigma(te; rl)$ where te is a table expression and rl is a relational logical expression
then $\psi(E) \;=\; \psi(te) \mid rl$

7. **If** E is of the form $\rho(te; [\vec{b} \setminus \vec{a}])$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in$ AN and te is a table expression
then $\psi(E) \;=\; \psi(te)[\vec{b} \setminus \vec{a}]$

Let J' be the interpretation given to the table expressions and I' the interpretation of the schema expressions. The environment of the table expressions is given by the interpretation function j of the Rbase and that of the schema expressions by the interpretation function i of the Zbase. The function ψ has the properties that each attribute in a table expression becomes a variable with the same name in the corresponding schema expression, the domain of each attribute has the same name as the type of the corresponding variable, and the domain and corresponding type have the same semantics.

Lemma2 For each table expression te the following holds

1. $A(te) = V(\psi(te))$.
2. $AV(te) = VT(\psi(te))$.
3. $J' \circ AV(te) = I' \circ VT(\psi(te))$.

(We omit the proof)

Theorem2 For every table expression te, $J'(te) = I'(\psi(te))$ holds.

The proof is given in appendix G and uses induction on the structure of table expressions.

4.3 Summary

We have constructed a trivial total function F from Zbases to Rbases and a partial function ϕ from Zscripts to Rscripts such that for every Zbase Zb and every 'correct' Zscript Zs of that Zbase Zb, the Rscript ϕ(Zs) is a Rscript of the Rbase F(Zb). Moreover the function ϕ is such that for every Zbase Zb and every 'correct' Zscript Zs of Zb, the semantics of the Zscript Zs in environment Zb.I is the same as that of the Rscript ϕ(Zs) in environment F(Zb).I, where Zb.I is the interpretation function of Zb and F(Zb).I is the interpretation function of the Rbase F(Zb). The other way around we constructed a total function G from Rbases to Zbases and a total semantics preserving function ψ from Rscripts to Zscripts.
It can be easily verified that $\phi \circ \psi$ is the identity on Rscripts, i.e. for every Rbase Rb and every Rscript of Rs it is true that $\phi(\psi(\text{Rs})) = \text{Rs}$. Besides this, $\psi \circ \phi$ is a subset of the identity on Zscripts, i.e. for every Zbase Zb and every 'correct' Zscript Zs of Zb, $\psi(\phi(\text{Zs})) = \text{Zs}$.

5 Conclusions

A simple formal semantics of Z, based on naive, untyped settheory, is developed. The approach opens the possibility of a modular way of specifications. This is important for the reusability of specifications.
In this approach the semantics of a schema is a possible infinite table. And there the link with the relational algebra occurs. In fact Z turns out to be an extension of the relational algebra, namely by implicitly defined tables.

References

[Bjø82] D. Bjørner and C.B. Jones
Formal semantics and software development, Prentice hall, 1982, chptr 12.

[Hay87] I. Hayes
Specification case studies, Prentice Hall, 1987.

[Sch88] D.A. Schmidt
Denotational Semantics, Brown Publishers, 1988.

[Ull82] J.D. Ullman
Principles of database systems, 2nd edition, Computer Science Press, 1982.

[Spi88] J.M. Spivey
Understanding Z, Cambridge University Press, 1988.

A Mathematical notation

- $\upharpoonright$, restriction
  If f is a function and A a set then $f \upharpoonright A = \{< x, y > \in f \mid x \in A\}$
- $\circ$, function composition
  If f and g are functions then $f \circ g = \{< x, g(f(x)) > \mid x \in dom(f) \wedge f(x) \in dom(g)\}$
- $\oplus$
  For function collections $\mathcal{F}$ and $\mathcal{G}$ with resp. domains F and G
  $\mathcal{F} \oplus \mathcal{G} = \{h \mid dom(h) = F \cup G \wedge h \upharpoonright F \in \mathcal{F} \wedge h \upharpoonright G \in \mathcal{G}\}$
  Note $F = G \Rightarrow \mathcal{F} \oplus \mathcal{G} = F \cup G$
- $\prod$, generalised product
  for a setvalued function F, $\prod$(F) is defined by
  $\prod$(F)={f | f is a function over dom(F) and $\forall_{x \in dom(f)} \cdot$f(x)$\in$F(x)}.
- The operator ∞ is defined as follows :
  If A and B are sets and T is a set of functions over A and h is a bijection from B into A then T$\infty h = \{t \circ h \mid t \in$ T$\}$.
- $\vec{v}$ is shorthand for $\{v_1, \ldots, v_n\}$ for arbitrary n.
- $(\vec{v})$ is shorthand for $(v_1, \ldots, v_n)$ for arbitrary n.
- $[\vec{w} \setminus \vec{v}]$ is shorthand for $[v_1 \setminus w_1, \ldots, v_n \setminus w_n]$ for arbitrary n.
- $\vec{v} : \vec{t}$ is shorthand for $v_1 : t_1, \ldots, v_n : t_n$ for arbitrary n.
- $\vec{v} := \vec{t}$ is shorthand for $v_1 := t_1, \ldots, v_n := t_n$ for arbitrary n.

- setcomplement, for sets V and W, $V^c = W \setminus V$.
- $\not\rightarrow$, partial function.

B Context conditions for a Zscript

The syntax rules have to obey several context rules. We only give some important context conditions, the obvious ones are omitted. We first introduce the functions V and VT.
The function $VT : sexp \not\rightarrow (\text{FN} \cup \text{CN} \cup \text{VN} \not\rightarrow \text{TN}^*)$ gives for each schema expression, satisfying the context conditions imposed on the syntax, the declared variables and their type. So, the function VT gives the signature for each schema expression. The signature for a schema name s in a schema definition of the form $s := se$ equals $VT(se)$.
The function $V : sexp \not\rightarrow \mathcal{P}(\text{FN} \cup \text{CN} \cup \text{VN})$ gives for each schema expression, satisfying the context conditions, the declared variables.

W.r.t. schema expressions denoting operations, i.e. a schema consisting of input, output, state-before, and state-after components and a predicate relating these parts, we state the following conventions:

- UD = 'set of all undashed variables in FN $\cup$ CN $\cup$ VN not ending in ? of !' (state-before components).
- DH = 'set of all dashed variables in FN $\cup$ CN $\cup$ VN' (state-after components).
- IP = 'set of all variables in FN $\cup$ CN $\cup$ VN ending in ?' (inputs).
- OP = 'set of all variables in FN $\cup$ CN $\cup$ VN ending in !' (outputs).
- The function *basename* assigns to every variable in FN $\cup$ CN $\cup$ VN the variable stripped of its declarations (?,!,').

The function VT is recursively defined by:

1. **If** E is a schema name $s \in sn$ **then** $VT(s) = \text{sgn}_S(s)$.
2. **If** E is of the form $[\vec{v} : \vec{t} \mid l]$ where $v_1, \ldots, v_n \in$ VARS , $t_1, \ldots, t_n$ are type expressions and l is a logical expression
   then $VT(E) = \{< v_1, S(t_1) >, \ldots, < v_n, S(t_n) >\}$
   where $S(t) = t$ for a type name t,
   and $S(t) = t_1 \ldots t_n t_0$ for $t = t_0 \leftarrow t_1 \times t_2 \times \ldots \times t_n$.
3. **If** E is of the form $s_1 \wedge s_2$ or $s_1 \otimes s_2$ where s_1, s_2 are schema expressions
   then $VT(E) = VT(s_1) \cup VT(s_2)$.
4. **If** E is of the form $\neg(s)$ where s is a schema expression
   then $VT(E) = VT(s)$.
5. **If** E is of the form $s \mid l$ where s is a schema expression and l is a logical expression
   then $VT(E) = VT(s)$.
6. **If** E is of the form $s\lceil(\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
   then $VT(E) = VT(s) \uparrow \vec{v}$.
7. **If** E is of the form $s \setminus (\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
   then $VT(E) = VT(s) \uparrow \vec{v}^c$.
8. **If** E is of the form $s[\vec{w} \setminus \vec{v}]$ where $v_1, \ldots, v_n, w_1, \ldots, w_n \in$ VN and s is a schema expression and $\{v_1, \ldots, v_n\} = dom(VT(s))$
   then $VT(E) = VT(s) \circ h$

where h is the bijection from $\{w_1, \ldots, w_n\}$ into $\{v_1, \ldots, v_n\}$ that satifies $\forall 1 \leq i \leq n \bullet h(w_i) = v_i$

9. **If** E is of the form *pre* s where s is a schema expression **then** $VT(E) = VT(s) \uparrow (DH \cup OP)^c$.
10. **If** E is of the form *post* s where s is a schema expression **then** $VT(E) = VT(s) \uparrow (UD \cup IP)^c$.
11. **If** E is of the form $s_1 \gg s_2$ where s_1, s_2 are schema expressions **then** $VT(E) =$
$$(VT(s_1) \uparrow \{v \in dom(VT(s_1)) \cap OP \mid (\exists w \in dom(VT(s_2)) \cap IP \bullet basename(v) = basename(w))\})$$
$$\cup$$
$$(VT(s_1) \uparrow \{v \in dom(VT(s_2)) \cap IP \mid (\exists w \in dom(VT(s_1)) \cap OP \bullet basename(v) = basename(w))\})$$

The function V is defined by: $V(s) = dom(VT(s))$.

Context-conditions

1. Each constantname in a constant expression must be a constantname in cn or there must be a variable declaration in Context where the same name is declared as a constant.
2. Each functionname in a term expression must be a functionname in fn or there must be a variable declaration in Context where the same name is declared as a function (not a constant).
3. Each variable declared in the schema Context must be in FN $\cup$ CN and must have a type of the form t or $t_0 \leftarrow t_1 \times \ldots \times t_n$ where $t, t_0, \ldots t_n \in$ TN and each such type name must be an element of tn or there must be a type definition with the same name in the left-hand side (of the ':=' sign).
4. Each variable declared in a schema expression being part of a schema definition (not Context) must be in VN and must have its type in TN and each such typename must be an element of tn or there must be a type definition with the same name in the left-hand side (of the ':=' sign).
5. A term expression of the form $f(\vec{tm})$ where f is a function and $tm_1, \ldots, tm_n$ are terms is allowed if f has arity n and for all $i : 1 \leq i \leq n$ the type of tm_i is a subtype of $\mathrm{sgn}_F(i)$.
6. In a schema of the form $[\vec{v} : \vec{t} \mid l]$ where $v_1, \ldots, v_n \in$ VN , $t_1, \ldots, t_n$ are type expressions and l is a logical expression the only free variables in l may be the variables $v_1, \ldots, v_n$, the elements of fn and the variables declared in Context.
7. A schema expression of the form $s_1 \wedge s_2$ or $s_1 \otimes s_2$ where s_1, s_2 are schema expressions must satisfy $\forall v \in V(s_1) \cap V(s_2) \bullet VT(s_1)(v) = VT(s_2)(v)$.
8. In a schema expression of the form $s \mid l$ where s is a schema expression and l is a logical expression the only free variables in l may be the variables $V(s)$, the elements of fn and the variables declared in Context.
9. A schema expression of the form $s[\vec{w} \setminus \vec{v}]$ where s is a schema expression and $v_1, \ldots, v_n, w_1, \ldots, w_n \in$ VN must satisfy the following conditions
a) $\{v_1, \ldots, v_n\} = V(s)$
b) $\forall 1 \leq i < j \leq n \bullet w_i \neq w_j$

10. In case of the operators *pre*, *post*, $\otimes$ and $\gg$ the schema expressions must denote operations.
11. A schema expression of the form $s_1 \gg s_2$ where s_1, s_2 are schema expressions must satisfy
    a) $\forall v \in V(s_1) \cap V(s_2) \bullet VT(s_1)(v) = VT(s_2)(v)$.
    b) $\forall v \in V(s_1) \cap OP \bullet$
    $(\exists w \in V(s_2) \cap IP \bullet basename(v) = basename(w) \Rightarrow VT(s_1)(v) = VT(s_2)(w))$
    c) $\forall v \in V(s_2) \cap IP \bullet$
    $(\exists w \in V(s_1) \cap OP \bullet basename(v) = basename(w) \Rightarrow VT(s_1)(w) = VT(s_2)(v))$
12. The variables declared in a schema either in the declaration part or in the predicate part in a set definition or as a quantified expression, must be all different.
13. The names assigned to schema expressions and type expressions may not be in sn $\cup$ tn.

C The semantics of Z

The semantics are given by the function J and the language to define J is based on the naive, untyped settheory. Only the extension of the environment is explicated. Under the assumption that all context conditions are satisfied the function J is defined by :

1. **If** E is of the form n where n is a name, and e is the environment
   then $J_e(n) = e(n)$.
2. **If** E is of the form $\mathcal{P}(t)$ where t is a type expression
   then $J(E) = power(J(t))$.
3. **If** E is of the form $(t_1 \times, \ldots, \times t_n)$ where $t_1, \ldots, t_n$ are type expressions
   then $J(E) = product(J(t_1), \ldots, J(t_n))$.
4. **If** E is of the form $t_1 \rightarrow t_0$ where t_1, t_0 are type expressions
   then $J(E) = J(t_1) \rightarrow J(t_0)$.
5. **If** E is of the form $[\vec{v} : \vec{t}]$ where $v_1, \ldots, v_n \in$ VN and $t_1, \ldots, t_n$ are type expressions
   then $J(E) = \prod(\{< v_1, J(t_1) >, \ldots, < v_n, J(t_n) >\})$.
6. **If** E is a constantname c
   then $J(E) = J(\text{Context})(c)$.
7. **If** E is of the form $\vec{c}$ where $c_1, \ldots, c_n \in$ CN
   then $J(E) = J(\vec{c})$.
8. **If** E is of the form $< \vec{c} >$ where $c_1, \ldots, c_n \in$ CN
   then $J(E) = < J(\vec{c}) >$.
9. **If** E is of the form $< \vec{v} : \vec{c} >$ where $v_1, \ldots, v_n \in$ VN and $c_1, \ldots, c_n \in$ CN
   then $J(E) = < \vec{v} : J(\vec{c}) >$.
10. **If** E is of the form $f(\vec{tm})$ where f is a functionname and $tm_1, \ldots, tm_n$ are terms
    then if f is a function in fn
    then $J(E) = J(f) \cdot (J(\vec{tm}))$
    else (f is a function in Context) $J(E) = (J(\text{Context})(f)) \cdot (J(\vec{tm}))$.
11. **If** E is of the form $v(\vec{tm})$ where $v \in$ VN and $tm_1, \ldots, tm_n$ are terms
    then $J(E) = J(v) \cdot (J(\vec{tm}))$.
12. **If** E is of the form $\{\vec{v} : \vec{t} \mid l\}$ where $v_1, \ldots, v_n \in$ VN , $t_1, \ldots, t_n$ are type expressions en l is a logical expression and $v_1, \ldots, v_n$ are the only free variables in l
    then $J(E) = \{\vec{x} : J(\vec{t}) \mid J_{\vec{v}/\vec{x}}(l) = true\}$.

13. **If** E is of the form $tm_1\ \Theta\ tm_2$ where tm_1, tm_2 are terms en $\Theta : \in, \subset, \leq, \geq, =, >, <$
then if $J(tm_1)$ and $J(tm_2)$ are both defined and comparable
then true **if** $J(tm_1)\ \Theta J(tm_2)$ is true
else false
else undef.

14. **If** E is of the form $l_1\ \Theta\ l_2$ where l_1, l_2 are logical expressions and $\Theta : \wedge, \vee, \Rightarrow, \Leftrightarrow$
then the semantics of $l_1\ \Theta\ l_2$ is given by the truth table for three-valued logic.

15. **If** E is of the form $\neg l$ where l is a logical expression
then if $J(l)$ = true **then** $J(E)$ = false.
if $J(l)$ = false **then** $J(E)$ = true.
else $J(l)$ = undef.

16. **If** E is of the form $\forall[v : t \mid l]$ where $v \in$ VN, t is a type expression and l is a logical expression and v is the only free variable in l
then if for all $x \in J(t) : J_{v/x}(l)$ = true **then** $J(E)$ = true.
if for one $x \in J(t) : J_{v/x}(l)$ = false **then** $J(E)$ = false.
else $J(E)$ = undef.

17. **If** E is of the form $\exists[v : t \mid l]$ where v is a varname, t is a type expression and l is a logical expression and v is the only free variable in l
then if for one $x \in J(t) : J_{v/x}(l)$ = true **then** $J(E)$ = true.
if for all $x \in J(t) : J_{v/x}(l)$ = false **then** $J(E)$ = false.
else$J(E)$ = undef.

18. **If** E is of the form $[\vec{v} : \vec{t} \mid l]$ where $v_1, \ldots, v_n \in$ VARS , $t_1, \ldots, t_n$ are type expressions and l is a logical expression
then $J(E) = \{x \in \prod(\{< v_1, J(t_1) >, \ldots, < v_n, J(t_n) >\})$
$\mid J_{\vec{v}/x\vec{:}v}(l) = true\}$.

19. **If** E is of the form $s_1\ \wedge s_2$ where s_1, s_2 are schema expressions
then $J(E)\ =\ J(s_1) \oplus J(s_2)$.

20. **If** E is of the form $\neg s$ where s is a schema expression
then $J(E)\ =\ \prod(J \circ VT(s)) \setminus J(s)$.

21. **If** E is of the form $s \mid l$ where s is a schema expression and l is a logical expression and $\{v_1, \ldots, v_n\} = V(s)$
then $J(E)\ =\ \{x \in J(s) \mid J_{\vec{v}/x\vec{:}v}(l) = true\}$.

22. **If** E is of the form $s \setminus (\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
then
$J(E) = \{x \uparrow \vec{v}^c \mid x \in J(s)\}$.

23. **If** E is of the form $s\lceil(\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression
$J(E) = \{x \uparrow \vec{v} \mid x \in J(s)\}$,

24. **If** E is of the form $s[\vec{w} \setminus \vec{v}]$ where $v_1, \ldots, v_n, w_1, \ldots, w_n \in$ VN and s is a schema expression
then $J(E)\ =\ J(s)\infty h$
where h is the bijection from $\{w_1, \ldots, w_n\}$ into $\{v_1, \ldots, v_n\}$ that satisfies
$\forall 1 \leq i \leq n\ \bullet\ h(w_i) = v_i$.

The operator ∞ is defined as follows :
If A and B are sets and T is a set of functions over A and h is a bijection from B into A **then** $\mathrm{T}\infty h = \{t \circ h \mid t \in \mathrm{T}\}$.

25. **If** E is of the form *pre* s where s is a schema expression
$J(E) = \{x \uparrow (OP \cup DH)^c \mid x \in J(s)\}$.

26. **If** E is of the form *post* s where s is a schema expression
$J(E) = \{x \uparrow (IP \cup UD)^c \mid x \in J(s)\}$.

27. **If** E is of the form $s_1 \otimes s_2$ where s_1, s_2 are schema expressions **then** $J(E) = \{x \uparrow (V(s_1) \setminus SA) \cup (V(s_2) \setminus SB) \mid x \in J(s_1) \infty h_1^{-1} \oplus J(s_2) \infty h_2^{-1}\}$
    - $SA = \{v \in V(s_1) \cap DH \mid \exists w \in V(s_1) \cap UD \bullet \mathit{basename}(v) = w\}$.
    - $SB = \{v \in V(s_2) \cap UD \mid \exists w \in V(s_2) \cap DH \bullet \mathit{basename}(w) = v\}$.
    - h_1 is a bijection from $V(s_1)$ into $B \cup (V(s_1) \setminus SA)$ and h_2 is a bijection from $V(s_2)$ into $B \cup (V(s_2) \setminus SB)$ where

      (-) B is a set of names, $B \cap ((V(s_1) \setminus SA) \cup (V(s_2) \setminus SB)) = \{\}$, $\mid B \mid = \mid SA \mid = \mid SB \mid$

      (-) $\forall v \in SA \bullet h_1(v) \in B$

      (-) $\forall v \in SB \bullet h_2(v) = h_1(w)$ where $w \in SA$ and $\mathit{basename}(w) = v$

      (-) $\forall v \in V(s_1) \setminus SA \bullet h_1(v) = v$

      (-) $\forall v \in V(s_2) \setminus SB \bullet h_2(v) = v$
28. **If** E is of the form $s_1 \gg s_2$ where s_1, s_2 are schema expressions **then** analoqous with the preceding item with the only difference that instead of the state-after components and state-before components the inputs and outputs are of importance.
29. **If** E is of the form $\vec{s} := \vec{se}$ where $s_1, \ldots, s_n \in$ SN and $se_1, \ldots, se_n$ are schema expressions **then** $J(E) = \{< s_i, J(se_i) > \mid 1 \leq i \leq n\}$.
30. **If** E is of the form $\vec{t} := \vec{te}$ where $t_1, \ldots, t_n \in$ TN and $te_1, \ldots, te_n$ are type expressions **then** $J(E) = \{< t_i, J(te_i) > \mid 1 \leq i \leq n\}$.
31. **If** E is of the form Context $:= se$ where se is a schema **then** $J(E) = \{< \text{Context}, J(se) >\}$.
32. **If** E is of the form $\vec{s} := \vec{se}, \text{Context} := se, \vec{t} := \vec{te}$ where $s_1, \ldots, s_n \in$ SN, $se_1, \ldots, se_n$ are schema expressions $t_1, \ldots, t_n \in$ TN and $te_1, \ldots, te_n$ are type expressions and se is a schema **then** $J_i(E) = i \cup J(\vec{t} := \vec{te})$
    $$\cup J_{\vec{t}/J(\vec{te})}(\text{Context} := se)$$
    $$\cup J_{\vec{t}/J(\vec{te}), \text{Context}/J_{\vec{t}/J(\vec{te})}(se)}(\vec{s} := \vec{se})$$

D Context conditions for a Rscript

We introduce the functions A and AV.
The function $AV : \mathit{texp} \not\to (\text{ AN} \not\to \text{dn })$ gives for each table expression, satisfying all context conditions, the attributes and their domains, and is recursively defined by :

1. **If** E is a table name $t \in \mathit{tabn}$ **then** $AV(E) = \text{sgn}_T(t)$.
2. **If** E is of the form $te_1 \cup te_2$ or $te_1 - te_2$ where te_1, te_2 are table expressions **then** $AV(E) = AV(te_1)$.
3. **If** E is of the form $te_1 \bowtie te_2$ where te_1, te_2 are table expressions **then** $AV(E) = AV(te_1) \cup AV(te_2)$.
4. **If** E is of the form $te \Uparrow \vec{a}$ where $a_1, \ldots, a_n \in$ AN and te is a table expression **then** $AV(E) = AV(te) \uparrow \vec{a}^c$.

5. **If** E is of the form $\sigma(te; rl)$ where te is a table expression and rl is a relational logical expression
   then $AV(E) = AV(te)$.
6. **If** E is of the form $\rho(te; [\vec{b} \setminus \vec{a}])$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in$ AN and te is a table expression
   then $AV(E) = AV(te) \circ h$
   where h is the bijection from $\{b_1, \ldots, b_n\}$ into $\{a_1, \ldots, a_n\}$ that satisfies
   $\forall 1 \leq i \leq n \bullet h(b_i) = v_i$

The function $A : texp \nrightarrow \mathcal{P}(\text{AN})$ gives for each table expression te, satisfying all context conditions, its attributes and is defined by :
$A(te) = dom(AV(te))$

Context conditions

1. A table expression of the form $te_1 \cup te_2$ or $te_1 - te_2$ where te_1, te_2 are table expressions must satisfy
   $AV(te_1) = AV(te_2)$.
2. In a table expression of the form $\sigma(te; rl)$ where te is a table expression and rl is a relational logical expression only the attributes $A(te)$ are allowed to be free in rl.
3. A table expression of the form $\rho(te; [\vec{b} \setminus \vec{a}])$ where te is a table expression and $a_1, \ldots, a_n, b_1, \ldots, b_n \in$ AN must satisfy
   a) $\{a_1, \ldots, a_n\} = A(te)$
   b) $\forall 1 \leq i < j \leq n \bullet b_i \neq b_j$
4. the names assigned to table expressions must be disjoint to tabn.

E The semantics of the relational algebra

The semantics are given by the function J and the language to define J is based on naive, untyped settheory. Under the assumption that all context condition are satisfied the function J is defined by :

1. **If** E is of the form n where n is a name, and e is the environment
   then $J_e(E) = e(n)$.
2. **If** E is of the form $tm_1 \ \Theta \ tm_2$ where tm_1, tm_2 are terms and $\Theta : \leq, \geq, =, >, <$
   then if $J(tm_1)$ and $J(tm_2)$ are both defined and comparable
   then true **if** $J(tm_1) \ \Theta J(tm_2)$ is true
   else false
   else undef.
3. **If** E is of the form $l_1 \ \Theta \ l_2$ where l_1, l_2 are logical expressions and $\Theta : \wedge, \vee, \Rightarrow, \Leftrightarrow$
   then the semantics of $l_1 \ \Theta \ l_2$ is given by the the truth table for three-valued logic.
4. **If** E is of the form $te_1 \cup te_2$ where te_1, te_2 are table expressions
   then $J(E) = J(te_1) \cup J(te_2)$.
5. **If** E is of the form $te_1 - te_2$ where te_1, te_2 are table expressions
   then $J(E) = J(te_1) \setminus J(te_2)$.
6. **If** E is of the form $te_1 \bowtie te_2$ where te_1, te_2 are table expressions
   then $J(E) = J(t_1) \oplus J(t_2)$.
7. **If** E is of the form $te \Uparrow \vec{a}$ where $a_1, \ldots, a_n \in$ AN and te is a table expression
   then $J(E) = \{x \uparrow \vec{a} \mid x \in J(te)\}$.

8. **If** E is of the form $\sigma(te; rl)$ where te is a table expression and rl is a relational logical expression and $\{a_1, \ldots, a_n\} = A(te)$
then $J(E) \ = \ \{x \in J(te) \mid J_{\vec{a}/x\vec{.}a}(rl) = true\}$.

9. **If** E is of the form $\rho(te; [\vec{b} \setminus \vec{a}])$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in$ AN and te is a table expression
then $J(E) \ = \ J(te) \infty h$
where h is the bijection from $\{b_1, \ldots, b_n\}$ into $\{a_1, \ldots, a_n\}$ that satisfies
$\forall 1 \leq i \leq n \ \bullet \ h(b_i) = a_i$

10. **If** E is of the form $\vec{t} := \vec{te}$ where $t_1, \ldots, t_n \in$ TABN and $te_1, \ldots, te_n$ are table expressions
then $J_j(E) = j \cup \{< t_i, te_i > \mid 1 \leq i \leq n\}$.

F Proof of theorem1

Proof With induction to the structure of schema expressions

1. **If** E is a schema name s $\in$sn **then**

   $J'(\phi(s))$

   =%E1%

   $j(s)$

   =%i(s) = j(s)%

   $i(s)$

   =%C1%

   $I'(s)$

2. **If** E is of the form $s_1 \ \wedge s_2$ where s_1, s_2 are schema expressions **then**

   $J'(\phi(s_1) \ \bowtie \ \phi(s_2))$

   =%E6, $\forall v \in V(s_1) \cap V(s_2) \bullet VT(s_1)(v) = VT(s_2)(v)$ so by Lemma1
   % $\forall v \in A(\phi(s_1)) \cap A(\phi(s_2)) \bullet AV(\phi(s_1))(v) = AV(\phi(s_2))(v)$%

   $J'(\phi(s_1)) \oplus J'(\phi(s_2))\}$

   =%Lemma1, I.H.%

   $I'(s_1) \oplus I'(s_2)$

   =%C19 %

   $I'(s_1 \wedge s_2)$

3. **If** E is of the form $\neg s$ where s is a schema expression **then**

$J'(\bowtie \{d(v : VT(s)(v)) \mid v \in V(s)\} - \phi(s))$

$=$%E5 %

$J'(\bowtie \{d(v : VT(s)(v)) \mid v \in V(s)\}) \setminus J'(\phi(s))$

$=$%E6 %

$\prod(\{< v, J'(AV(d(v : VT(s)(v)))(v)) >\mid v \in V(s)\}) \setminus J'(\phi(s))$

$=$%Lemma1, I.H., calculus %

$\prod(I' \circ VT(s)) \setminus I'(s)$

$=$%C20 %

$I'(\neg s)$

4. **If** E is of the form $s \mid rl$ where s is a schema expression and rl is a relational logical expression and $\{v_1, \ldots, v_n\} = V(s)$ **then**

$J'(\sigma(\phi(s); rl)$

$=$%E8, by Lemma1 $\{v_1, \ldots, v_n\} = A(\phi(s))$ %

$\{x \in J'(\phi(s)) \mid J'_{\vec{v}/x.\vec{v}}(rl) = true\}$

$=$%rl is a relational logical expression , I.H. %

$\{x \in I'(s) \mid I'_{\vec{v}/x.\vec{v}}(rl) = true\}$

$=$%C21 %

$I'(s \mid rl)$

5. **If** E is of the form $s\lceil(\vec{v})$ where $v_1, \ldots, v_n \in$ VN and s is a schema expression **then**

$J'(\phi(s) \Uparrow \vec{v})$

$=$%E7, by Lemma1 $A(\phi(s)) = V(s)$ %

$\{x \uparrow \vec{v} \mid x \in J'(\phi(s))\}$

$=$% I.H., Lemma1 %

$\{x \uparrow \vec{v} \mid x \in I'(s)\}$

$=\%$C22 $\%$

$$I'(s\lceil(\vec{v}))$$

6. **If** E is of the form $s \setminus (\vec{v})$ where $v_1, \dots, v_n \in$ VN and s is a schema expression **then** analogous to 5.
7. **If** E is of the form $s[\vec{w} \setminus \vec{v}]$ where $v_1, \dots, v_n, w_1, \dots, w_n \in$ VN and s is a schema expression **then**

   $$J'(\rho(\phi(s)); [\vec{w} \setminus \vec{v}]))$$

   $=\%$E8, Lemma1 $\%$

   $$J'(\phi(s)) \infty h$$

   $=\%$ I.H. $\%$

   $$I'(s) \infty h$$

   $=\%$ C24 $\%$

   $$I'(s[\vec{w} \setminus \vec{v}])$$

8. **If** E is of the form *pre* s where s is a schema expression **then** analogous to 5.
9. **If** E is of the form *post* s where s is a schema expression **then** analogous to 5.
10. **If** E is of the form $s_1 \otimes s_2$ where s_1, s_2 are schema expressions **then**
    Let $h_1 v = [h_1(\vec{v}) \setminus \vec{v}]$
    Let $h_2 v = [h_2(\vec{z}) \setminus \vec{z}]$
    Let $W = (V(s_1) \setminus SA) \cup (V(s_2) \setminus SB)$

    $$J'(\rho(\phi(s_1); h_1 v) \bowtie \rho(\phi(s_2); h_2 z)) \Uparrow W$$

    $=\%$E8, E6,
    $\%$ $\forall v \in V(s_1) \cap V(s_2) \bullet VT(s_1)(v) = VT(s_2)(v)$ so by Lemma1 and def h_1, h_2
    $\%$ $\forall v \in A(\rho(\phi(s_1); h_1 v)) \cap A(\rho(\phi(s_2); h_2 z))$
    $\%$ $\quad \bullet AV(\rho(\phi(s_1); h_1 v))(v) = AV(\rho(\phi(s_2); h_2 z))(v)$ $\%$

    $$\{x \uparrow W \mid x \in J'(\rho(\phi(s_1); h_1 v)) \oplus J'(\rho(\phi(s_2); h_2 z))\}$$

    $=\%$E9, def AV, def h_1, def h_2, Lemma1 $\%$

    $$\{x \uparrow W \mid x \in (J'(\phi(s_1)) \infty h_1^{-1}) \oplus (J'(\phi(s_2)) \infty h_2^{-1})\}$$

    $=\%$ I.H., calculus $\%$

    $$\{x \uparrow W \mid x \in (I'(s_1) \infty h_1^{-1}) \oplus (I'(s_2) \infty h_2^{-1})\}$$

=%C27 %

$I'(s_1 \otimes s_2)$

11. **If** E is of the form $s_1 \gg s_2$ where s_1, s_2 are schema expressions **then** analogous to 10.

End of Proof

G Proof of theorem2

Proof

1. **If** E is a table name tab $\in$tabn **then**

   $I'(tab))$

   =%C1 %

   $i(tab)$

   =%$i(tab) = j(tab)$ %

   $j(tab)$

   =%E1 %

   $J'(tab)$

   .

2. **If** E is of the form $te_1 \cup te_2$ where te_1, te_2 are table expressions **then**

   $I'(\neg(\neg\psi(te_1) \wedge \neg\psi(te_2))$

   =%C19, $VT(\psi(te_1)) = VT(\psi(te_2)) = VT(\neg\psi(te_1)) = VT(\neg\psi(te_2))$ %

   $\prod(I' \circ VT(\psi(te_1))) \setminus I'(\neg\psi(te_1) \wedge \neg\psi(te_2))$

   =%C19%

   $\prod(I' \circ VT(\psi(te_1))) \setminus (I'(\neg\psi(te_1)) \oplus I'(\neg\psi(te_2)))$

   =%$VT(\neg\psi(te_1)) = VT(\neg\psi(te_2))$ so
   =%$I'(\neg\psi(te_1)) \oplus I'(\neg\psi(te_2)) = I'(\neg\psi(te_1)) \cup I'(\neg\psi(te_2))$ %

   $(\prod(I' \circ VT(\psi(te_1))) \setminus I'(\neg\psi(te_1))) \cup (\prod(I' \circ VT(\psi(te_2))) \setminus I'(\neg\psi(te_2)))$

   =%C19, calculus %

$I'(\psi(te_1)) \cup I'(\psi(te_2))$

$=$%I.H., Lemma2%

$J'(te_1) \cup J'(te_2)$

$=$%E3 %

$J'(te_1 \cup te_2)$

3. **If** E is of the form $te_1 \; - \; te_2$ where te_1,te_2 are table expressions **then**

   $I'(\psi(te_1) \;\wedge\; \neg\psi(te_2))$

   $=$%C19, $AV(te_1) = AV(te_2)$ so by Lemma2 and def VT
   % $VT(\psi(te_1)) = VT(\psi(te_2)) = VT(\neg\psi(te_2))$ %

   $I'(\psi(te_1)) \oplus I'(\neg\psi(te_2))$

   $=$%C20 %

   $I'(\psi(te_1)) \oplus \prod(I' \circ VT(\psi(te_2))) \setminus I'(\psi(te_2))$

   $=$%Lemma2, I.H., calculus %

   $J'(te_1) \oplus \prod(J' \circ AV(te_2)) \setminus I'(te_2)$

   $=$%E5, calculus %

   $J'(te_1 - te_2)$

4. **If** E is of the form $te_1 \;\bowtie\; te_2$ where te_1,te_2 are table expressions **then**

   $J'(\psi(te_1) \;\wedge\; \psi(te_2))$

   $=$%C19, $\forall a \in A(te_1) \cap A(te_2) \bullet AV(te_1)(a) = AV(te_2)(a)\}$so by Lemma2
   % $\forall a \in V(\psi(te_1)) \cap V(\psi(te_2)) \bullet VT(\psi(te_1))(a) = VT(\psi(te_2))(a)$ %

   $I'(\psi(te_1)) \oplus I'(\psi(te_2))$

   $=$%Lemma2, I.H. %

   $J'(te_1) \oplus J'(te_2)$

   $=$%E6 %

   $J'(te_1 \bowtie te_2)$

5. If E is of the form $te \Uparrow \vec{a}$ where $a_1, \ldots, a_n \in$ AN and te is a table expression **then**

$I'(\psi(te)\lceil(\vec{a}))$

$=$%C23 %

$\{x \uparrow \vec{a} \mid x \in I'(\psi(te))\}$

$=$% I.H., Lemma2 %

$\{x \uparrow \vec{a} \mid x \in J'(te)\}$

$=$%E7 %

$J'(te \Uparrow \vec{a})$

6. If E is of the form $\sigma(te; rl)$ where te is a table expression and rl is a relational logical expression and $A(te) = \{a_1, \ldots, a_n\}$ **then**

$I'(\psi(te) \mid rl)$

$=$%C21, by Lemma2 $V(\psi(te)) = \{a_1, \ldots, a_n\}$ %

$\{x \in I'(\psi(te)) \mid I'_{\vec{a}/x.\vec{a}}(rl) = true\}$

$=$% I.H., Lemma2 rl is a relational logical expression %

$\{x \in J'(te) \mid J'_{\vec{a}/x.\vec{a}}(rl) = true\}$

$=$%E8 %

$J'(\sigma(te; rl))$

7. If E is of the form $\rho(te; [\vec{b} \setminus \vec{a}]$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in$ AN and te is a table expression **then**

$I'(\psi(te)[\vec{b} \setminus \vec{a}])$

$=$%C24 %

$I'(\psi(te)) \infty h$

$=$%I.H.%

$J'(te) \infty h$

$=$%E9 %

$J'(\rho(te; [\vec{b} \setminus \vec{a}]))$

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS--circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.

88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.

89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude.	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J.van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.