A formal semantics for Z and the link between Z and the relational algebra
van Diepen, M.J.; van Hee, K.M.

Published: 01/01/1990

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

Link to publication

Citation for published version (APA):

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

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

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

Download date: 22. Oct. 2018
A formal semantics for Z and the link between Z and the relational algebra

by

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

October, 1990
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

This CSNote is a revised version of CSNote 89/17, appeared in the Proceedings of the VDM'90 Symposium, Springer Verlag (1990).
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 powerful 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 ([Hayes87]) and Spivey ([Spivey88]).

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 $\psi$ from Rscripts, i.e. scripts in the relational algebra, to Zscripts and a partial mapping $\phi$ the other way around such that $\psi$ and $\phi$ are semantics preserving, $\phi \circ \psi$ is the identity on Rscripts 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 set theory 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
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 approach 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.

**Definition 1** Formally a Zbase is a 6-tuple $<$tn, gn, sgnF, sn, sgns, 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 null (no arguments). The set of constants is denoted by $cn$. The set of functions is denoted by $fn$.
- $sgns$ 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 set theory 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 $\Pi I(\text{sgn}_S(s))$ where for a setvalued function $F$, $\Pi(F)$ is defined by $\Pi(F) = \{ f | f \text{ is a function over } \text{dom}(F) \text{ and } \forall x \in \text{dom}(f), f(x) \in F(x) \}$. $\Pi$ is called the generalised cartesian product.

For example, a possible Zbase would be

```
<
  \{nat\}
  , \{+, *, 0, 1, \ldots\}
  , \{<+, \text{nat}^3>, <*, \text{nat}^3>, <0, \text{nat}>, \ldots\}
  , \{S\}
  , \{<S, \{<x, \text{nat}>, <y, \text{nat}>\}\}
  , I
>
```

where $I$ is defined such that

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


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 function names 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 U VN U CN)

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, gn, sgnF, sn, sgnS, I > the syntax is given by

Type expressions

\[
type :: = \begin{array}{l}
  \text{tn} \\
  \set P(type) \\
  \set x \cdots \times type \\
  type \times type \\
  \set VN;type
\end{array}
\]

The simplest types are the type set names of tn. More complex types can be build in combination with the type constructors \(P\) (powerset), \(\rightarrow\) (function) and a special tuple type constructor \([x_1 : T_1, \ldots, x_n : T_n]\).

The tuple type constructor provides in our need of schemas to be used as types.

Logical expressions

\[
\begin{align*}
\text{constant} & ::= \text{CN} \cup \text{VN} \\
                & \cup \{ < \text{constant} > \} \\
                & \cup < \text{constant}, \ldots, \text{constant} \geq \} \\
                & \cup < \text{VN};\text{constant} \geq \}
\end{align*}
\]

\[
\begin{align*}
\text{term} & ::= \text{constant} \\
           & \cup \text{VN}\{\text{term}\} \\
           & \cup \text{FN}( < \text{term} > ) \\
           & \cup \{ < \text{VN};\text{type} > | \text{logex} \}
\end{align*}
\]

4
atom ::= term Θ term
   where Θ : ∈, ⊆, =, ≤, ≥, <, >

logex ::= atom
   | logex Θ logex where Θ : ∧, ∨, ⇒, ⇔
   | ¬ logex
   | ∀ VN : type | logex
   | ∃ VN : type | logex

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

Schemas
\[ \text{schema ::= } [ \langle \text{VARS}_{\text{type}} | \text{TN} \rangle ] \upharpoonright \text{logex} \]

Schema-expressions
\[ \text{sexp ::= schema} \]
| sn | (conjunction) | (negation) | (restriction) | (projection) | (hiding) | (renaming) | (precondition) | (postcondition) | (composition) | (piping) |
| sexp \& sexp | sexp | sexp[logex] | sexp[\langle VN \rangle] | sexp[\langle VN\rangle] | pre sexp | post sexp | sexp \& sexp | sexp \& sexp |

Schema definitions
\[ \text{sdef ::= SN ::= sexp} \]

Type definitions
\[ \text{tdef ::= TN ::= type} \]

Function definitions
\[ \text{fdefs ::= Context ::= schema} \]
Zscript

\[
\text{zscript} ::= [\text{tdef}] [\text{fdefs}] [\text{sdef}]
\]

Remark All new function (constant) definitions are encapsulated within one schema. The schema name \text{Context} is reserved for this purpose. The variables declared in the schema \text{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 \text{Context} must have a type of the form \text{t} or \text{to} \rightarrow \text{tn} \rightarrow \ldots \rightarrow \text{tn} where \text{t, to, ... , tn} are typenames and each such typename must be an element of \text{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 foregoing section, a possible Zscript would be

\[
\text{Context} := \{ \text{faculty : nat} \rightarrow \text{nat} \mid \\
\forall n : \text{nat} \mid \\
(n = 0 \lor n = 1) \Rightarrow \text{faculty}(n) = 1 \land \\
(n \geq 2 \Rightarrow \text{faculty}(n) = n \times \text{faculty}(n - 1)) \}
\]

\[
\mathcal{T} := [i? : \text{nat}, j? : \text{nat} \mid j? = \text{faculty}(i?)]
\]

2.2 Semantics

The next step is to give the formal semantics for the Zscript. This implicitly 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 \text{nat} and \(y\) is of type \(\mathcal{P}\text{nat}\) 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. \(J_e(E)\) denotes the semantics of expression \(E\) to be evaluated in environment \(e\). Only in those cases where the semantics are to be defined in
an extended environment this is explicitly defined. \( J_{n/f}(E) \) 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 foregoing sections. In fact, we will only determine the semantics of the schema Context.

Let \( I_1 = ((n = 0 \lor n = 1) \Rightarrow \text{faculty}(n) = 1) \).

Let \( I_2 = (n \geq 2 \Rightarrow \text{faculty}(n) = n \times \text{faculty}(n - 1)) \).

The interpretation function \( I \) of the Zbase assigns the names \( nat, +, 0, 1, \ldots \) their semantic value, which respectively are \( N, X, 0, 1, \ldots \).

\[
J_I(\text{Context}) = \{x \in \Pi(\{< \text{faculty}, J(nat \to nat)>) \mid J_{\text{faculty}/x.\text{faculty}}(\forall[n: nat \mid I_1 \land I_2])\}
\]

\[
= \{x \in \Pi(\{< \text{faculty}, N \to N)>) \mid \forall n \in J(nat) : J_{n/n}(I_1 \land I_2)\}
\]

\[
= \{x \in \Pi(\{< \text{faculty}, N \to N)>) \mid \forall n \in N : ((n = 0 \lor n = 1) \Rightarrow x.\text{faculty}(n) = 1) \land (n \geq 2 \Rightarrow x.\text{faculty}(n) = n \times x.\text{faculty}(n - 1)).
\]

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 \( ngn \) be the set of functions(constants) defined in the Zscript \( Zs \) (\( V(\text{Context}) \)).

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 schemanames in the Zbase (the function \( I \)), each name in \( ntn, ngn \) and \( nsn \) a semantic value.

The new Zbase is defined by \(< tn', gn', sgn'_F, sn', sgn'_S, I' > \) where

- \( tn' = tn \cup ntn \),
• $g' = gn \cup ngn$,

• $sgn'(f) = sgn_F(f), f \in gn$,
  
  sgn $F(f) = VT(Context)(f), f \in ngn$,

• $s' = sn \cup nsn$,

• $sgn'(s) = sgn_S(s), s \in sn$,

  sgn $'s(s) = VT(se), s \in nsn$ and $se$ is the schema expression associated with $s$ in $Zs$ by
  
  $s := se$,

• $I'(n) = J(n), n \in tn \cup ntn \cup gn \cup sn \cup nsn$,

  $I'(n) = J(Context)(n), n \in ngn$.

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.
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 =, ≤, ≥, <, > 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 \( \langle dn, cn, sgn_C, tabn, sgn_T, I \rangle \) where

- \( dn \) is a subset of DN, \( cn \) is a subset of CN and \( tabn \) is a subset of TABN,
- \( sgn_C \subset cn \rightarrow dn \), domain of each constant,
- \( sgn_T \subset tabn \rightarrow (AN \neq dn) \),
- \( I \) is an interpretation function for \( dn, cn \) and \( tabn \). The definition of \( I \) is based on the naive, untyped settheory
  - 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 \( \langle dn, cn, sgn_C, tabn, sgn_T, I \rangle \) the syntax is given by

Relational logical expressions

\begin{align*}
constant & := \text{cn} \\
\text{term} & := constant \\
& | \ AN
\end{align*}
atom ::= term Θ term
   where Θ ::= =, ≤, ≥, <, >

rlogex ::= atom
   | rlogex Θ rlogex where Θ ::= ∧, ∨, →, ↔
   | ¬rlogex

Table expressions

texp ::= tabn
   | texp_∪ texp  (union)
   | texp_− texp  (difference)
   | texp_× texp  (join)
   | texp{< AN >}  (projection)
   | α(texp;rlogex)  (selection)
   | ρ(texp, < AN\AN >)  (renaming)

Table definitions

tabdef ::= TABN := 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, sgn₁, tabn, sgnᵣ, 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', sgn₁', tabn', sgnᵣ', J > where

- tabn' = tabn ∪ ntabn, ntabn ∩ tabn = ∅
- sgnᵣ'(tab) = sgnᵣ(tab), tab ∈ tabn,
  sgnᵣ'(tab) = AV(te), tab ∈ 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 \([x: \text{nat}; y: \text{nat} \mid x = y]\) 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 Zbases.

Let a Zbase \(Zb = < \text{tn}, \text{gn}, \text{sgn}_F, \text{sn}, \text{sgn}_S, I >\) be given.

The function \(F\) With every schemaname in \(\text{sn}\) we associate a tablename. 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 the schema \([x : t \mid \text{true}]\). The set of variable declarations can be extracted from the signature function \(\text{sgn}_S\). Let \(\text{dtn}\) be the set of these tablenames and let \(d\) be the bijection assigning to each variable declaration a tablenamne in \(\text{dtn}\).

The Rbase \(Rb\) corresponding to \(Zb\) is given by \(< \text{dn}, \text{cn}, \text{sgn}_C, \text{tabn}, \text{sgn}_T, J >\) where

- \(\text{dn}\) equals the set of type set names \(\text{tn}\).
• \( cn \) equals the set of constants in \( gn \).

• \( \text{sgnc}(c) = \text{sgnP}(c) \), \( c \in cn \).

• \( \text{tabn} = sn \cup dtn \), \( sn \cap dtn = \emptyset \).

• \( \text{sgnt}(\text{tab}) = \text{sgns}(\text{tab}) \), \( \text{tab} \in sn \),

\[
\text{sgnt}(\text{tab}) = \begin{cases} 
\{ <x, t> \}, \text{tab} \in dtn \text{ and tab is assigned to the declaration } x : t, \text{ i.e. } \\
\text{d}^{-1}(\text{tab}) = x : t. 
\end{cases}
\]

• \( J(t) = I(t) \), \( t \in t_n \),

\[
J(c) = I(c), \text{ } c \in cn, \\
J(\text{tab}) = I(\text{tab}), \text{ } \text{tab} \in sn, \\
J(\text{tab}) = \{ <x, I(t)> \}, \text{tab} \in dtn \text{ and tab is assigned to the declaration } x : t, \text{ i.e. } \\
\text{d}^{-1}(\text{tab}) = x : t. 
\]

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

1. If \( E \) is a schemaname \( s \in sn \)
   then \( \phi(E) = s. \)

2. If \( E \) is of the form \( s_1 \land s_2 \) where \( s_1, s_2 \) are schema expressions
   then \( \phi(E) = \phi(s_1) \land \phi(s_2). \)

3. If \( E \) is of the form \( \neg s \) where \( s \) is a schema expression
   then \( \phi(E) = \neg \left( \{ d(v : V T(s)(v)) \mid v \in V(s) \} \setminus \phi(s) \right). \)

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 \( S[v] \) where \( v_1, \ldots, v_n \in VN \) and \( s \) is a schema expression
   then \( \phi(E) = \phi(s) \uparrow (V(s) \setminus \{ v \}). \)

6. If \( E \) is of the form \( s \uparrow v \) where \( v_1, \ldots, v_n \in VN \) and \( s \) is a schema expression
   then \( \phi(E) = \phi(s) \uparrow (V(s) \setminus \{ v \}). \)

7. If \( E \) is of the form \( s[\bar{w}] \) where \( w_1, \ldots, w_n \in VN \) and \( s \) is a schema expression
   then \( \phi(E) = \rho(\phi(s); [\bar{w} \setminus \{ w \}]). \)

8. If \( E \) is of the form \text{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 \text{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 \( \psi(E) = (\rho(\phi(s_1); [h_1(v) \setminus \{ v \}]) \preceq \rho(\phi(s_2); [h_2(z) \setminus \{ z \}]) \uparrow (V(s_1) - \text{SA}) \cup (V(s_2) \setminus \text{SB}). \)

where

- \( \text{SA} = \{ v \in V(s_1) \cap DH \mid \exists w \in V(s_2) \cap UD \} \) \text{ basename}(v) = w \}.
- \( \text{SB} = \{ v \in V(s_2) \cap UD \mid \exists w \in V(s_1) \cap DH \} \) \text{ 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 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 attribute counterpart have the same type (domain) and due to the semantic functions $I'$ and $J'$ the type and corresponding domain have the same semantics.

Lemma 1 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)

Theorem 1 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 $\psi$ 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 $< tn, gu, sgn_F, sn, sgn_S, I >$ where

- $tn = dn$.
- $gu = cn$.
- $sgn_F(c) = sgn_C(c), c \in cn$.
- $sn = tabn$.
- $sgn_S(s) = sgn_T(s), s \in tabn$.
- $I(d) = J(d), d \in dn$,
  $I(c) = J(c), c \in cn$,
  $i(s) = j(s), s \in tabn$. 

13
The function $\psi$. The function $\psi$ is defined by

1. If $E$ is a tablename $\text{tab} \in \text{tabn}$
   then $\psi(E) = \text{tab}$.

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

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

4. If $E$ is of the form $te_1 \setminus te_2$ where $te_1, te_2$ are table expressions
   then $\psi(E) = \psi(te_1) \land \psi(te_2)$

5. If $E$ is of the form $te \upharpoonright a$ where $a_1, \ldots, a_n \in \text{AN}$ and $te$ is a table expression
   then $\psi(E) = \psi(te)[\{a\}]$

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 $\omega(te; \{b \mid a\})$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in \text{AN}$ and $te$ is a table expression
   then $\psi(E) = \psi(te)[\{b \mid a\}]$

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 $\psi$ 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.

Lemma 2 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)

Theorem 2 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 $\phi$ from Zscripts to Rscripts such that for every Zbase $Zb$ and every 'correct' Zscript $Zs$ of that Zbase $Zb$, the Rscript $\phi(Zs)$ is a Rscript of the Rbase $F(Zb)$. Moreover the function $\phi$ is such that for every Zbase $Zb$ and every 'correct' Zscript $Zs$ of $Zb$, the semantics of the Zscript $\phi(Zs)$ 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 $\psi$ 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 $\phi(\psi(Rs)) = Rs$. 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 $Zb$, $\psi(\phi(Zs)) = Zs$. 

14
5 Conclusions

A simple formal semantics of Z, based on naive, untyped set theory, 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

[Bjørner82] D. Bjørner and C.B. Jones

[Hayes87] I. Hayes

[Schmidt88] D.A. Schmidt

[Ullman82] J.D. Ullman

[Spivey88] J.M. Spivey
A Mathematical notation

• \( \uparrow \), restriction
  If \( f \) is a function and \( A \) a set then \( f \uparrow A = \{ <x,y> | x \in f \mid x \in A \} \)

• \( \circ \), function composition
  If \( f \) and \( g \) are functions then \( f \circ g = \{ <x,g(f(x))> | x \in \text{dom}(f) \land f(x) \in \text{dom}(g) \} \)

• \( \oplus \)
  For function collections \( F \) and \( G \) with resp. domains \( F \) and \( G \)
  \( F \oplus G = \{ h \mid \text{dom}(h) = F \cup G \land h \uparrow F \in F \land h \uparrow G \in G \} \)
  Note \( F = G \Rightarrow F \oplus G = F \cup G \)

• \( \Pi \), generalised product
  for a setvalued function \( F \), \( \Pi(F) \) is defined by
  \( \Pi(F) = \{ f \mid f \text{ is a function over } \text{dom}(F) \land \forall x \in \text{dom}(f) f(x) \in F(x) \} \).

• The operator \( \circ \) 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 \( T \circ h = \{ t \circ h \mid t \in T \} \).

• \( \tilde{\sigma} \) is shorthand for \( \{ v_1, \ldots, v_n \} \) for arbitrary \( n \).

• \( \tilde{(\sigma)} \) is shorthand for \( (v_1, \ldots, v_n) \) for arbitrary \( n \).

• \( [w \backslash \tilde{\sigma}] \) is shorthand for \( [w_1 \backslash v_1, \ldots, w_n \backslash v_n] \) for arbitrary \( n \).

• \( \tilde{\sigma} : \tilde{I} \) is shorthand for \( v_1 : t_1, \ldots, v_n : t_n \) for arbitrary \( n \).

• \( \tilde{\sigma} := \tilde{I} \) is shorthand for \( v_1 := t_1, \ldots, v_n := t_n \) for arbitrary \( n \).

• setcomplement, for sets \( V \) and \( W \), \( V^c = W \setminus V \).

• \( \not\mathcal{A} \), 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 $VT : \text{sexp} \not\in (\text{FN} \cup \text{CN} \cup \text{VN} \not\in \text{TN}^* )$ 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 $V : \text{sexp} \not\in \mathcal{P}(\text{FN} \cup \text{CN} \cup \text{VN})$ 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:

- $UD = \text{set of all undashed variables in } \text{FN} \cup \text{CN} \cup \text{VN} \text{ not ending in ? of}! \text{ (state-before components).}$
- $DH = \text{set of all dashed variables in } \text{FN} \cup \text{CN} \cup \text{VN} \text{.}$ (state-after components).
- $IP = \text{set of all variables in } \text{FN} \cup \text{CN} \cup \text{VN} \text{ ending in ?} \text{ (inputs).}$
- $OP = \text{set of all variables in } \text{FN} \cup \text{CN} \cup \text{VN} \text{ ending in !} \text{ (outputs).}$
- The function $\text{basename}$ assigns to every variable in $\text{FN} \cup \text{CN} \cup \text{VN}$ 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 $[\overline{v} : \overline{t} ]\text{ l}$ where $v_1, \ldots, v_n \in \text{VARS}$, $t_1, \ldots, t_n$ 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 $s(\overline{v})$ where $v_1, \ldots, v_n \in \text{VN}$ and $s$ is a schema expression then $VT(E) = VT(s) \uparrow \overline{v}$.
7. If $E$ is of the form $s \setminus (\overline{v})$ where $v_1, \ldots, v_n \in \text{VN}$ and $s$ is a schema expression then $VT(E) = VT(s) \downarrow \overline{v}$.
8. If $E$ is of the form $s(\overline{w} \setminus \overline{v})$ where $v_1, \ldots, v_n, w_1, \ldots, w_n \in \text{VN}$ and $s$ is a schema expression and $\{ v_1, \ldots, v_n \} = \text{dom}(VT(s))$ 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 satisfies $\forall 1 \leq i \leq n \bullet h(w_i) = v_i$
9. If \( E \) is of the form \( \text{pre } s \) where \( s \) is a schema expression
   then \( \text{VT}(E) = \text{VT}(s) \uparrow (DH \cup OP)^c \).

10. If \( E \) is of the form \( \text{post } s \) where \( s \) is a schema expression
    then \( \text{VT}(E) = \text{VT}(s) \downarrow (UD \cup IP)^c \).

11. If \( E \) is of the form \( s_1 \gg s_2 \) where \( s_1, s_2 \) are schema expressions
    then \( \text{VT}(E) = \) \[
    (\text{VT}(s_1) \uparrow \{ v \in \text{dom}(\text{VT}(s_1)) \cap OP \mid (\exists w \in \text{dom}(\text{VT}(s_2)) \cap IP \cdot \text{basename}(v) = \text{basename}(w))\}) \cup \\
    (\text{VT}(s_1) \downarrow \{ v \in \text{dom}(\text{VT}(s_2)) \cap IP \mid (\exists w \in \text{dom}(\text{VT}(s_1)) \cap OP \cdot \text{basename}(v) = \text{basename}(w))\})
    \]

The function \( V \) is defined by: \( V(s) = \text{dom}(\text{VT}(s)) \).

Context-conditions

1. Each constant name in a constant expression must be a constant name in \( cn \) or there must be a variable declaration in \( \text{Context} \) where the same name is declared as a constant.

2. Each function name in a term expression must be a function name in \( \text{fn} \) or there must be a variable declaration in \( \text{Context} \) where the same name is declared as a function (not a constant).

3. Each variable declared in the schema \( \text{Context} \) must be in \( \text{VN} \cup \text{CN} \) and must have a type of the form \( t \) or \( to + ... + tn \) where \( t, to, ... , tn \in \text{TN} \) and each such type name must be an element of \( \text{tn} \) 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 \( \text{Context} \)) must be in \( \text{VN} \) and must have its type in \( \text{TN} \) and each such typename must be an element of \( \text{tn} \) 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 \( f(t_1, ..., t_n) \) where \( f \) is a function and \( t_1, ..., t_n \) are terms is allowed if \( f \) has arity \( n \) and for all \( i : 1 \leq i \leq n \) the type of \( t_i \) is a subtype of \( \text{sgn}_F(i) \).

6. In a schema of the form \( [v : i \mid l] \) where \( v_1, ..., v_n \in \text{VN} \), \( t_1, ..., t_n \) are type expressions and \( l \) is a logical expression the only free variables in \( l \) may be the variables \( v_1, ..., v_n \), the elements of \( gn \) and the variables declared in \( \text{Context} \).

7. A schema expression of the form \( s_1 \wedge s_2 \) or \( s_1 \oplus s_2 \) where \( s_1, s_2 \) are schema expressions must satisfy
   \[
   \forall v \in V(s_1) \cap V(s_2) \cdot \text{VT}(s_1)(v) = \text{VT}(s_2)(v) .
   \]

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 \( gn \) and the variables declared in \( \text{Context} \).

9. A schema expression of the form \( s[w \mid \bar{v}] \) where \( s \) is a schema expression and \( v_1, ..., v_n, w_1, ..., w_n \in \text{VN} \) must satisfy the following conditions
   a) \( \{v_1, ..., v_n\} = V(s) \)
   b) \( \forall 1 \leq i < j \leq n \cdot w_i \neq w_j \)
10. In case of the operators \textit{pre}, \textit{post}, \& and \textit{⇒} the schema expressions must denote operations.

11. A schema expression of the form \( s_1 \Rightarrow s_2 \) where \( s_1, s_2 \) are schema expressions must satisfy
   a) \( \forall v \in V(s_1) \cap V(s_2) \cdot VT(s_1)(v) = VT(s_2)(v) \).

   
b) \( \forall v \in V(s_1) \cap \text{OP} \cdot \\
   (\exists w \in V(s_2) \cap \text{IP} \cdot \text{basename}(v) = \text{basename}(w) \Rightarrow VT(s_1)(v) = VT(s_2)(w)) \)

   c) \( \forall v \in V(s_2) \cap \text{IP} \cdot \\
   (\exists w \in V(s_1) \cap \text{OP} \cdot \text{basename}(v) = \text{basename}(w) \Rightarrow VT(s_1)(w) = VT(s_2)(v)) \)

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 set theory. 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 \( P(t) \) where \( t \) is a type expression
   then \( J(E) = \text{power}(J(t)) \).
3. If \( E \) is of the form \( (t_1, \ldots, t_n) \) where \( t_1, \ldots, t_n \) are type expressions
   then \( J(E) = \text{product}(J(t_1), \ldots, J(t_n)) \).
4. If \( E \) is of the form \( t_1 \to t_0 \) where \( t_1, t_0 \) are type expressions
   then \( J(E) = J(t_1) \to J(t_0) \).
5. If \( E \) is of the form \([\vec{v} : \ell]\) where \( v_1, \ldots, v_n \in \text{VN} \) 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 constant name \( c \)
   then \( J(E) = J(\text{Context})(c) \).
7. If \( E \) is of the form \( c \) where \( C_1, \ldots, C_n \in \text{CN} \)
   then \( J(E) = J(c) \).
8. If \( E \) is of the form \( <\vec{c}> \) where \( c_1, \ldots, c_n \in \text{CN} \)
   then \( J(E) = <J(c)> \).
9. If \( E \) is of the form \( \langle\vec{v}:\vec{c}\rangle \) where \( v_1, \ldots, v_n \in \text{VN} \) and \( c_1, \ldots, c_n \in \text{CN} \)
   then \( J(E) = \langle\vec{v}:J(\vec{c})\rangle \).
10. If \( E \) is of the form \( f(\vec{t}) \) where \( f \) is a function name and \( tm_1, \ldots, tm_n \) are terms
    then if \( f \) is a function in \( \text{gn} \)
        then \( J(E) = J(f) \cdot (J(\vec{t})) \)
        else ( \( f \) is a function in \( \text{Context} \) ) \( J(E) = (J(\text{Context})(f)) \cdot (J(\vec{t})) \).
11. If \( E \) is of the form \( v(\vec{t}) \) where \( v \in \text{VN} \) and \( tm_1, \ldots, tm_n \) are terms
    then \( J(E) = J(v) \cdot (J(\vec{t})) \).
12. If \( E \) is of the form \( \{\vec{v} : \ell \} \) where \( v_1, \ldots, v_n \in \text{VN} \), \( t_1, \ldots, t_n \) are type expressions en
    \( l \) is a logical expression and \( v_1, \ldots, v_n \) are the only free variables in \( l \)
    then \( J(E) = \{\vec{x} : J(t) | J_{\ell/l}(l) = \text{true}\} \).
13. If \( E \) is of the form \( tm_1 \Theta tm_2 \) where \( tm_1, tm_2 \) are terms en \( \Theta : \in, \subseteq, \le, >, =, >, < \)
    then if \( J(tm_1) \) and \( J(tm_2) \) are both defined and comparable
        then \( \text{true if } J(tm_1) \Theta J(tm_2) \) is true
        else \( \text{false} \)
        else \( \text{undef} \).
14. If \( E \) is of the form \( l_1 \Theta l_2 \) where \( l_1, l_2 \) are logical expressions and \( \Theta : \land, \lor, \Rightarrow, \Leftrightarrow \)
    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 $\forall [v : t \mid I]$ where $v \in VN$, $t$ is a type expression and $l$ is a logical expression
   then if for all $x \in J(t)$ : $J_{/x}(l) = true$ then $J(E) = true$.
   if for one $x \in J(t)$ : $J_{/x}(l) = false$ then $J(E) = false$.
   else $J(E) = undef$.

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

18. If $E$ is of the form $[v : t \mid I]$ where $v$ is a varname, $t$ is a type expression and $I$ is a logical expression
   then $J(E) = \{x \in J(s) \mid J_{/v}(l) = true\}$.

19. If $E$ is of the form $s \cdot A$ where $s$ is a schema expression
   $J(E) = J(s) \cup J(A)$.

20. If $E$ is of the form $s \cdot B$ where $s$ is a schema expression
   then $E$ is the same as $J(s)$.

21. If $E$ is of the form $s \cdot s$ where $s$ is a schema expression and $l$ is a logical expression
   then $J(E) = \{x \in J(s) \mid J_{/v}(l) = true\}$.

22. If $E$ is of the form $s \cdot (v)$ where $v$ is a varname and $s$ is a schema expression
   then $J(E) = \{x \in J(s) \mid v \in J(s)\}$.

23. If $E$ is of the form $s \cdot (v)$ where $v$ is a varname and $s$ is a schema expression
   then $J(E) = \{x \in J(s) \mid v \in J(s)\}$.

24. If $E$ is of the form $s \cdot s$ where $s$ is a schema expression and $l$ is a logical expression
   then $J(E) = \{x \in J(s) \mid J_{/v}(l) = true\}$.

25. If $E$ is of the form $s \cdot s$ where $s$ is a schema expression
   then $J(E) = \{x \in J(s) \mid v \in J(s)\}$.

26. If $E$ is of the form $s \cdot s$ where $s$ is a schema expression
   then $J(E) = \{x \in J(s) \mid v \in J(s)\}$.
• \( SA = \{ v \in V(s_1) \cap DH \mid \exists w \in V(s_1) \cap UD \cdot \text{basename}(v) = w \} \).
• \( SB = \{ v \in V(s_2) \cap UD \mid \exists w \in V(s_2) \cap DH \cdot \text{basename}(w) = v \} \).
• \( h_1 \) 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)) = \{ \}, |B| = |SA| = |SB|
(-) \( \forall v \in SA \cdot h_1(v) \in B \)
(-) \( \forall v \in SB \cdot h_2(v) = h_1(w) \) where \( w \in SA \) and \( \text{basename}(w) = v \)
(-) \( \forall v \in V(s_1) \setminus SA \cdot h_1(v) = v \)
(-) \( \forall v \in V(s_2) \setminus SB \cdot h_2(v) = v \)

28. If \( E \) is of the form \( s_1 \gg s_2 \) where \( s_1, s_2 \) are schema expressions
then analogous 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 \( \tilde{s} := \tilde{s}e \) where \( s_1, \ldots, s_n \in SN \) and \( s_1, \ldots, s_n \) are schema expressions
then \( J(E) = \{< s_i; J(se_i)> | 1 \leq i \leq n \} \).

30. If \( E \) is of the form \( \tilde{t} := \tilde{t}e \) where \( t_1, \ldots, t_n \in TN \) and \( t_1, \ldots, t_n \) are type expressions
then \( J(E) = \{< t_i; J(te_i)> | 1 \leq i \leq n \} \).

31. If \( E \) is of the form \( \text{Context} := se \) where \( se \) is a schema
then \( J(E) = \{< \text{Context}, J(se)> \} \).

32. If \( E \) is of the form \( \tilde{s} := \tilde{s}e, \text{Context} := se, \tilde{t} := \tilde{t}e \) where
\( s_1, \ldots, s_n \in SN, s_1, \ldots, s_n \) are schema expressions
\( t_1, \ldots, t_n \in TN \) and \( t_1, \ldots, t_n \) are type expressions
and \( se \) is a schema
then \( J(E) = J \cup J(\tilde{t} := \tilde{t}e) \)
  \( \cup J_{j_1,j_2}(\text{Context} := se) \)
  \( \cup J_{j_1,j_2}(\text{Context} := j_1,j_2(se) \tilde{s} := \tilde{s}e) \)
D  Context conditions for a Rscript

We introduce the functions $A$ and $AV$.

The function $AV : texp \to (AN \to \mathbb{D})$ 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 $t \in \text{tabn}$
   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 $te_1 \land te_2$ 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 $te \uparrow \bar{a}$ where $a_1, \ldots, a_n \in \text{AN}$ and $te$ is a table expression
   then $AV(E) = AV(te) \uparrow \bar{a}$.

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 $\rho(te; [\bar{b} \backslash \bar{a}])$ where $a_1, \ldots, a_n, b_1, \ldots, b_n \in \text{AN}$ 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
   $\forall 1 \leq i \leq n \cdot h(b_i) = a_i$.

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

$A(te) = \text{dom}(AV(te))$

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 $\rho(te; [\bar{b} \backslash \bar{a}])$ where $te$ is a table expression and $a_1, \ldots, a_n, b_1, \ldots, b_n \in \text{AN}$ must satisfy
   a) $\{a_1, \ldots, a_n\} = A(te)$
   b) $\forall 1 \leq i < j \leq n \cdot b_i \neq b_j$

4. The names assigned to table expressions must be disjoint to $\text{tabn}$.
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 set theory. 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 undefined.

3. If $E$ is of the form $l_1 \Theta l_2$ where $l_1, l_2$ are logical expressions and $\Theta : \land, \lor, \Rightarrow, \Leftrightarrow$
   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 $J(E) = J(te_1) \bowtie J(te_2)$.

7. If $E$ is of the form $te \upharpoonright \bar{a}$ where $a_1, \ldots, a_n \in AN$ and $te$ is a table expression
   then $J(E) = \{ x \upharpoonright \bar{a} \mid x \in J(te) \}$.

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 $J(E) = \{ z \in J(te) \mid J_{4/2,3}(rl) = true \}$.

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

10. If $E$ is of the form $t_i := \bar{te}$ where $t_1, \ldots, t_n \in TABN$ and $te_1, \ldots, te_n$ are table expressions
    then $J_I(E) = I \cup \{ < t_i, te_i > \mid 1 \leq i \leq n \}$.
F  Proof of theorem1

Proof With induction to the structure of schema expressions

1. If E is a schema name s ∈ sn then

   \[ J'(\phi(s)) = %E1 \%

   J(s) = %I(s) = J(s) \%

   I(s) = %C1 \%

   I'(s)

2. If E is of the form \[ s_1 \land s_2 \] where \[ s_1, s_2 \] are schema expressions then

   \[ J'(\phi(s_1) \land \phi(s_2)) = %E6, \forall v \in V(s_1) \land V(s_2) \land VT(s_1)(v) = VT(s_2)(v) \text{ so by Lemma} \%

   \[ J'(\phi(s_1)) \land J'(\phi(s_2)) \%

   = %Lemmal, I.H. \%

   I'(s_1) \land I'(s_2) = %C19 \%

   I'(s_1 \land s_2) \]

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

   \[ J'(\neg \{d(v : VT(s)(v)) | v \in V(s)\} \land \neg \phi(s)) = %E5 \%

   J'(\neg \{d(v : VT(s)(v)) | v \in V(s)\}) \land J'(\phi(s)) = %E6 \%

   \[ \Pi(\{< v, J'(AV(d(v : VT(s)(v)))(v)) > | v \in V(s)\}) \land J'(\phi(s)) \]
\begin{enumerate}
\item 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
\[ J'(\phi(s); rl) \]
\[ = \% E8, \text{ by Lemma} 1 \{v_1, \ldots, v_n\} = A(\phi(s)) \%
\[ \{x \in J'(\phi(s)) \mid J'_{\phi(x)}(rl) = true\} \]
\[ = \% rl \text{ is a relational logical expression}, \text{ I.H.} \%
\[ \{x \in I'(s) \mid I'_{\phi(x)}(rl) = true\} \]
\[ = \% C21 \%
\[ I'(s \mid rl) \]
\item If \( E \) is of the form \( s[(\bar{v}) \) where \( v_1, \ldots, v_n \in VN \) and \( s \) is a schema expression then
\[ J'((\phi(s) \triangleright \bar{v})) \]
\[ = \% E7, \text{ by Lemma} 1 A(\phi(s)) = V(s) \%
\[ \{x \triangleright \bar{v} \mid x \in J'(\phi(s))\} \]
\[ = \% \text{ I.H., Lemma} 1 \%
\[ \{x \triangleright \bar{v} \mid x \in I'(s)\} \]
\[ = \% C22 \%
\[ I'(s[(\bar{v})]) \]
\item If \( E \) is of the form \( s \setminus (\bar{v}) \) where \( v_1, \ldots, v_n \in VN \) and \( s \) is a schema expression then
\item analogous to 5.
\item If \( E \) is of the form \( s[\bar{w} \setminus \bar{v}] \) where \( v_1, \ldots, v_n, w_1, \ldots, w_n \in VN \) and \( s \) is a schema expression then
\end{enumerate}
\[ J'(\rho(\phi(s)); [\bar{w} \setminus \bar{v}]) \]

= \%E8, Lemma 1 %

\[ J'(\phi(s)) \odot h \]

= \% I.H. %

\[ \phi'(s) \odot h \]

= \% C24 %

\[ \phi'(s[\bar{w} \setminus \bar{v}]) \]

8. If \( E \) is of the form \( \text{pre} \ s \) where \( s \) is a schema expression then analogous to 5.

9. If \( E \) is of the form \( \text{post} \ s \) where \( s \) is a schema expression then analogous to 5.

10. If \( E \) is of the form \( s_1 \odot s_2 \) where \( s_1, s_2 \) are schema expressions then analogous to 10.

Let \( h_1 v = [h_1(\bar{v}) \setminus \bar{v}] \)

Let \( h_2 v = [h_2(\bar{v}) \setminus \bar{v}] \)

Let \( W = (V(s_1) \setminus SA) \cup (V(s_2) \setminus SB) \)

\[ J'(\rho(\phi(s_1)); h_1 v) \not= \rho(\phi(s_2); h_2 z)) \uparrow W \]

= \%E8, E6,

\% \( \forall v \in V(s_1) \cap V(s_2) \bullet VT(s_1)(v) = VT(s_2)(v) \) so by Lemma 1 and def \( h_1, h_2 \)

\% \( \forall v \in A(\rho(\phi(s_1); h_1 v)) \cap A(\rho(\phi(s_2); h_2 z)) \)

\% \( \bullet AV(\rho(\phi(s_1); h_1 v))(v) = AV(\rho(\phi(s_2); h_2 z))(v) \) %

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

= \%E9, def AV, def \( h_1, h_2 \), Lemma 1 %

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

= \% I.H., calculus %

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

= \% C27 %

\[ \phi'(s_1 \odot s_2) \]

11. If \( E \) is of the form \( s_1 \Rightarrow s_2 \) where \( s_1, s_2 \) are schema expressions then analogous to 10.

\textbf{End of Proof}
G Proof of theorem2

Proof

1. If $E$ is a table name $\text{tab} \in \text{tabn}$ then

   $I'(\text{tab})$
   
   $=\%C1$
   
   $I(\text{tab})$
   
   $=\%I(\text{tab}) = J(\text{tab})$
   
   $J(\text{tab})$
   
   $=\%E1$
   
   $J'(\text{tab})$

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

   $I'(-\neg\psi(te_1) \land \neg\psi(te_2))$
   
   $=\%C19, VT(\psi(te_1)) = VT(\psi(te_2)) = VT(-\neg\psi(te_1)) = VT(-\neg\psi(te_2))$
   
   $\prod(I' \circ VT(\psi(te_1))) \setminus I'(-\psi(te_1) \land \neg\psi(te_2))$
   
   $=\%C19$
   
   $\prod(I' \circ VT(\psi(te_1))) \setminus (I'(-\psi(te_1)) \oplus I'(-\psi(te_2)))$
   
   $=\%VT(-\neg\psi(te_1)) = VT(-\neg\psi(te_2))$ so
   
   $=\%I'(-\neg\psi(te_1)) \oplus I'(-\psi(te_2)) = I'(-\psi(te_1)) \cup I'(-\psi(te_2))$
   
   $((\prod(I' \circ VT(\psi(te_1))) \setminus I'(-\psi(te_1))) \cup (\prod(I' \circ VT(\psi(te_2))) \setminus I'(-\psi(te_2))))$
   
   $=\%C19$, calculus
   
   $I'(\psi(te_1)) \cup I'(\psi(te_2))$
   
   $=\%I.H., \text{Lemma}2$
   
   $J'(te_1) \cup J'(te_2)$
   
   $=\%E3$

   $J'(te_1 \cup te_2)$
3. If $E$ is of the form $te_1 - te_2$ where $te_1, te_2$ are table expressions then

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

$$= \% C19, AV(te_1) = AV(te_2) \text{ so by Lemma2 and def } VT$$

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

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

$$= \% C20 \%$$

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

$$= \% \text{Lemma2, I.H., calculus} \%$$

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

$$= \% E5, \text{ calculus} \%$$

$$J'(te_1 - te_2)$$

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

$$J'(\psi(te_1) \land \psi(te_2))$$

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

$$% \forall a \in V(\psi(te_1)) \cap V(\psi(te_2)) \bullet VT(\psi(te_1))(a) = VT(\psi(te_2))(a) \%$$

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

$$= \% \text{Lemma2, I.H.} \%$$

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

$$= \% E6 \%$$

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

5. If $E$ is of the form $te \uparrow a$ where $a_1, \ldots, a_n \in AN$ and $te$ is a table expression then

$$I'(\psi(te)[(\bar{a})])$$

$$= \% C23 \%$$

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

$$= \% \text{I.H., Lemma2} \%$$

29
{x \uparrow \bar{a} \mid x \in J'(te)}

=\%E7\%

J'(te \uparrow \bar{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 \( V(\psi(te)) = \{a_1, \ldots, a_n\} \) %

\[ \{x \in I'(\psi(te)) \mid I_{\bar{a}/x.a}(rl) = true\} \]

=\% I.H., Lemma2 \( rl \) is a relational logical expression %

\[ \{x \in J'(te) \mid J_{\bar{a}/x.a}(rl) = true\} \]

=\%E8 %

\[ J'(\sigma(te; rl)) \]

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

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

=\%C24 %

\[ I'(\psi(te))\circ h \]

=\% I.H. %

\[ J'(te)\circ h \]

=\%E9 %

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

End of Proof
In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits.</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes.</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films.</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff H.M.L.J. Schols</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.</td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems.</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing K.M. van Hee M. Voorhoeve</td>
<td>ELISA, A language for formal specification of information systems.</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures.</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben J. Paredaens K.M. van Hee</td>
<td>The partition of an information system in several systems.</td>
</tr>
<tr>
<td>86/05</td>
<td>J.J.G. Dietz K.M. van Hee</td>
<td>A framework for the conceptual modeling of discrete dynamic systems.</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoef</td>
<td>Nondeterminism and divergence created by concealment in CSP.</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth L. Shira</td>
<td>On proving communication closedness of distributed layers.</td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing R. Gerth W.P. de Roever</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language.</td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing.</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responder's commentary (IFIP86).</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher R. Gerth</td>
<td>A timed failures model for extended communicating processes.</td>
</tr>
<tr>
<td>86/13</td>
<td>R. Gerth W.P. de Roever</td>
<td>Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica</td>
</tr>
<tr>
<td>Year</td>
<td>Number</td>
<td>Authors</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>86/14</td>
<td>R. Koymans</td>
<td>Specifying passing systems requires extending temporal logic.</td>
</tr>
<tr>
<td>87/01</td>
<td>R. Gerth</td>
<td>On the existence of sound and complete axiomatizations of the monitor concept.</td>
</tr>
<tr>
<td>87/02</td>
<td>Simon J. Klaver, Chris F.M. Verbeme</td>
<td>Federatieve Databases.</td>
</tr>
<tr>
<td>87/03</td>
<td>G.J. Houben, J. Paredaens</td>
<td>A formal approach to distributed information systems.</td>
</tr>
<tr>
<td>87/04</td>
<td>T. Verhoeff</td>
<td>Delay-insensitive codes - An overview.</td>
</tr>
<tr>
<td>87/05</td>
<td>R. Kuiper</td>
<td>Enforcing non-determinism via linear time temporal logic specification.</td>
</tr>
<tr>
<td>87/06</td>
<td>R. Koymans</td>
<td>Temporele logica specificatie van message passing en real-time systemen (in Dutch).</td>
</tr>
<tr>
<td>87/07</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems with real-time temporal logic.</td>
</tr>
<tr>
<td>87/08</td>
<td>H.M.J.L. Schols</td>
<td>The maximum number of states after projection.</td>
</tr>
<tr>
<td>87/10</td>
<td>T. Verhoeff</td>
<td>Three families of maximally nondeterministic automata.</td>
</tr>
<tr>
<td>87/11</td>
<td>P. Lemmens</td>
<td>Eldorado ins and outs. Specifications of a database management toolkit according to the functional model.</td>
</tr>
<tr>
<td>87/12</td>
<td>K.M. van Hee and A. Lapinski</td>
<td>OR and AI approaches to decision support systems.</td>
</tr>
<tr>
<td>87/13</td>
<td>J.C.S.P. van der Woude</td>
<td>Playing with patterns - searching for strings.</td>
</tr>
<tr>
<td>87/14</td>
<td>J. Hooman</td>
<td>A compositional proof system for an occam-like real-time language.</td>
</tr>
<tr>
<td>87/16</td>
<td>H.M.M. ten Eikelder, J.C.F. Wilmont</td>
<td>Normal forms for a class of formulas.</td>
</tr>
<tr>
<td>87/17</td>
<td>K.M. van Hee, G.-J. Houben, J.L.G. Dietz</td>
<td>Modelling of discrete dynamic systems framework and examples.</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>87/18</td>
<td>C.W.A.M. van Overveld</td>
<td>An integer algorithm for rendering curved surfaces.</td>
</tr>
<tr>
<td>87/19</td>
<td>A.J. Seebregts</td>
<td>Optimalisering van file allocatie in gedistribueerde database systemen.</td>
</tr>
<tr>
<td>87/20</td>
<td>G.J. Houben, J. Paredaens</td>
<td>The ( R^2 )-Algebra: An extension of an algebra for nested relations.</td>
</tr>
<tr>
<td>87/21</td>
<td>R. Gerth, M. Codish, Y. Lichtenstein, E. Shapiro</td>
<td>Fully abstract denotational semantics for concurrent PROLOG.</td>
</tr>
<tr>
<td>88/01</td>
<td>T. Verhoeff</td>
<td>A Parallel Program That Generates the Möbius Sequence.</td>
</tr>
<tr>
<td>88/03</td>
<td>T. Verhoeff</td>
<td>Settling a Question about Pythagorean Triples.</td>
</tr>
<tr>
<td>88/04</td>
<td>G.J. Houben, J. Paredaens, D. Tahon</td>
<td>The Nested Relational Algebra: A Tool to Handle Structured Information.</td>
</tr>
<tr>
<td>88/06</td>
<td>H.M.J.L. Schols</td>
<td>Notes on Delay-Insensitive Communication.</td>
</tr>
<tr>
<td>88/09</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>A Tutorial for Data Modelling.</td>
</tr>
<tr>
<td>88/10</td>
<td>J.C. Ebergen</td>
<td>A Formal Approach to Designing Delay Insensitive Circuits.</td>
</tr>
<tr>
<td>88/12</td>
<td>A.E. Eiben</td>
<td>Abstract theory of planning.</td>
</tr>
<tr>
<td>88/13</td>
<td>A. Bijlsma</td>
<td>A unified approach to sequences, bags, and trees.</td>
</tr>
<tr>
<td>88/14</td>
<td>H.M.M. ten Eikelder, R.H. Mak</td>
<td>Language theory of a lambda-calculus with recursive types.</td>
</tr>
</tbody>
</table>
An introduction to the category theoretic solution of recursive domain equations.

Bottom-up tree acceptors.

Executable specifications for discrete event systems.

Discrete event systems: concepts and basic results.

Fasering en documentatie in software engineering.

EXSPECT, the functional part.

Reconstruction of a 3-D surface from its normal vectors.

A systolic design for dynamic programming.

Some category theoretical properties related to a model for a polymorphic lambda-calculus.

Compositionality and modularity in process specification and design: A trace-state based approach.

Networks of Communicating Processes and their (De-)Composition.

Characterizations of Delay-Insensitive Communication Protocols.

A systematic design of a parallel program for Dirichlet convolution.

A general theory of genetic algorithms.

Discrete event systems: Dynamic versus static topology.

A new efficient implementation of CSP with output guards.

Algebraic specification and implementation of infinite processes.

A concise formal framework for data modeling.
<table>
<thead>
<tr>
<th>No.</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>89/14</td>
<td>H.C. Haesen</td>
<td>ELDA, data manipulatie taal.</td>
</tr>
<tr>
<td>89/15</td>
<td>J.S.C.P. van der Woude</td>
<td>Optimal segmentations.</td>
</tr>
<tr>
<td>89/16</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>Towards a framework for comparing data models.</td>
</tr>
<tr>
<td>89/17</td>
<td>M.J. van Diepen, K.M. van Hee</td>
<td>A formal semantics for Z and the link between Z and the relational algebra.</td>
</tr>
<tr>
<td>90/2</td>
<td>K.M. van Hee, P.M.P. Rambags</td>
<td>Dynamic process creation in high-level Petri nets, pp. 19.</td>
</tr>
<tr>
<td>90/4</td>
<td>A. Peeters</td>
<td>Decomposition of delay-insensitive circuits, p. 25.</td>
</tr>
<tr>
<td>90/8</td>
<td>M.B. Josephs</td>
<td>Receptive Process Theory, p. 16.</td>
</tr>
<tr>
<td>90/9</td>
<td>A.T.M. Aerts, P.M.E. De Bra, K.M. van Hee</td>
<td>Combining the functional and the relational model, p. 15.</td>
</tr>
<tr>
<td>90/10</td>
<td>M.J. van Diepen, K.M. van Hee</td>
<td>A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).</td>
</tr>
<tr>
<td>90/11</td>
<td>P. America, F.S. de Boer</td>
<td>A proof system for process creation, p. 84.</td>
</tr>
<tr>
<td>90/12</td>
<td>P. America, F.S. de Boer</td>
<td>A proof theory for a sequential version of POOL, p. 110.</td>
</tr>
<tr>
<td>90/14</td>
<td>F.S. de Boer</td>
<td>A proof system for the language POOL, p. 70.</td>
</tr>
<tr>
<td>90/15</td>
<td>F.S. de Boer</td>
<td>Compositionality in the temporal logic of concurrent systems,</td>
</tr>
</tbody>
</table>
A fully abstract model for concurrent logic languages, p. 23.

On the asynchronous nature of communication in concurrent logic languages: a fully abstract model based on sequences, p. 29.