Die spesifikasietaal Z

- Skema:

$$\begin{array}{|l}\hline \textit{naam} \\ \textit{verklarings van veranderlikes} \\ (\text{“}\textit{signature}\text{''}) \\ \hline \textit{predikaat wat verwys na veranderlikes} \\ (\text{“}\textit{predicate}\text{''}) \\ \hline \end{array}$$

- Skemas word gebruik om *datastrukture* en *bewerkings* te beskryf

- Voorbeeld van 'n skema:

$$\begin{array}{|l}\hline \textit{Pop} \\ \textit{stack}, \textit{stack}' : \mathrm{seq}\, \textit{Item} \\ \textit{elem}! : \textit{Item} \\ \hline \langle \textit{elem}! \rangle \frown \textit{stack}' = \textit{stack} \\ \hline \end{array}$$

Notasie en konvensies

- Lineêre vorm van skema:

$$A \mathrel{\widehat{=}} [a, b : \mathbb{N};\ c : \mathbb{P}\,\mathbb{N} \mid a \in c \land b \in c]$$

- Ekwivalente skema (grafiese vorm):

$$\begin{array}{|l}\hline A \\ a, b : \mathbb{N} \\ c : \mathbb{P}\,\mathbb{N} \\ \hline a \in c \\ b \in c \\ \hline \end{array}$$

- Konvensie: aparte lyne in predikaat-deel word verbind met $\land$

Doel van spesifikasies

- Spesifikasies word geskryf as dokumente in natuurlike taal (in die industrie gewoonlik in Engels) met formele dele om belangrike begrippe te formaliseer.

- Spesifikasies op verskillende vlakke van detail:
  - beskrywing van *wat* 'n stelsel moet doen
  - beskrywing van *hoe* dit gedoen gaan word (dis die ontwerp, wat ontwerpsbesluite reflekteer)

Birthday Book Specification: Spivey

Simple system to record people's birthdays and issue a reminder. The system state is described by the following schema:

$$\begin{array}{|l}\hline \textit{BirthdayBook} \\ \textit{known} : \mathbb{P}\,\textit{NAME} \\ \textit{birthday} : \textit{NAME} \mathrel{+\!\!\!\rightarrow} \textit{DATE} \\ \hline \textit{known} = \mathrm{dom}\, \textit{birthday} \\ \hline \end{array}$$

- *known* is the set of recorded names.

- *birthday* is a function that gives the birthday associated with a given name.

- $\textit{known} = \mathrm{dom}\, \textit{birthday}$ is a system invariant to be maintained by every operation.

- No premature implementation decisions made at this stage:
  - maximum number of names that may be recorded not specified
  - no details of format for recording names and birthdays
  - no details about whether entries will be stored in any particular order
- Precision at conceptual level:
  - each person can have only one birthday.
  - two (or more) people can have the same birthday.

An operation to add a new entry:

$$
\begin{array}{|l}
\textit{AddBirthday} \\
\hline
\Delta \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{date}? : \textit{DATE} \\
\hline
\textit{name}? \notin \textit{known} \quad (\textit{precondition}) \\
\textit{birthday}' = \textit{birthday} \cup \{\textit{name}? \mapsto \textit{date}?\} \\
\hline
\end{array}
$$

Alternative notation:

$\textit{birthday}' = \textit{birthday} \cup \{(\textit{name}?, \textit{date}?)\}$

- The "Δ" indicates that the schema describes a *state change*.
- $\Delta \textit{BirthdayBook}$ stands for the combination of the schemas BirthdayBook and BirthdayBook'. (The union of the signature parts of the two schemas and the conjunction of their predicate parts.)
- Input variables are marked with a "?" suffix.

An operation to determine a person's birthday:

$$
\begin{array}{|l}
\textit{FindBirthday} \\
\hline
\Xi \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{date}! : \textit{DATE} \\
\hline
\textit{name}? \in \textit{known} \\
\textit{date}! = \textit{birthday}(\textit{name}?) \\
\hline
\end{array}
$$

- The "Ξ" indicates that the operation *does not change the state*.
- A name with a "!" suffix indicates that it represents an output.

An operation to determine which persons have birthdays on a given date:

```
┌ Remind ──────────────────────────────
│ ΞBirthdayBook
│ today? : DATE
│ cards! : ℙ NAME
├──────────────
│ cards! = {n : known | birthday(n) = today?}
└──────────────────────────────────────
```

Operation to initialise the system:

```
┌ InitBirthdayBook ────────────────────
│ BirthdayBook′
├──────────────
│ known′ = ∅
└──────────────────────────────────────
```


Comments regarding error situations

- What happens if the precondition of operation *AddBirthday* ($name \notin known$) does not hold?
- It seems that some operations will have to be refined to handle error situations.
- Refinements mean more detail.
- Specifications with too much detail are difficult to understand.
- Alternative: describe "normal" behaviour and "error" behaviour separately.
- Combine separate specifications using Z schema calculus.

Strengthening the specification

- Add an extra output *result!* to every operation so that the outcome of each operation can be described.
- Define an operation Success which produces the result "ok":

```
┌ Success ─────────────────────────────
│ result! : REPORT
├──────────────
│ result! = ok
└──────────────────────────────────────
```

- Use $\wedge$ operation of Z schema calculus to combine the descriptions of *AddBirthday* and *Success*:

  $AddBirthday \wedge Success$

- Define an operation to produce an error result for each error that can occur.
- If *name*! is already known when operation *AddBirthday* is executed, the operation *AlreadyKnown* will produce the result "already_known":

```
┌ AlreadyKnown ────────────────────────
│ ΞBirthdayBook
│ name? : NAME
│ result! : REPORT
├──────────────
│ name? ∈ known
│ result! = already_known
└──────────────────────────────────────
```

Robust operations

- A robust version of operation *AddBirthday* can now be defined:

  $RAddBirthDay \mathrel{\widehat{=}}$
  $(AddBirthday \land Success) \lor AlreadyKnown.$

- *Remind* has no precondition and the robust version is:

  $RRemind \mathrel{\widehat{=}} Remind \land Success.$

- In general, schemas can be combined by using operators of Z schema calculus to form new schemas.

RAddBirthday specified directly

$$
\begin{array}{l}
\textit{RaddBirthday} \\
\hline
\Delta \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{date}? : \textit{DATE} \\
\textit{result}! : \textit{REPORT} \\
\hline
(\textit{name}? \notin \textit{known} \land \\
\quad \textit{birthday}' = \textit{birthday} \cup \\
\quad\quad \{\textit{name}? \mapsto \textit{date}?\} \land \\
\quad \textit{result}! = \textit{ok}) \\
\lor \\
(\textit{name}? \in \textit{known} \land \\
\quad \textit{birthday}' = \textit{birthday} \land \\
\quad \textit{result}! = \textit{already\_known}) \\
\hline
\end{array}
$$

Advantages of Z Schema Calculus

- Schemas are kept simple by concentrating on just one issue in each schema.
- Unrelated issues are described separately.
- It does not mean that everything must be implemented separately.
- In general, no design framework is prescribed.
- Specification is a readable document that describes each issue separately and precisely.

Specification of a buffer

- The buffer will be used as temporary storage area.
- Items should be kept in first-in-first-out order.
- The buffer can store only a fixed number of items.
- Operations are needed to insert and remove items.

High-level description

Model the buffer which can store m natural numbers as a sequence. Use a counter c to keep track of the number of items stored. The state of the buffer is described by the following schema:

$$
\begin{array}{|l}
\hline
\textit{Buffer} \\
b : \operatorname{seq} \mathbb{N} \\
c : \mathbb{N} \\
\hline
c < m + 1 \\
\hline
\end{array}
$$

Operation *Insert* inserts items if there is enough space in the buffer:

$$
\begin{array}{|l}
\hline
\textit{Insert} \\
\Delta \textit{Buffer} \\
\textit{new}? : \mathbb{N} \\
\hline
c < m \\
c' = c + 1 \land b' = b \frown \langle \textit{new}? \rangle \\
\hline
\end{array}
$$

Operation *Remove* removes the first item from the buffer if it is not empty and returns the removed item in *item*!:

$$
\begin{array}{|l}
\hline
\textit{Remove} \\
\Delta \textit{Buffer} \\
\textit{item}! : \mathbb{N} \\
\hline
c > 0 \\
c' = c - 1 \land b = \langle \textit{item}! \rangle \frown b' \\
\hline
\end{array}
$$

Design (detailled specification)

- A design represents specific ideas about *how* a specification should be implemented.
- Design decisions are influenced by what is expected of a system: the level of efficiency, the memory requirements, etc.
- Designs are simply more detailed specifications and can be described in Z or other notations.
- Design decision for buffer: use an array of size m and indexes h (head) and t (tail) to indicate the first and last items in the buffer. Indexes wrap around at the end of the array.

Schema *BufferD* describes the design in more detail. An invariant is given to define the relation between h, t and c.

$$
\begin{array}{|l}
\hline
\textit{BufferD} \\
b : (0..m-1) \pinj \mathbb{N} \\
c : 0..m \\
h, t : 0..m-1 \\
\hline
t = (h + c) \bmod m \\
\hline
\end{array}
$$

How to initialise the buffer is described by the schema *Init*:

$$
\begin{array}{|l}
\hline
\textit{Init} \\
\Delta \textit{BufferD} \\
\hline
h' = 0 \land t' = 0 \land c' = 0 \\
\hline
\end{array}
$$

The schema *InsertD* describes how items are added to the buffer in terms of the proposed design:

$$
\begin{array}{|l}
\hline
\textit{InsertD} \\
\Delta \textit{BufferD} \\
\textit{new}? : \mathbb{N} \\
\hline
c < m \\
c' = c + 1 \land t' = (t + 1) \bmod m \\
b'(t) = \textit{new}? \\
\hline
\end{array}
$$

Schema *RemoveD* the detail of removing items from the buffer:

$$
\begin{array}{|l}
\hline
\textit{RemoveD} \\
\Delta \textit{BufferD} \\
\textit{item}! : \mathbb{N} \\
\hline
c > 0 \\
c' = c - 1 \land h' = (h + 1) \bmod m \\
\textit{item}! = b'(h) \\
\hline
\end{array}
$$