Модели на софтуерни системи

доц. Олга Георгиева

СУ ФМИ

катедра “Софтуерни технологии”

Модели на софтуерни системи

Лекция 5: Z нотация - Схеми

Олга Георгиева

СУ, Факултет по Математика и информатика,
Катедра СТ

Z нотация. Схеми: Съдържание

- **Начин за представяне на машини на състоянието**

    – Z глобални декларации

    - Z схеми (schemas)

(SCHEME[ski:m] 1. план, проект, програма;
2. схема, скица, диаграма, таблица, система, метод, начин)

- **Начин на разсъждение**

- **Начин за отразяване на връзките на машини на състоянието**

- **Предизвикателства:** сложност, разбираемост, елегантност

Въведение

Изискването за ясна, точна спецификация е в основата на всяко формализирано описание.

Една формална спецификация би трябвало да съдържа голямо количество проза. Тя трябва да съпоставя математическите обекти към особеностите на проектираната система: състояния на системата, структури от данни, техни свойства и операции с тях.

Съществуват два езика при **Z нотацията:**

– **математическият:** теорията на множествата и математическата логика;

- **езикът на схемите** се използва за *структурни* и *композиционни* описания: събиране на части от информацията, формулиране на общи описания и доказателства, необходими при следващо приложение.

Методът на Z

- **Комбинира математиката с прозата** като
  - > обяснява значението на символите;
  - > мотивира проектни решения

- **Използва идиоми за структуриране на големи спецификации** (структурни и композиционни описания)
  - > за да направи всяка част разбираема
  - > за да използва вече създадени обекти и части в спецификацията

- **Фокусира върху абстрактните характеристики на системата**
  - > свежда математическите описания на обектите *до описания на множества*, алгоритми, структури от данни,

Z математиката

- **Схемите** са именувани записи:
    - на тип
    - на декларация
    - на предикат

- **Теория на типовите множества и логика (Typed set theory and logic)**
    - множества, релации, функции
    - предикатна логика от първи ред

- **Изчисления със схеми (Schema calculus)**
    - Математически оператори, чрез които се изграждат по-големи схеми чрез по-малки

- **Синтактични събирания (Syntactic conventions)**
    - за да се направят описанията лесни за възприемане;
    - за да улесним програмирането.

Z схеми

Схемата съдържа:

Декларация: списък (деклариране) на променливите;

Предикат: дефинира/ограничава стойностите на променливите

Форми на запис:

хоризонтално: $[\mathit{declaration} \mid \mathit{predicate}]$

Вертикално:

declaration

predicate

Именуване:

$\mathit{Name} \mathrel{\widehat{=}} [\mathit{declaration} \mid \mathit{predicate}]$

Name

declaration

predicate

BirthdayBook

J.M. Spivey, The Z Notation, 1989;

Задание: A small database for recording people's names and birthdays.

Решение:

$$[NAME, DATE]$$

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

BirthdayBook

known – множество от имена със записани рождени дати;
birthday – частична функция

Едно възможно състояние на системата:

$$known = \{ \text{John}, \text{Mike}, \text{Susan} \}$$

$$\begin{aligned} birthday = \{ & \text{John} \mapsto \text{25–Mar}, \\ & \text{Mike} \mapsto \text{20–Dec}, \\ & \text{Susan} \mapsto \text{20–Dec} \}. \end{aligned}$$

Инвариантността се удовлетворява: (защо?...)

known = dom *birthday*

Наблюдение

От въведената спецификация е ясно, че:

Системата:

- няма ограничение за броя на въведените имена;
- няма установен/означен ред на въведените имена;
- няма ограничение на формата на въведената информация.

Системата има точно представена информация за:

- всеки човек има само един рожден ден;
- двама (и повече) могат да имат една и съща рождена дата;
- базата данни може да не е пълна.

Схеми – характеристики и използване

Те са основно средство за структуриране на формалните спецификации:
структури, които описват променливи, чиито стойности са ограничени по някакъв начин.

Схемата предоставя възможност за описание на:

- *структури от данни*
- *състояние на системата*
- *операции/действия на системата*

По този начин може да:

- Запишем връзките между отделните части на сложната система;
- Приложим мислене/заключение относно формалните спецификации за изследване на характеристиките на системата

Предимства на ***езика на схемите***:

1. Повторното използване (мултиплициране) е в основата на успешното използване на формалните методи;
2. Спомага за създаването и поддържането а добър стил на спецификацията;

Схеми

- **Еквивалентност**

Две схеми са еквивалентни, ако те въвеждат **едни и същи променливи** и дефинират **едни и същи ограничения** върху тях. Някои от ограниченията могат да бъдат скрити в декларативната част.

- *Четимост*: всяка декларация на отделен ред.

- За системи *без ограничения*: предикатната част се пропуска.

Схема като тип

Схемите като тип – възможност да се ***композира* тип** от различни компоненти.

Пр. Въвеждаме съставна схема с 2 компонента – цяло число ***a*** и множество от цели числа ***c***.

$SchemaOne$

$a : \mathbb{Z}$

$c : \mathbb{P}\,\mathbb{Z}$

За да запишем *обектите на схемата*, трябва да съставим списък от всички компоненти и техните стойности:

$$\langle\!|\, a \leadsto 2, c \leadsto \{1, 2, 3\} \,|\!\rangle$$

Схема като тип: пример

Example 11.9 A date is an object consisting of two named components: the name of a month and the number of a day. We may define *Month* as a free type with twelve constants:

$$Month ::= jan \mid feb \mid mar \mid apr \mid may \mid jun \mid jul \mid aug \mid sep \mid oct \mid nov \mid dec$$

Дефинирайте чрез схема множеството от датите (месец и ден).

$$\begin{array}{|l}
\textbf{Date} \\
\hline
month : Month \\
day : 1 \,..\, 31 \\
\hline
month \in \{sep, apr, jun, nov\} \Rightarrow day \leq 30 \\
month = feb \Rightarrow day \leq 29 \\
\hline
\end{array}$$

A binding $\langle\!| month \rightsquigarrow m, day \rightsquigarrow d |\!\rangle$ is a valid date provided that there are at least d days in month m. $\square$

Схема като тип

Компонентите на схемите са запазват не като позиция, а като имена. За да се отнесем до конкретен компонент използваме оператора за отделяне (*selection operator*) “ _**.**_ “.

Пр.:

Ако ***s*** е обект от типа схема ***SchemaOne***, то записите ***s.a*** и ***s.c*** отбелязват съответно компонентите цяло число и на множество от цели числа на обекта.

Зад.: Ако имаме деклариран обект от подобластта на типа Date, който да представя рождения ден на Fleur:

| *Fleur's birthday* : *Date*

запишете как ще се отнесете до месеца и деня на рождената дата на Fleur.

Отг.

Схемата като декларация

Ефектът на схемата като декларация е да се въведат променливи, споменати в декларативната част и ограничени в предикатната част.

Пример:

$$\begin{array}{|l}
\multicolumn{1}{l}{_\, SchemaTwo \,__} \\
a : \mathbb{Z} \\
c : \mathbb{P}\,\mathbb{Z} \\
\hline
a \in c \land c \neq \varnothing \\
\hline
\end{array}$$

The following set consists of those sets of integers c that contain the integer 0:

$$\{\, SchemaTwo \mid a = 0 \bullet c \,\}$$

The same effect could be achieved by replacing *SchemaTwo* with a list of declarations and a constraint:

$$\{\, a : \mathbb{Z};\ c : \mathbb{P}\,\mathbb{Z} \mid a \in c \land c \neq \varnothing \land a = 0 \bullet c \,\}$$

or by declaring an object of subrange type *SchemaTwo* and selecting the two components:

$$\{\, s : SchemaTwo \mid s.a = 0 \bullet s.c \,\}$$

The first expression, in which *SchemaTwo* is used as a declaration, is both more concise and more readable.

Схемата като декларация

Пример: Ако *Date* е схема, дефинирана в примера по-горе, каква информация описва следният запис?

$$\{\ Date \mid day = 31 \bullet month\ \}$$

Отг.:?

Схемата като предикат

Схемата може да се използва и като предикат, показвайки, че всеки компонент от схемата е вече бил деклариран като променлива от даден тип. Ефектът е да въведем ограничение, еквивалентно на предикатната информация, запазена в схемата.

Пр.: Всяко цяло ***a*** и множество от цели числа ***c*** , удовлетворяващи предиката на ***SchemaThree,*** трябва да удовлетворяват предиката на ***SchemaTwo***.

$$\forall\, a : \mathbb{Z};\ c : \mathbb{P}\,\mathbb{Z} \mid SchemaThree \bullet SchemaTwo$$

SchemaTwo

$a : \mathbb{Z}$

$c : \mathbb{P}\,\mathbb{Z}$

$a \in c \land c \neq \varnothing$

SchemaThree

$a : \mathbb{Z}$

$c : \mathbb{P}\,\mathbb{Z}$

$c \neq \varnothing \land a \in c$

$c \subseteq \{0, 1\}$

This is logically equivalent to the following statement:

$$\forall\, a : \mathbb{Z};\ c : \mathbb{P}\,\mathbb{Z} \mid c \notin \varnothing \land a \in c \land c \subseteq \{0, 1\} \bullet c \notin \varnothing \land a \in c$$

Схемата като предикат - нормализация

- Декларативната част може да съдържа ограничения.

$$
\begin{array}{|l}
\textit{SchemaTwo} \\
\hline
a : \mathbb{Z} \\
c : \mathbb{P}\,\mathbb{Z} \\
\hline
a \in c \land c \neq \emptyset \\
\hline
\end{array}
\qquad\qquad
\begin{array}{|l}
\textit{SchemaFour} \\
\hline
a : \mathbb{N} \\
c : \mathbb{P}\,\mathbb{N} \\
\hline
a \in c \land c \neq \emptyset \\
\hline
\end{array}
$$

За цялото число ***a*** и множество от цели числа ***c*** *са наложени допълнителни ограничения в* ***SchemaFour .***

- **Нормализация**: Декларативната част е редуцирана до уникална (единствена), канонична форма.

$$
\begin{array}{|l}
\textit{SchemaFourNormalised} \\
\hline
a : \mathbb{Z} \\
c : \mathbb{P}\,\mathbb{Z} \\
\hline
a \in \mathbb{N} \\
c \in \mathbb{P}\,\mathbb{N} \\
a \in c \land c \neq \emptyset \\
\hline
\end{array}
$$

Схеми – отрицание

Ако *S* е ***нормализирана*** схема

S

$a : A$

$b : B$

P

то отрицанието и’ е:

$\neg S$

$a : A$

$b : B$

$\neg P$

Преименуване

Въвежда нова колекция от променливи със същия образ/начин на деклариране и ограничения, вече зададени в дадена схема.

***Schema*[*new/old*]**

Пр.:

***SchemaTwo*[*q/a; s/c*]**

$$
\begin{array}{|l}
\hline
q : \mathbb{Z} \\
s : \mathbb{P}\,\mathbb{Z} \\
\hline
s \neq \varnothing \land q \in s \\
\hline
\end{array}
$$

Преименуване: пример

The variables *start_month* and *start_day* represent the month and the day on which a contract of employment is due to start. The requirement that this should be a valid date can be encapsulated by an appropriate renaming of the schema *Date*:

$$
\begin{array}{|l}
\textit{StartDate} \\
\hline
\textit{start\_month} : \textit{Month} \\
\textit{start\_day} : 1 \mathinner{\ldotp\ldotp} 31 \\
\hline
\textit{start\_month} \in \{\textit{sep}, \textit{apr}, \textit{jun}, \textit{nov}\} \Rightarrow \textit{start\_day} \leq 30 \\
\textit{start\_month} = \textit{feb} \Rightarrow \textit{start\_day} \leq 29 \\
\hline
\end{array}
$$

We may use another renaming to describe the set of all valid finish dates for our contract:

$$\textit{FinishDate} \mathrel{\widehat{=}} \textit{Date}[\textit{finish\_month} / \textit{month}, \textit{finish\_day} / \textit{day}]$$

A start date and a finish date are quite different objects, although each has a component of type *Month* and another of type $\mathbb{Z}$. If $s \in \textit{StartDate}$ and $f \in \textit{FinishDate}$, then the value of $s = f$ is undefined: these are variables of different types.

Общи схеми (Generic schemas)

Запазва се структурата на схемата, но се декларират различни типове.

Пр.:

$SchemaFive[X]$

$a : X$

$c : \mathbb{P}\, X$

$a \in c$

При $X = Z$ получаваме ***SchemaTwo* , *а при*** $X = N$ ***съответно* SchemaFour.**

Операции със схеми

- **Начин за комбиниране на информацията** в схемите и описание на поведението на системите.

- Въвежда се **език на логическите оператори на схемите** – конюнкция, дизюнкция, отрицание, композиция.

- Подходът се базира на концепцията за **абстрактните типове от данни**: колекция от променливи и списък операции, които могат да променят техните стойности**.**

- Операцията, която засяга промяната на състоянието на системата може да се разглежда като ***релация*** **между обектите на схемата** т.е. връзките между обектите на състоянието *преди и след прилагане на дадената операция*.

Схеми: конюнкция (1)

$S \wedge T$ *е конюнкция* на двете схеми: нова схема, в която декларативните части на S и T са слети, а предикатните са конюнктивно свързани:

$$\begin{array}{|l}
\multicolumn{1}{l}{S} \\ \hline
a : A \\
b : B \\ \hline
P \\ \hline
\end{array}
\qquad\qquad
\begin{array}{|l}
\multicolumn{1}{l}{T} \\ \hline
b : B \\
c : C \\ \hline
Q \\ \hline
\end{array}$$

$S \wedge T$

$$\begin{array}{|l}
\hline
a : A \\
b : B \\
c : C \\ \hline
P \wedge Q \\ \hline
\end{array}$$

Тази операция позволява да специфицираме *различни* аспекти (например състояния) *поотделно,* а след това да *комбинираме*, за да дефинираме по-сложни връзки (например преход на състояния).

Пример: BirthdayBook

***Пр.*:** Write a small database for recording people's names and birthdays. The system should allow us to:

- add new people to the database;
- lookup the birthday of a person;
- find the names of people with birthdays on a given day;

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

Схеми: конюнкция (2)

Вариант на конюнктивно свързани схеми:

включване на едната схема в декларативната част на друга. Така се:

- отразява йерархична структура;
- изпъква/повишава значението на дадено състояние.

Пр.:

1. **Начално състояние**

InitBirthdayBook

BirthdayBook

---

birthday = ∅

known = ∅

$InitBirthdayBook$

$BirthdayBook$

$known = \varnothing$

Схеми: декорация(1)

- Нека състоянието на система е моделирано със схема ***State***:

$$\begin{array}{|l} \textit{State} \\ \hline a : A \\ b : B \\ \hline P \\ \hline \end{array}$$

Всеки обект на схемата представя инвариантно състояние: свързани променливи ***a*** и ***b***, за които ***P*** винаги се удовлетворява.

- За да се **опише операция** върху това състояние се правят **две** копия на ***State*** : едно за състоянието преди и второ за състоянието след операцията.

- **Различимост** на двете състояния – чрез *декорация* на компонентите на втората схема и *модификация* на предикатната и’ част:

$$\begin{array}{|l} \textit{State}' \\ \hline a' : A \\ b' : B \\ \hline P[a'/a, b'/b] \\ \hline \end{array}$$

Схеми: декорация(2)

Описание на операция чрез включване на двете схеми в декларативната част на схемата за операцията:

Operation

State

State′

. . .

- Резултатната схема е с 4 компонента, като ***a*** и ***b*** валидират първото състояние, а ***a’*** и ***b’*** - второто.

- Предикатната част дефинира операцията като се описва ефекта върху стойностите на променливите на състоянието.

Схеми: вход и изход

За описание на операции, които включват вход и/или изход, се въвеждат допълнителни компоненти в декларативната част на схемата - операция, които предикатната част да използва.

Operation

State

State'

$i? : I$

$o! : O$

$\ldots$

i? – вход

o! - изход

BirthdayBook: Операции в системата (1)

Въвеждане на нова информация

> състояние преди операцията;

> състояние след операцията

> вход и изход

BirthdayBook

$known : \mathbb{P}\ NAME$

$birthday : NAME \pfun DATE$

$known = \mathrm{dom}\ birthday$

Декларация за промяна в състоянието: Δ

Пр.: **Add new people to the database**

Δ AddBirthdayBook

known, known': ℙ NAME

birthday, birthday': NAME⇸DATE

---

known = *dom* birthday

known' = *dom* birthday'

AddBirthday

$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

BirthdayBook: Разсъждения/Reasoning

Аавтоматично увеличаване на имената в системата с нововъведеното:

$$known' = known \cup \{name?\}$$

Доказателство: Твърдението се доказва с помощта на спецификацията на *AddBirthday* чрез инвариантите на състоянието преди и след операцията:

$$\begin{aligned}
known' &= \operatorname{dom} birthday' && \text{[invariant after]}\\
&= \operatorname{dom}(birthday \cup \{name? \mapsto date?\}) && \text{[spec. of } AddBirthday\text{]}\\
&= \operatorname{dom} birthday \cup \operatorname{dom} \{name? \mapsto date?\} && \text{[fact about `dom']}\\
&= \operatorname{dom} birthday \cup \{name?\} && \text{[fact about `dom']}\\
&= known \cup \{name?\}. && \text{[invariant before]}
\end{aligned}$$

? Кои два закона на математическите типове (данни) са използвани, за да докажем горното твърдение?

BirthdayBook: Операции в системата (2)

2. **Намиране на рождена дата на човек, познат на системата**.

Ξ - декларация, че състоянието не се променя т.е. $known' = known$

$$birthday' = birthday$$

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

Операции в системата (3)

3. **Напомняне – търсене на хора с дадена рождена дата**.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Remind}} \\ \hline
\Xi BirthdayBook \\
today? : DATE \\
cards! : \mathbb{P}\, NAME \\ \hline
cards! = \{\, n : known \mid birthday(n) = today? \,\} \\ \hline
\end{array}
$$

Тогава името ***m*** ще е в изходящия списък ***cards!***, ако е *познато* на системата и ако рождения ден, записан за него е ***today***.

$$
\begin{aligned}
&m \in \{\, n : known \mid birthday(n) = today? \,\} \\
&\qquad \Leftrightarrow m \in known \land birthday(m) = today? \,.
\end{aligned}
$$

Обобщение

Пространство на състоянията

+

Операции

Ако условието (pre-condition) за някоя операция се наруши, то системата може да:

– игнорира операцията;
– се провали (crash);
– прекъсне по-късно.

Обобщение - Справяне с грешките

- Грешките, които могат да бъдат отчетени, както и съответните желани отговори, могат да бъдат описани отделно от първата спецификация.

- С помощта на операциите със Z схеми можем да комбинираме двете описания в по-строга спецификация.

Пр.: Да се модифицира всяка операция, така че да връща резултат *result!*, указващ една от трите възможни стойности на резултата от извършената операция:

REPORT ::= ok | already-known | not-known

Success

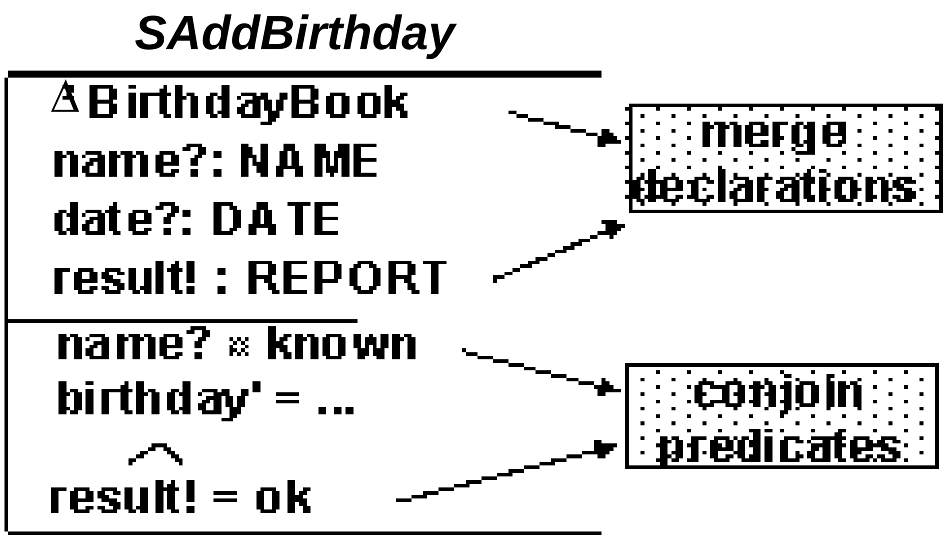
$result! : REPORT$

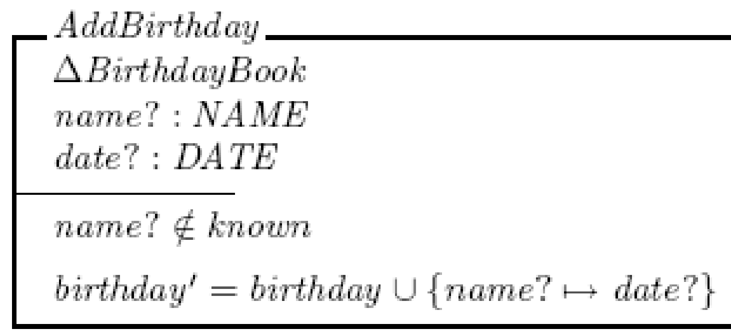
$result! = ok$

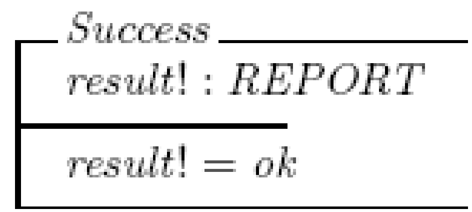
Add Birthday – revised (1)

SAddBirthday $\widehat{=}$ ***AddBirthday*** $\wedge$ ***Success***

или алтернативно:







Зад. (самостоятелна работа):

А) Да се реализира схема за случая, когато името вече е записано в системата.

Б) Да се реализира схема за случая, когато при операцията за проверка на рождена дата името не е познато.

Add Birthday – revised (2)

Отг. А)

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AlreadyKnown}} \\
\hline
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \in known \\
result! = already\_known \\
\hline
\end{array}
$$

Кратък запис?

Б)

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NotKnown}} \\
\hline
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \notin known \\
result! = not\_known \\
\hline
\end{array}
$$

Кратък запис?

Схеми – дизюнкция

Описание на алтернативи в поведението на система:

If

S

$a : A$
$b : B$

P

T

$b : B$
$c : C$

Q

then $S \vee T$ is the schema

$a : A$
$b : B$
$c : C$

$P \vee Q$

Add Birthday – робастна версия

Робастна версия:

RAddBirthday $\widehat{=}$ (AddBirthday $\wedge$ Success) $\vee$ AlreadyKnown

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

Пример: BirthdayBook - други робастни операции

Дефинирайте робастна версия на Find Birthday

$$RFindBirthday \mathrel{\widehat{=}} (FindBirthday \land Success) \lor NotKnown\,.$$

и на Remind:

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

Непознато име:

$$
\begin{array}{|l}
\hline
\textit{NotKnown} \\
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \notin known \\
result! = not\_known \\
\hline
\end{array}
$$

Предимства на подхода:

• Разделение на работата:

> разглежда всяка идея отделно

• Фокусира върху

> раздробяване на работата/изпълнението на разумно големи парчета

• Модулен принцип

Наблюдение

• Могат да бъдат комбинирани спецификации с използване на конюнкция и дизюнкция за изчисления със схеми.

even though you can't combine programs!

Друго използване на изчисленията със схеми

Отделя:

> единичен обект (process, file, record) и го поставя на съответното му място в по-голямата система;

> различни погледи на една и съща система;

> описва фукционирането на системата и управлява достъпа;

Заключение

- **Z е проста математическа рамка**, в която да:
  - опишем системата абстрактно и същевременно точно;
  - композираме системата с малки части/елементи;
  - използваме стари спецификации, за да построим нови;
  - аргументираме характеристиките на системата;
  - обясним връзките в системата (relate views of a system);