The Z Schema Calculus

Slides by
Prof. Shmuel Katz
Computer Science Department
The Technion
The idea of Z

- Precise set-based: clear meaning
- Allows understanding and analyzing structure
- Allows clean modularity (e.g., for exceptions)
- Easy to extend with new operations
- Defines an abstract system state, and then how each operation changes the state
- Can be used to analyze legacy systems

Problems:
- too dense/mathematical
- Operations as “instantaneous” mathematical functions
Reminder: Schemas

- A notation specifying both system states and operations
- One way: \( S = [ \text{declarations} \mid \text{predicate} ] \)
- Means: The state components in the declaration satisfy the predicate
- Another way to write a schema:

\[
S \\
\text{declarations} \\
\text{predicate}
\]
Reminder: Library Data Schema

Library

stock: Copy \rightarrow Book
issued: Copy \rightarrow Reader
shelved: \mathcal{F} Copy
readers: \mathcal{F} Reader
limit: \mathbb{N}

shelved \cup \text{dom} \text{issued} = \text{dom} \text{stock}
shelved \cap \text{dom} \text{issued} = \{ \}
ran \text{issued} \subseteq \text{readers}
\forall r : \text{readers} \cdot \#(\text{issued} \triangleright \{ r \} ) \leq \text{limit}

Signature: variables
And their type

Invariants- relations among the variables true in every system state
Reminder: Borrowing a Book Operation Schema

Borrow

Library . Library'
c? : Copy
r? : Reader

c? ∈ shelved
r? ∈ readers
#(issued ▹ { r? }) < limit
issued' = issued ⊕ { c? ↦ r? }
readers' = readers
stock' = stock
Lecture Outline

• A Second Example: Telephone Net
  > Adapted from "Telephone Network", by Carroll Morgan
  > In Specification Case Studies, Ian Hayes (ed)

• Concepts
  > Use of strong state invariants to make concise models
  > Disj, (Generic Definitions)
  > Framing
  > Reasoning with state invariants

• Schema Calculus
  • Use of schema calculus to simplify descriptions and associated reasoning
Telephone Net

◊ Telephones
Requests for Connection

Telephones

Requests
Connections

- Telephones
- Requests
- Connections
Operations

• *Call*: Request a connection between two phones
• *Hangup*: Terminate a connection
• *Busy*: Report whether a phone is busy
State Space

\[ \text{PHONE} \]

\text{CONNECTION} == \mathcal{P} \text{PHONE}

\begin{align*}
\text{TelephoneNet} & \\
\text{reqs, cons:} & \mathcal{P} \text{CONNECTION} \\
\text{cons} & \subseteq \text{reqs} \\
\text{disj cons} &
\end{align*}
Example

PHONE = \{1, 2, 3 \} 

CONNECTIONS = 
\{ \{\}, \{1\}, \{2\}, \{3\}, \{1,2\} \{1,3\}, \{2,3\}, \{1,2,3\} \} 

TelephoneNet

reqs: \{ \{1,2\}, \{2,3\} \} 

cons: \{ \{1,2\} \}
Pairwise disjoint sets

**Definition**: (informal) A set of sets is disjoint iff no pair of those sets has elements in common.

**Definition**: (formal)

\[ \text{disj cons} \iff \forall c1, c2 : \text{cons} \cdot c1 \neq c2 \Rightarrow c1 \cap c2 = \emptyset \]

**Examples**: Which sets are \textit{disj}?

(1) \{ \{a\}, \{b,c\}, \{d\} \} \hspace{1cm} (2) \{\emptyset, \{a\}\}  
(3) \{\{a,b\}, \{c,d\}, \{a,c\}\} \hspace{1cm} (4) \{\{a,b\}, \{b, a\}\}
What's Wrong With This Net?

Telephones
Requests
Connections
An Inefficient Net

InefficientNet :
\[ \exists r : \text{reqs} \setminus \text{cons} \cdot \text{disj} (\text{cons} \cup \{r\}) \]
Efficient Networks

EfficientTN

TelephoneNet

¬ (∃ r: reqs \ cons • disj (cons \cup \{r\}))
EfficientTN 

\[
\begin{align*}
\text{reqs, cons: } & \mathbb{P} \text{ CONNECTION} \\
\text{cons } & \subseteq \text{ reqs} \\
\text{disj cons} \\
\neg (\exists \ r: \text{ reqs } \setminus \text{ cons} \cdot \text{disj (cons } \cup \{r\} ))
\end{align*}
\]
Initial Telephone Net

InitTN

EfficientTN'

reqs' = cons' = ∅
Operation: Call

\[ \text{Call} \]

\[ \triangle \text{EfficientTN} \]

\[
\begin{align*}
\text{ph?, dialled?} & : \text{PHONE} \\
\text{reqs'} & = \text{reqs} \cup \{ \{\text{ph?}, \text{dialled?}\} \}
\end{align*}
\]
Example
Example

Must the new request become a connection?
Example

Yes, EfficientTN.

Is this the only possible result of Call?
Example

Is this allowed?
Operation: Call

Call

\[ \Delta \text{EfficientTN} \]
\[ \text{ph?}, \text{dialled?} : \text{PHONE} \]
\[ \text{reqs'} = \text{reqs} \cup \{ \{\text{ph?}, \text{dialled?}\} \} \]

EfficientTN will take care of the connection, but what about pre-existing connections?

EfficientTN only promises \textit{cons} to be a \textit{disj} subset of \textit{reqs}!
Framing: Event

\[ \Delta \text{EfficientTN} \]

\[ \forall c: \text{CONNECTION} \cdot c \in (\text{cons} \cap \text{reqs'}) \Rightarrow c \in \text{cons}' \]
This Really Means ...

Event

\[ \text{reqs, cons: P PHONE} \]
\[ \text{reqs', cons': P PHONE} \]

\[ \text{cons} \subseteq \text{reqs} \]
\[ \text{disj cons} \]
\[ \text{cons'} \subseteq \text{reqs'} \]
\[ \text{disj cons'} \]
\[ \neg ( \exists r: \text{reqs} \setminus \text{cons} \cdot \text{disj (cons} \cup \{r\}) \) \]
\[ \neg ( \exists r: \text{reqs'} \setminus \text{cons'} \cdot \text{disj (cons'} \cup \{r\}) \) \]
\[ \forall c: \text{CONNECTION} \cdot c \in (\text{cons} \cap \text{reqs'}) \Rightarrow c \in \text{cons'} \]

But we would not be expected to write it out this way.
Updated Call

Call

Event

\[ \text{ph?, dialled? : PHONE} \]

\[ \text{reqs'} = \text{reqs} \cup \{ \{\text{ph?}, \text{dialled?}\} \} \]
Is this realistic?
YesOrNo ::= Yes | No

Busy

Event
ph? : PHONE
busy! : YesOrNo

reqs' = reqs
busy! = Yes ↔ ph? ∈ cons

What happens to cons?
Reasoning about the Specification

- **Theorem**: (Informal) If an operation doesn't change any of the requests then it doesn't change any of the connections.
- **Proof**: (Informal)
  1. The constraint on Event tells us that any original connection won't be broken if it is still requested.
  2. If an operation doesn't change the requests, all original connections will still be requested.
  3. Hence, no connections are broken.
  4. Also, no new connections are added because if a connection could be added afterwards, it could have been added before.
  5. So connections aren't changed.
Schema Calculus

- Use of schema calculus to simplify descriptions and associated reasoning
- Inclusion (like Library inside Borrow):
  - merge the declarations, conjunction of the predicates
- $S \land T$:
  - symmetric version of inclusion
- $S \lor T$:
  - merge declarations, $\lor$ between predicates
- $\neg S$:
  - like $S$, but negating the predicate
Schema Variable Substitution

- **S[ new/old] substitution:**
  > S with *new* in place of *old*
  » What if *new* already exists?

---

**Sample**

\[
\begin{align*}
x, \ y, \ z & : \ \mathbb{N} \\
\text{x < y} \land \text{y < z}
\end{align*}
\]

**Sample[w/y]**

\[
\begin{align*}
x, \ w, \ z & : \ \mathbb{N} \\
\text{x < w} \land \text{w < z}
\end{align*}
\]
Hiding

• $S \setminus \{ v_1, \ldots, v_n \}$: hiding variables
  > remove $v_1, \ldots, v_n$ from $S$’s declaration, and instead
  add $\exists v_1, \ldots, v_n$ before $S$’s predicate
  » Now just says “there are values” that satisfy the
  predicate

Sample

\[
\begin{align*}
x, y, z : \mathbb{N} \\
x < y \land y < z
\end{align*}
\]

Sample $\setminus \{y\}$

\[
\begin{align*}
x, z : \mathbb{N} \\
\exists y : \mathbb{N}. \ x < y \land y < z
\end{align*}
\]
pre S— A Derived Schema

- pre S is the precondition for applying S
  - Defined as $S \setminus \{\text{all variables with } ! \text{ or } '\}$
  - Hide everything from the result or output
- for Borrow?

```
  └───────────────────────────
    r? ∈ readers
    issued' = issued + \{ c? ← r? \}
    readers' = readers
    stock' = stock
```

```
  └───────────────────────────
    c? ∈ shelved
    #\{r?\} < limit
```

```
  ΔLibrary
  c?: Copy
  r?: Reader
```

```
pre Borrow

- **Pre Borrow == Borrow \ {Library’}**

```
Borrow ─────────────────────────────────
\Library
 c?: Copy
 r?: Reader


├───────────────────────────
| c? ∈ shelved
| r? ∈ readers
| #\(\text{issued} \triangleright \{r?\}\) < limit

∃\text{issued’} : Copy → Reader . \text{issued’} = \text{issued} ∪ \{c? ↔ r?\}
∃\text{readers’} : \lnot Reader . \text{readers’} = \text{readers}
∃\text{stock’} : Copy → Book . \text{stock’} = \text{stock}
```

©S. Katz, 2011
A (standard) trick

- Can get rid of the $\exists$ by substituting an expression equal to the hidden variable:
  \[ \exists x:S \cdot ((x = T) \land P) \iff (T \in S) \land P[T/x] \]

Doing this for the example, and simplifying, gives:

<table>
<thead>
<tr>
<th>pre Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
</tr>
<tr>
<td>c?: Copy</td>
</tr>
<tr>
<td>r?: Reader</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>c? ∈ shelved</td>
</tr>
<tr>
<td>r? ∈ readers</td>
</tr>
<tr>
<td>#(issued △ { r? }) &lt; limit</td>
</tr>
</tbody>
</table>
Schema Composition $A \bowtie B$

- Idea: connect the result of $A$ with the initial state of $B$, and hide them both
- Formally: $A \bowtie B \equiv (A[\text{new}/x'] \land B[\text{new}/x]) \setminus \{\text{new}\}$
  > Both should declare $x$ and $x'$
  > Substitution occurs for all state variables

```
F ┌─── s, s', i? : \mathbb{N} │ T ┌─── s, s', o! : \mathbb{N}│
└──── s' = i? + s │    └──────── s' = 2s \land o! = s'
```

- Then $F \bowtie T$ is

```
F;T ┌─── s, s', i?, o! : \mathbb{N} │
└──── \exists \text{ new}: \mathbb{N} \cdot \text{new} = i? + s \land s' = 2\text{new} \land o! = s'
```

Apply trick..
Modularity: an example

- A symbol table; \( ST : STR \rightarrow VAL \)

\[
\begin{align*}
\text{Enter} & \quad \text{st}, \text{st}^{'}, \text{ST} \\
\text{Enter} & \quad \text{s?}, \text{v?}, \text{STR}, \text{VAL} \\
\text{Init} & \quad \text{st}^{'}, \text{ST} \\
\text{Init} & \quad \text{st}^{'}, \text{ST} \\
\text{Init} & \quad \text{st}^{'}, \text{ST} \\
\text{Init} & \quad \text{st}^{'}, \text{ST} \\
\end{align*}
\]
What if $s?$ is “illegal”?

\[\text{BadLookup} \]
\[
\exists \text{st} : \text{ST} \\
\text{s?} : \text{STR} ; \text{rep!} : \text{STR} \times \text{STR}
\]
\[
\text{s?} \notin \text{dom} \text{ st} \\
\text{rep!} = (\text{“string not in table: ”}, \text{s?})
\]

A more robust Lookup operation is then: $R\text{lookup} = \text{Lookup} \lor \text{BadLookup}$
Logging

Lookup

\[ \text{st, st}' : ST \]
\[ \text{s? : STR ; v! : VAL} \]

\[ \text{s?} \in \text{dom st} \]
\[ \text{v!} = \text{st} \text{s?} \]
\[ \text{st}' = \text{st} \]

Could also have:

Log

\[ \text{s? : STR ; rep! : STR} \times \text{STR} \]

\[ \text{rep!} = ("a successful lookup: ", \text{s?}) \]

AugLookup = (Lookup \land Log) \lor BadLookup
Zed Tools

- **Espino, Luis**, *ERZ: Tool for to transform ER model to Z Notation equivalent*.
- **Community Z Tools (CZT)** (project), Source forge.
- **Z Word tools** (project), Source forge for developing and checking Z specifications in *Microsoft Word*.
- Spivey, Michael ‘Mike’, *Fuzz Type-Checker for Z*.
- **Z/Eves — A proof checker for the Z notation** (German site but all manuals in English)
- **Z/EVES** Documentation, papers, and manuals on Z/EVES
- **ZETA** open-source system for development software specifications in Z
- **HOL-Z** open-source proof environment for Z in Isabelle/HOL
- **CADiZ**, a set of free software tools that assist use of Z notation
Zed Tools

- **ProofPower**, a suite of open-source tools supporting specification and proof in HOL and in the Z notation
- **Vimes** An independently developed type checker.
- **z-vimes** Alternate source of Vimes.
- **Z User Group (ZUG)**
- **Community Z Tools (CZT) project**
- **Other formal methods** (and languages using formal specifications):
  - Z++ and **Object-Z**: object extensions for the Z notation
  - Abstract Machine Notation (AMN), used in **B-Method**
  - **Alloy**, a specification language inspired by Z notation and implementing the principles of **Object Constraint Language** (OCL).
- **Fastest** is a model-based testing tool for the Z notation.