Reasoning in Z

Software Engineering

Andreas Zeller • Saarland University

$$Forward$$
$$\Delta Editor$$
$$ch? : CHAR$$
$$ch? = right\_arrow$$
$$right \neq \langle\rangle$$
$$left' = left \frown head(right)$$
$$right' = tail(right)$$

1

These slides are based on Jackys “The Way of Z”

Outline

- The schema calculus
- Case Study: Text Editor
- Formal Reasoning
- Case Study: Expert System

2

The Schema Calculus

Z includes two languages: the language of ordinary discrete mathematics and the *schema language.*

Schemas are *the* characteristic construct of Z!

They allow:

- Macro like inclusion and reuse of definitions,
- incremental specification,
- specification as a mix between text and formal definitions.

3

Including State Schemas

Editor

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$

Init

Editor ?

$left = right = \langle\,\rangle$

Including State Schemas

Editor

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$

Init

Editor ?

$left = right = \langle\,\rangle$

actually means

Init

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$
$left = right = \langle\,\rangle$

Including Operation Schemas

Insert

$\Delta Editor$?
$ch? : CHAR$

$ch? \in printing$
$left' = left \frown \langle ch? \rangle$
$right' = right$

What does $\Delta Editor$ mean?

Including Operation Schemas

We can decorate whole schemas with suffixes, prefixes, etc. This affects all variables defined by the schema.

Editor

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$

Editor′

Including Operation Schemas

We can decorate whole schemas with suffixes, prefixes, etc. This affects all variables defined by the schema.

Editor

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$

Editor′

$$left', right' : TEXT$$

$$\#(left' \frown right') \leq maxsize$$

Including Operation Schemas

ΔS is defined for all schemas as:

ΔS

$$S$$
$$S'$$

If we expand $\Delta Editor$, we get:

$\Delta Editor$

$$left, right : TEXT$$
$$left', right' : TEXT$$

$$\#(left \frown right) \leq maxsize$$
$$\#(left' \frown right') \leq maxsize$$

Including Operation Schemas

Thus the *Insert* schema expands to:

Insert

$$left, right, left', right' : TEXT$$
$$ch? : CHAR$$

$$ch? \in printing$$
$$\#(left \frown right) \leq maxsize$$
$$\#(left' \frown right') \leq maxsize$$
$$left' = left \frown \langle ch? \rangle$$
$$right' = right$$

Including Operation Schemas

Ξ is a further abbreviation—it stands for the schema where everything remains *unchanged.*

$\Xi Editor$

$$\Delta Editor$$

$$left' = left$$
$$right' = right$$

expands to:

$\Xi Editor$

$$left, right, left', right' : TEXT$$

$$\#(left \frown right) \leq maxsize$$
$$\#(left' \frown right') \leq maxsize$$
$$left' = left$$
$$right' = right$$

Schema Operators

Quotient

$$n, d, q, r : \mathbb{N}$$

$$d \neq 0$$
$$n = q * d + r$$

Remainder

$$r, d : \mathbb{N}$$

$$r < d$$

What is

$$Division \mathrel{\widehat{=}} Quotient \land Remainder \quad ?$$

Schema Calculus: Conjunction

Quotient

$n, d, q, r : \mathbb{N}$

$d \neq 0$

$n = q * d + r$

Remainder

$r, d : \mathbb{N}$

$r < d$

Division $(\widehat{=}$ *Quotient* $\wedge$ *Remainder*$)$

$n, d, q, r : \mathbb{N}$

$d \neq 0$

$n = q * d + r$

$r < d$

Schema Calculus: Overview

A schema conjunction

- combines all declarations and
- all predicates are combined to a *conjunction*

Analog—A *disjunction*

- combines all declarations and
- all predicates are combined to a *disjunction*

Schema Calculus: Disjunction

DivideByZero

$d, q, r : \mathbb{N}$

$d = 0 \wedge q = 0 \wedge r = 0$

$T\_Division \mathrel{\widehat{=}} Division \vee DivideByZero$

Schema Calculus: Disjunction

DivideByZero

$$d, q, r : \mathbb{N}$$

$$d = 0 \land q = 0 \land r = 0$$

$T\_Division \mathrel{\widehat{=}} Division \lor DivideByZero$

results in:

T_Division

$$n, d, q, r : \mathbb{N}$$

$$(d \neq 0 \land r < d \land n = q * d + r) \lor$$
$$(d = 0 \land r = 0 \land q = 0)$$

Schema Calculus: Negation

The negation of a schema is not very useful.
Let's look at the EOF schema.

EOF

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$
$$right = \langle\,\rangle$$

Schema Calculus: Negation

The negation of a schema is not very useful.
Let's look at the EOF schema.

EOF

$$left, right : TEXT$$

$$\#(left \frown right) \leq maxsize$$
$$right = \langle\,\rangle$$

$\neg\, EOF$ becomes:

$\neg\, EOF$

$$left, right : TEXT$$

$$left \notin \operatorname{seq} CHAR \lor$$
$$right \notin \operatorname{seq} CHAR \lor$$
$$\#(left \frown right) > maxsize \lor$$
$$right \neq \langle\,\rangle$$

Schema Calculus: Negation

Better: A new schema that negates *some* of the predicates.

EOF

$$Editor$$

$$right = \langle\,\rangle$$

Becomes:

NotEOF

$$Editor$$

$$right \neq \langle\,\rangle$$

Case Study: Text Editor

Task: Breaking text into words.

$$words\langle H, o, w, \ , a, r, e, \ , y, o, u\rangle = \langle\langle H, o, w\rangle, \langle a, r, e\rangle, \langle y, o, u\rangle\rangle$$

Case Study: Text Editor

Task: Breaking text into words.

$$words\langle H, o, w, \ , a, r, e, \ , y, o, u\rangle = \langle\langle H, o, w\rangle, \langle a, r, e\rangle, \langle y, o, u\rangle\rangle$$

We define the basic types:

$$[CHAR]$$

$$blank : \mathbb{P}\, CHAR$$

$$TEXT == \mathrm{seq}\, CHAR$$
$$SPACE == \mathrm{seq}_1\, blank$$
$$WORD == \mathrm{seq}_1 (CHAR \setminus blank)$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet$$
$$words(\langle\,\rangle) = \langle\,\rangle$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet$$
$$words(\langle\,\rangle) = \langle\,\rangle \land$$
$$words(s) = \langle\,\rangle$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet$$
$$words(\langle\,\rangle) = \langle\,\rangle \land$$
$$words(s) = \langle\,\rangle \land$$
$$words(w) = \langle w \rangle$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\begin{aligned}
&\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet \\
&\quad words(\langle\,\rangle) = \langle\,\rangle \land \\
&\quad words(s) = \langle\,\rangle \land \\
&\quad words(w) = \langle w \rangle \land \\
&\quad words(s \frown r) = words(r)
\end{aligned}$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\begin{aligned}
&\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet \\
&\quad words(\langle\,\rangle) = \langle\,\rangle \land \\
&\quad words(s) = \langle\,\rangle \land \\
&\quad words(w) = \langle w \rangle \land \\
&\quad words(s \frown r) = words(r) \land \\
&\quad words(l \frown s) = words(l)
\end{aligned}$$

Case Study: Text Editor

The definition of *words* looks like this:

$$words : TEXT \rightarrow \text{seq}\, WORD$$

$$\begin{aligned}
&\forall\, s : SPACE;\ w : WORD;\ l, r : TEXT \bullet \\
&\quad words(\langle\,\rangle) = \langle\,\rangle \land \\
&\quad words(s) = \langle\,\rangle \land \\
&\quad words(w) = \langle w \rangle \land \\
&\quad words(s \frown r) = words(r) \land \\
&\quad words(l \frown s) = words(l) \land \\
&\quad words(l \frown s \frown r) = words(l) \frown words(r)
\end{aligned}$$

Case Study: Text Editor

We want to count the number of words and line breaks—analog to the UNIX wc tool:

```
$ wc zed.tex
1060    2750   22413 zed.tex
$ _
```


Case Study: Text Editor

We want to count the number of words and line breaks—analog to the UNIX wc tool:

```
$ wc zed.tex
1060    2750   22413 zed.tex
$ _
```

This is not very difficult.

$$wc : TEXT \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

$$\forall\, file : TEXT \bullet wc(file) = (\#lines(file), \#words(file), \#file)$$

Exercise: Define *lines*!

Case Study: Text Editor

Another task—filling paragraphs:

```
Almost any text editor provides a fill
operation. The fill operation transforms ragged-right text
with lines of
different lengths into nicely formatted text with lines
nearly the same length.
```

should become

```
Almost any text editor provides a fill operation. The
fill operation transforms ragged-right text with lines
of different lengths into nicely formatted text with lines
nearly the same length.
```

Case Study: Text Editor

$width : \mathbb{N}$

Format

$t, t' : TEXT$

$words(t') = words(t)$
$\forall\, l : \operatorname{ran} lines(t') \bullet \#l \leq width$

Case Study: Text Editor

$width : \mathbb{N}$

Format

$t, t' : TEXT$

$words(t') = words(t)$
$\forall\, l : \operatorname{ran} lines(t') \bullet \#l \leq width$

Fill

Format

$\#lines(t') = min\{t' : TEXT \mid Format \bullet \#lines(t')\}$

Exercise: What does the last schema mean?

Formal Reasoning

Example:

A train moves at a constant velocity of sixty miles per hour. How far does the train travel in four hours?

As Z specification:

$distance, velocity, time : \mathbb{N}$

$distance = velocity * time$
$velocity = 60$
$time = 4$

Question: What is *distance*?

Proof

We proof that $distance = 240$ holds:

$distance = velocity * time$ [Definition]

Proof

We proof that $distance = 240$ holds:

$distance = velocity * time$ [Definition]
$= 60 * time$ [$velocity = 60$]

Proof

We proof that $distance = 240$ holds:

$distance = velocity * time$ [Definition]
$= 60 * time$ [$velocity = 60$]
$= 60 * 4$ [$time = 4$]

Proof

We proof that $distance = 240$ holds:

$distance = velocity * time$ [Definition]
$= 60 * time$ [$velocity = 60$]
$= 60 * 4$ [$time = 4$]
$= 240$ [Arithmetic]

Proof

We proof that $distance = 240$ holds:

$distance = velocity * time$ [Definition]
$= 60 * time$ [$velocity = 60$]
$= 60 * 4$ [$time = 4$]
$= 240$ [Arithmetic]

This is *strict reasoning:* every step is based on a rule.

Proof

We prove that $distance = 240$ holds:

$distance = velocity * time$ [Definition]
$= 60 * time$ [$velocity = 60$]
$= 60 * 4$ [$time = 4$]
$= 240$ [Arithmetic]

This is *strict reasoning:* every step is based on a rule.

Written in one line, this becomes:

$distance = velocity * time = 60 * time = 60 * 4 = 240$

Non Arithmetic Reasoning

$$\frac{\begin{array}{l} philip : PERSON \\ adhesives, materials, research, manufacturing : \mathbb{P}\, PERSON \end{array}}{\begin{array}{l} adhesives \subseteq materials \\ materials \subseteq research \\ philip \in adhesives \end{array}}$$

Question: Does $philip \in research$ hold?

Non Arithmetic Reasoning

$$\frac{\begin{array}{l} philip : PERSON \\ adhesives, materials, research, manufacturing : \mathbb{P}\, PERSON \end{array}}{\begin{array}{l} adhesives \subseteq materials \\ materials \subseteq research \\ philip \in adhesives \end{array}}$$

Question: Does $philip \in research$ hold?

Yes, because $philip \in adhesives$ [Definition]
$\subseteq materials$ [Definition]
$\subseteq research$ [Definition]

Where we conclude $S \subseteq T \subseteq U \Rightarrow S \subseteq U$

Another Proof

$$\frac{x : \mathbb{Z}}{2 * x + 7 = 13}$$

We conclude the value of x:

$2 * x + 7 = 13$ [Definition]
$\Leftrightarrow 2 * x = 13 - 7$ [Subtract 7 on both sides]
$\Leftrightarrow 2 * x = 6$ [Arithmetic]

Another Proof

$$\frac{x : \mathbb{Z}}{2 * x + 7 = 13}$$

We conclude the value of x:

$2 * x + 7 = 13$	[Definition]
$\Leftrightarrow 2 * x = 13 - 7$	[Subtract 7 on both sides]
$\Leftrightarrow 2 * x = 6$	[Arithmetic]
$\Rightarrow (2 * x) \operatorname{div} 2 = 6 \operatorname{div} 2$	[Divide both sides by 2]
$\Leftrightarrow x = 6 \operatorname{div} 2$	[Division on the left; Algebra]
$\Leftrightarrow x = 3$	[Arithmetic]

Checking Specifications

Inconsistency: Specifications that contradict each other.

Example: *Existence of an initial state*

$$\exists\, Editor \bullet Init$$

„There is an Editor that satisfies the predicate of *Init*“

Checking Specifications

Inconsistency: Specifications that contradict each other.

Example: *Existence of an initial state*

$$\exists\, Editor \bullet Init$$

„There is an Editor that satisfies the predicate of *Init*“

Expands to:

$$\exists\, left, right : TEXT \mid \#(left \frown right) \leq maxsize \bullet \\ left = right = \langle\,\rangle$$

Obviously true—but how to prove this formally?

Determining Preconditions

Many program failures occur because programmers did not account for all the *preconditions*.

Insert

$\Delta Editor$
$ch? : CHAR$

$ch? \in printing$
$left' = left \frown \langle ch? \rangle$
$right' = right$

Are there some non-obvious preconditions?

Determining Preconditions

A precondition describes all states an operation is defined for.

A general precondition of operation schema Op on state schema S is that there is a *following schema* S'

$$\exists S' \bullet Op$$

In our case:

$$\exists Editor' \bullet Insert$$

Short: After *Insert* there is also an *Editor*.

Determining Preconditions

Editor

$left, right : TEXT$

$\#(left \frown right) \leq maxsize$

$$\exists Editor' \bullet Insert$$

becomes:

$$\exists left', right' : TEXT \mid \#(left' \frown right') \leq maxsize \bullet ch? \in printing \land left' = left \frown \langle ch? \rangle \land right' = right$$

Determining Preconditions

We have:

$$\exists\, left', right' : TEXT \mid \#(left' \frown right') \leq maxsize \bullet$$
$$ch? \in printing \land left' = left \frown \langle ch? \rangle \land right' = right$$

Determining Preconditions

We have:

$$\exists\, left', right' : TEXT \mid \#(left' \frown right') \leq maxsize \bullet$$
$$ch? \in printing \land left' = left \frown \langle ch? \rangle \land right' = right$$

We introduce the restrictions of the existence operator.

General rule—$(\exists\, d \mid p \bullet q) \Leftrightarrow (\exists\, d \bullet p \land q)$:

$$\exists\, left', right' : TEXT \bullet$$
$$ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

Determining Preconditions

We have:

$$\exists\, left', right' : TEXT \bullet$$
$$ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

Determining Preconditions

We have:

$$\exists\, left', right' : TEXT \bullet$$
$$ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

We remove the existence quantifier with the *One-point rule*

$$(\exists\, x : T \bullet x = e \land p) \Leftrightarrow p[e/x]$$

($p[e/x]$: Replace e by x in p)

Determining Preconditions

We have:

$$\exists\, left', right' : TEXT \bullet$$
$$ch? \in printing \land \#(left' \frown right') \leq maxsize \land$$
$$left' = left \frown \langle ch? \rangle \land right' = right$$

We remove the existence quantifier with the *One-point rule*

$$(\exists\, x : T \bullet x = e \land p) \Leftrightarrow p[e/x]$$

($p[e/x]$: Replace e by x in p) and get:

$$ch? \in printing \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$$

where we used $left' = left \frown \langle ch? \rangle$ and $right' = right$

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]
$\quad \Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$ [Restricted $\exists$]

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]
$\quad \Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$ [Restricted $\exists$]
$\quad \Leftrightarrow pr \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$ [One-point rule]

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]
$\quad \Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$ [Restricted $\exists$]
$\quad \Leftrightarrow pr \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$ [One-point rule]
$\quad \Leftrightarrow pr \land \#left + \#\langle ch? \rangle + \#right \leq maxsize$
$[\#(s \frown t) = \#s + \#t]$

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]
$\quad \Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$ [Restricted $\exists$]
$\quad \Leftrightarrow pr \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$ [One-point rule]
$\quad \Leftrightarrow pr \land \#left + \#\langle ch? \rangle + \#right \leq maxsize$
$[\#(s \frown t) = \#s + \#t]$
$\quad \Leftrightarrow pr \land \#left + 1 + \#right \leq maxsize$ $[\#\langle x \rangle = 1]$

Determining Preconditions

All together ($pr \equiv ch? \in printing$):

$\exists\, Editor' \bullet Insert$ [Definition of the preconditions]
$\quad \Leftrightarrow left', right' : TEXT \mid \ldots \bullet \ldots$ [Expanding schemas]
$\quad \Leftrightarrow left', right' : TEXT \bullet \ldots \land \ldots$ [Restricted $\exists$]
$\quad \Leftrightarrow pr \land \#(left \frown \langle ch? \rangle \frown right) \leq maxsize$ [One-point rule]
$\quad \Leftrightarrow pr \land \#left + \#\langle ch? \rangle + \#right \leq maxsize$
$[\#(s \frown t) = \#s + \#t]$
$\quad \Leftrightarrow pr \land \#left + 1 + \#right \leq maxsize$ $[\#\langle x \rangle = 1]$
$\quad \Leftrightarrow pr \land \#left + \#right < maxsize$ [Arithmetic]

Complete precondition:

$$ch? \in printing \land \#left + \#right < maxsize$$

Case Study: Expert System

Facts and rules:

$[FACT]$

$$stripes, fur, zebra, sharp\_teeth, carnivore, herbivore, mammal, tiger : FACT$$

Case Study: Expert System

Facts and rules:

$[FACT]$

$$stripes, fur, zebra, sharp\_teeth, carnivore, herbivore, mammal, tiger : FACT$$

$$rules : \mathbb{P}\, FACT \leftrightarrow FACT$$

$$\begin{aligned} rules = \{ & \\ & \vdots \\ & \{fur\} \mapsto mammal, \\ & \{sharp\_teeth\} \mapsto carnivore, \\ & \{stripes, mammal, carnivore\} \mapsto tiger, \\ & \{stripes, mammal, herbivore\} \mapsto zebra, \\ & \vdots \\ & \} \end{aligned}$$

Case Study: Expert System

Monotonic Reasoning—Facts are added to the knowledge base

$$deduce == (\lambda\, facts : \mathbb{P}\, FACT \bullet facts \cup rules(\!|\, \mathbb{P}\, facts \,|\!))$$

Example:

$$facts = \{stripes, sharp\_teeth, fur\}$$
$$deduce(facts) = \{stripes, sharp\_teeth, fur, mammal, carnivore\}$$
$$deduce(deduce(facts)) = \{stripes, sharp\_teeth, fur, mammal, carnivore, tiger\}$$

Case Study: Expert System

Monotonic Reasoning—Facts are added to the knowledge base

$$deduce == (\lambda\, facts : \mathbb{P}\, FACT \bullet facts \cup rules(\!|\, \mathbb{P}\, facts \,|\!))$$

Example:

$facts = \{stripes, sharp\_teeth, fur\}$
$deduce(facts) = \{stripes, sharp\_teeth, fur, mammal, carnivore\}$
$deduce(deduce(facts)) = \{stripes, sharp\_teeth, fur, mammal, carnivore, tiger\}$

Case Study: Expert System

Transitive closure ($^+$)—Function is applied until a fixpoint is reached.

$$complete == deduce^+$$

such that

$complete(facts)$
$= deduce(\ldots deduce(facts)\ldots)$
$= \{stripes, sharp\_teeth, fur, mammal, carnivore, tiger\}$

Case Study: Expert System

To get *consistent* rules, we define *mutually inconsistent* facts.

$inconsistent == \{$
$\vdots$
$\{fur, feathers, scales, \ldots\}$
$\{mammal, bird, fish, \ldots\}$
$\{tiger, zebra, ostrich, goldfish, \ldots\}$
$\vdots$
$\}$

Case Study: Expert System

A consistent set of facts contains no more than one element from each set of mutually exclusive alternatives.

$$consistent : \mathbb{P}(\mathbb{P}\, FACT)$$

$$\forall\, facts : consistent;\ mutually\_exclusive : inconsistent \bullet \\ \#(mutually\_exclusive \cap facts) \leq 1$$

Hereby we restrict *complete*:

$$\forall\, facts : consistent \bullet complete(facts) \in consistent$$

Different: *From a consistent set of facts, we can only deduce consistent facts.*

Case Study: Expert System

The core of the rule-based system is the *Deduce* schema:

Deduce

$$facts, facts' : \mathbb{P}\, FACT$$
$$data, goals, conclusions! : \mathbb{P}\, FACT$$

$$(\textbf{let}\ all == complete(facts \cup data) \bullet \\ facts \subseteq facts' \subseteq all \land \\ conclusions! = (goals \cap all) \subseteq facts')$$

facts, *facts′*: Facts (before and after)
data: New Facts
goals: Set of goals
conclusions!: Conclusions (Output)

Case Study: Expert System

Forward Chaining (data-driven) = Deduce from observations.

Deduce

$$facts, facts' : \mathbb{P}\, FACT$$
$$data, goals, conclusions! : \mathbb{P}\, FACT$$

$$(\textbf{let}\ all == complete(facts \cup data) \bullet \\ facts \subseteq facts' \subseteq all \land \\ conclusions! = (goals \cap all) \subseteq facts')$$

Forward

Deduce
$$observations? : \mathbb{P}\, FACT$$

$$data = observations?$$

$$observations? = \{stripes, sharp\_teeth, fur\} \land \\ goals = \{zebra, tiger, ostrich \ldots\}$$

Case Study: Expert System

Forward Chaining (data-driven) = Deduce from observations.

Deduce

$facts, facts' : \mathbb{P}\, FACT$
$data, goals, conclusions! : \mathbb{P}\, FACT$

$(\textbf{let}\ all == complete(facts \cup data) \bullet$
$\quad facts \subseteq facts' \subseteq all \land$
$\quad conclusions! = (goals \cap all) \subseteq facts')$

Forward

Deduce
$observations? : \mathbb{P}\, FACT$

$data = observations?$

$observations? = \{stripes, sharp\_teeth, fur\} \land$
$goals = \{zebra, tiger, ostrich \ldots\} \Rightarrow conclusions! = \{tiger\}$

70

Case Study: Expert System

Backward Chaining (goal-driven) = Which are the queries that match the facts?

Deduce

$facts, facts' : \mathbb{P}\, FACT$
$data, goals, conclusions! : \mathbb{P}\, FACT$

$(\textbf{let}\ all == complete(facts \cup data) \bullet$
$\quad facts \subseteq facts' \subseteq all \land$
$\quad conclusions! = (goals \cap all) \subseteq facts')$

Backward

Deduce
$queries? : \mathbb{P}\, FACT$

$goals = queries?$

$facts = \{stripes, sharp\_teeth, fur\} \land data = \varnothing \land$
$queries? = \{tiger, zebra\}$

71

Case Study: Expert System

Backward Chaining (goal-driven) = Which are the queries that match the facts?

Deduce

$facts, facts' : \mathbb{P}\, FACT$
$data, goals, conclusions! : \mathbb{P}\, FACT$

$(\textbf{let}\ all == complete(facts \cup data) \bullet$
$\quad facts \subseteq facts' \subseteq all \land$
$\quad conclusions! = (goals \cap all) \subseteq facts')$

Backward

Deduce
$queries? : \mathbb{P}\, FACT$

$goals = queries?$

$facts = \{stripes, sharp\_teeth, fur\} \land data = \varnothing \land$
$queries? = \{tiger, zebra\} \Rightarrow conclusions! = \{tiger\}$

72

Case Study: Expert System

The *Deduce* schema suggests a naive implementation.

1. Generate *all* possible facts (*all*),
2. determine all facts that appear in *goal*,
3. and return them as (*conclusions*!).

However, for a real implementation you do not have to proceed like this (and, actually you should not!)

A specification only describes the effects, not the procedure.

Alternative: *Executable specification* (slow, but useful—for prototypes or oracles)

Summary

- The schema calculus allows inclusion, conjunction, and disjunction of schemas.
- Schemas are the basis for program proofs.
- Common program proof: *precondition calculation*.

Literature

The Way of Z (Jonathan Jacky)
– all described examples and tutorials

The Z Notation
(http://www.comlab.ox.ac.uk/archive/z.html)

The Z Glossary (ftp://ftp.comlab.ox.ac.uk/pub/Zforum/zglossary.ps.Z)

Fuzz Type Checker
(http://spivey.oriel.ox.ac.uk/mike/fuzz/)
– Type checker and LaTeX macros for Linux