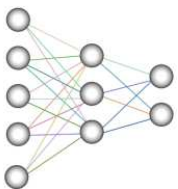


Z Tools

Watcharee Jumpamule, PhD.
Department of Computer Science
Chiangmai University

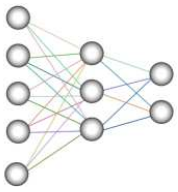
1



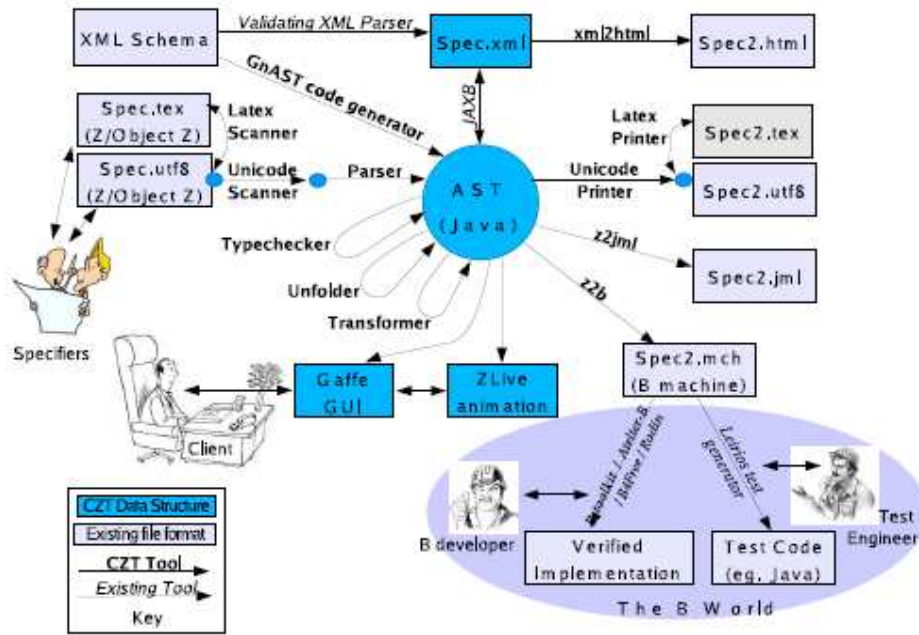
Outlines

- CZT <http://czt.sourceforge.net/>
- ZED <http://zed.c3po.it/>
- ZTC
- ZANS
- Etc.

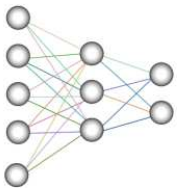
2



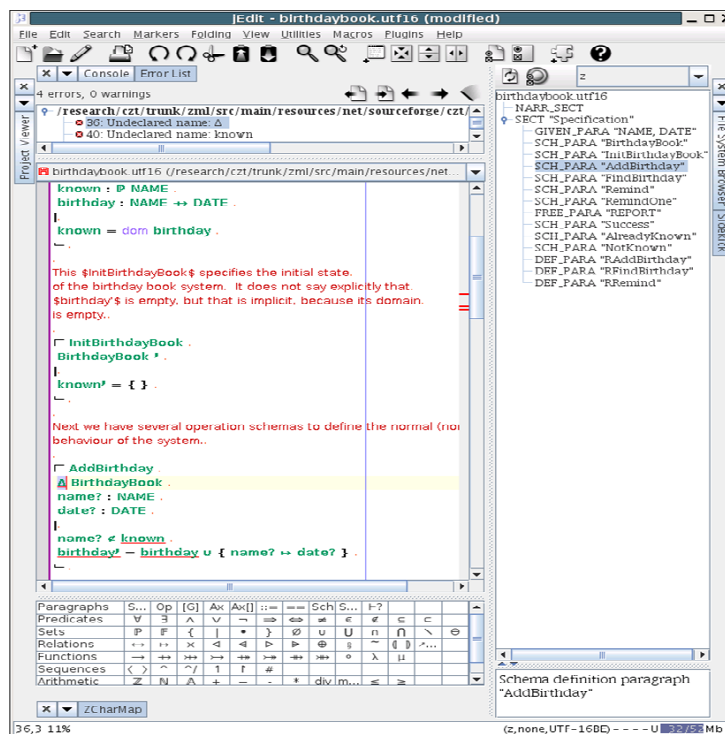
Overview of CZT Architecture

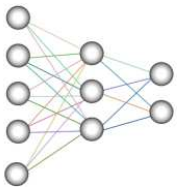


From CZT: A Framework for Z Tools, Petra Malik and Mark Utting, the University of Waikato, Hamilton, New Zealand



CZT plugged in JEdit

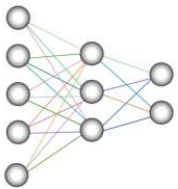




ZTC

- **A Type Checker for Z Notation**
- Platform
 - Microsoft Windows 9x, NT, 2000, and XP, and
 - Linux
- Input
 - LATEX
 - ZSL is as expressive of LATEX form

5



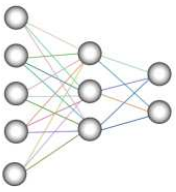
LATEX form

$[NAME, INFO]$

$dict : NAME \rightarrow INFO$
$defined : \mathbb{P} NAME$
$defined = \text{dom } dict$

```
\begin{spec}
\begin{zed}
  [ NAME, INFO ]
\end{zed}
\begin{schema}{DataDictionary}
  dict: NAME \pfun INFO \\
  defined: \power NAME
\where
  defined = \dom dict
\end{schema}
\end{spec}
```

6



ZSL

- ZSL has two style options: the *plain text* style and the *box* style. Using the ZSL plain text style, the Z specification above will be written as follows:

```

specification
[ NAME, INFO ]
schema
  DataDictionary
  dict : NAME +-> INFO;
  defined : P NAME
where
  defined = dom dict
end schema
end specification

```

```

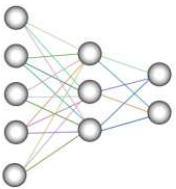
specification
[ NAME, INFO ]

--- DataDictionary -----
| dict      : NAME +-> INFO;
| defined   : P NAME
| -----
| defined = dom dict
| -----

end specification

```

7

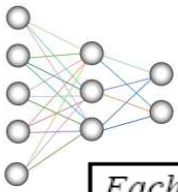


ZTC Features

- ZTC can translate a Z specification in ZSL to an equivalent one in LATEX, and *vice versa*.
- ZTC supports the following package-selection modes:
 - a) zed mode. Use the zed package. You can only use the commands defined in zed.
 - b) oz-zed compatible mode. Use the oz package. You can use all the commands defined in oz and zed. Incompatibilities are resolved in favor of zed.
 - c) oz native mode. Use the oz package. You can only use the commands defined in oz.

symbol	zed command	oz command
\emptyset	<code>\empty</code>	<code>\emptysetset</code>
$\hat{=}$	<code>\defs</code>	<code>\sdef</code>
<code>==</code>	<code>==</code>	<code>\defs</code>

8



Structure of Specification

Each specification must be enclosed by one of the following environments:

- `\begin{spec} ... \end{spec}`
- `\begin{document} ... \end{document}`

Everything outside these environments are ignored by ZTC.

A Z specification consists of formal and informal text.

ZTC will type check the formal text and ignore the informal text.

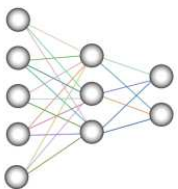
Formal text must be enclosed in one of the following formal environments:

- `\begin{axdef} ... \end{axdef}`
- `\begin{gendef} ... \end{gendef}`
- `\begin{schema} ... \end{schema}`
- `\begin{syntax} ... \end{syntax}`
- `\begin{zed} ... \end{zed}`

Anything outside these formal environments are considered informal text and ignored.

Specifically, formulae enclosed in \dots , $\backslash(\dots)$, and $\backslash[\dots]$ are considered informal text and ignored.

9



Formal Environments

- The `axdef` environment is used to define the *axiom boxes*.

```
\begin{axdef}
  MaxSize: \nat
\where
  MaxSize \leq 65535
\end{axdef}
```

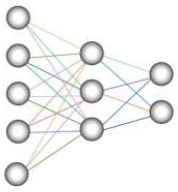
$MaxSize : \mathbb{N}$
$MaxSize \leq 65535$

- The `gendef` environment is used to define the *generic boxes*.

```
\begin{gendef}\{X,Y\}
  First: X \cross Y \fun X
\where
  \forall x: X; y: Y @ First(x,y) = x
\end{gendef}
```

$[X, Y]$
$First : X \times Y \rightarrow X$
$\forall x : X; y : Y \bullet First(x, y) = x$

10



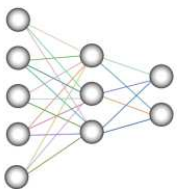
Formal Environments

- The schema environment is used to define the *schema boxes*

```
\begin{schema}{InsertOk}
\Delta DataDictionary \
  name? : NAME \
  info? : INFO \
  resp! : Response
\where
  name? \notin defined \
\# defined < MaxSize \
  dict' = dict \cup \{
  name? \mapsto info? \} \
  resp! = Success
\end{schema}
```

<i>InsertOk</i>
Δ DataDictionary
<i>name?</i> : NAME
<i>info?</i> : INFO
<i>resp!</i> : Response
<i>name?</i> \notin defined
$\#$ defined < MaxSize
<i>dict'</i> = dict \cup { <i>name?</i> \mapsto <i>info?</i> }
<i>resp!</i> = Success

11



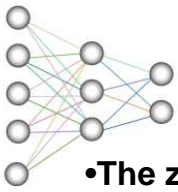
Formal Environments

- The syntax environment is used to define *free types*.
- A syntax environment contains a sequence of syntax rules separated by the `\also` command.

```
\begin{syntax}
  OP & ::= & plus | minus | times | divide
\also
  EXP & ::= & const \ldata \nat \rdata \
  & | & binop \ldata OP \cross
  EXP \cross EXP \rdata
\end{syntax}
```

$OP ::= plus minus times divide$
$EXP ::= const\langle\mathbb{N}\rangle$
$ binop\langle OP \times EXP \times EXP \rangle$

12



Formal Environments

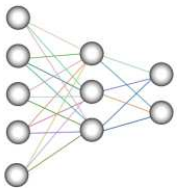
- The zed environment is used to define other paragraphs in Z, including *given sets*, *schema definitions*, *equivalence definitions*, and environment.
- A zed environment may contain several paragraphs. The paragraphs *predicates*. Short free type definitions can also be included in the zed in a zed environment must be separated by the `also` command

```

\begin{zed}
  [ADDR, PAGE]
\also
  DataDictInit \defs [ DataDictionary' | \
                    DataDictInit \hat{=} [DataDictionary' |
\text{3 defined' = \emptyset ]
                    DATABASE == ADDR \to PAGE
\also
  DATABASE == ADDR \fun PAGE
                    \exists n : NAME • birthday(n) \in December
\also
  REPORT ::= ok | unknown \langle\langle NAME \rangle\rangle
\exists n : NAME @ birthday(n) \in December
\also
  REPORT ::= ok | unknown \ldata NAME \rdata
\end{zed}

```

13



Formal Environments

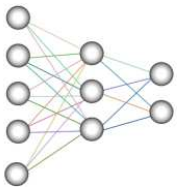
- The formal environments may not be nested.
- Note! You may use either syntax and zed environments to define free types, but they not inter-changable.
- Use the syntax environment for the vertical format, and use the zed environment for the horizontal format.

Informal comments or remarks inside formal environments can be introduced as follows:

- `\comm{ informal text };`
- `\remark{ informal text };`

ZTC ignores the arguments of these two commands.

14

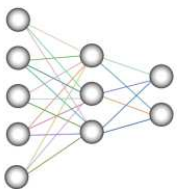


Line continuing command

A line continuing command is a linebreaking command followed by a TAB command, which is one of the following:
 $\backslash t0, \backslash t1, \backslash t2, \backslash t3, \backslash t4, \backslash t5, \backslash t6, \backslash t7, \backslash t8, \backslash t9.$
 Line continuing commands are treated as white spaces by ZTC.

- $\backslash t1$ indents the least and $\backslash t9$ the most amount of space.
- The TAB command is not only necessary for ZTC to perform type-checking properly, but also desirable for enhancing the readability of specifications. The following example shows the proper use of the line continuing command.

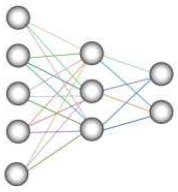
```
\begin{gendef}{X,Y}
  First: X \cross Y \fun X
\where
  \forall x: X; y: Y @ \\\
\backslash t1 First(x,y) = x
\end{gendef}
```

$$\begin{array}{l} [X, Y] \\ \hline First : X \times Y \rightarrow X \\ \hline \forall x : X; y : Y \bullet \\ \quad First(x, y) = x \end{array}$$


Running ZTC

- ZTC supports a number of command line options.

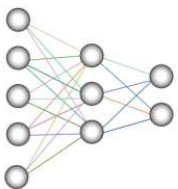
$$ztc [-l [tl]] infile [options]$$
- The input file name is required. It is no longer necessary to specify the input form when the full file name is given.
- ZTC will determine the input form automatically. You can still specify the input form explicitly with the -l switch, then the extension of the input file name may be omitted:
 - If LATEX form, the default extension is zed.
 - If ZSL form, the default extension is zsl.



ZANS

- A Z Animation System
- Xiaoping Jia, DePaul University, Chicago, Illinois, U.S.A
- ZANS is an animation tool for Z specifications.
- Its goals are:
 - Facilitate validation of Z specifications
 - Experiment design refinement and code synthesis based on Z specifications
 - Assist learning of the Z specification language

17



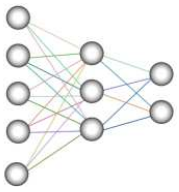
ZANS Features

It supports

- Type checking of Z specifications
- Expansion of schema expressions
- Evaluation of expression and predicates
- Execution of operation schemas

The input can be written in LATEX with zed or oz packages, or in ZSL---an ASCII version of Z

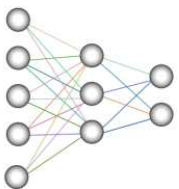
18



Command and Modes

- To invoke ZANS, type `ZANS` on command line
- All ZANS commands can be displayed by
 - `Zans> help`
- To exit ZANS, type the `exit` command.
- ZANS command format
 - Command-name [option] [arguments]*
 - `load classman`
 - `execute -t Enrol`

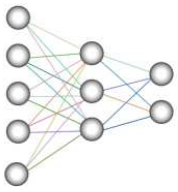
19



ZANS Operating modes

- ZANS has two operating modes : the *initial* mode and the *animation* mode.
- There are a pair of commands that will change the operating mode from one to the other:
 - `animate`, to switch from initial to animation
 - `clear`, to switch from animate to initial
- The two different modes are indicated by different command prompt
 - `zans>` indicates the initial mode
 - `anim>` indicates the animation mode

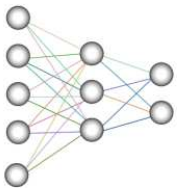
20



Evaluating expression and predicates

- The `eval` command allow you to evaluate any Z expression interactively.
- It is a multi-line command so that you can type in expressions that won't fit in one line. You must terminate a multi-line command with two consecutive return key

21



Function Definitions

- Define functions by using λ -expression

```
zans> para AddOne : \nat \fun \nat  
cont>
```

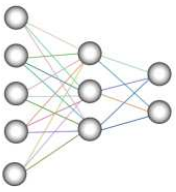
```
zans> assign AddOne:=(\lamda x: \nat @ x+1 )
```

```
lamda x : N @ x + 1
```

- Now you can apply the function to an expression.

```
zans> eval AddOne 2  
cont>  
3
```

22



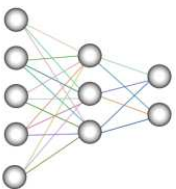
Animation

- Animations is to execute the operation schema in your specification to see if the specification accurately captures the requirements
- Preparation for animating
 1. Indicate which schema is the state schema, and which is the initialization schema by adding the following two line into the specification


```
%% state-schema <state schema name>
%% init-schema <initialization schema name>
```
 2. Optionally, you may also want to indicate which ones are user level operation


```
%% operation Enquire
%% operation Enrol
```

23



Animation

3. If you use global names, you need to assign a specific value for the animation to be carried out. For instance, size in the Class Manager's Assistant example, append the following paragraph at the end of the Class Manager's Assistant specification

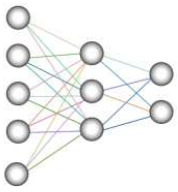
size = 6

4. Make implied predicate explicit.

ClassInit \cong [*Class* ' | *enrolled*' = \emptyset]

ClassInit \cong [*Class* ' | *enrolled*' = \emptyset \wedge *tested*' = \emptyset]

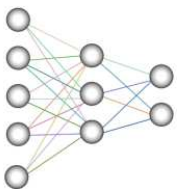
24



Loading Specification

- ZANS has extensive capabilities to manipulate schemas and schema expressions. Usually, you edit and save a specification in a file and type check it using ZTC.
- When the specification correctly typed, it is ready for animation.
 - `load`
 - `list`

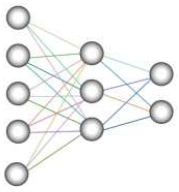
25



Creating and Running Scripts

- ZANS allow you to save commands in a script file.
- A script file is simply an ASCII file that contains ZANS commands.
- Each line contains a single command. A multi-line command must be followed by a blank line.
- It can be created using a text editor or `script` command in ZANS
- To record a script file, you first specify the script file name as follows:
 - `script scriptfile`
 - `script -on` to turn the recording on
 - `script -off` to turn the recording off

26



Run a script file

- `source scriptfile`
- `zans -f scriptfile`