

# The Great Book of Zebra

The Zebra Project

October 2, 2015

# Preface

This book is a collaborative work from the <https://github.com/fsvieira/zebrajs> project community and everyone is invited to participate.

The list of contributors is at the contributors section 1.3 and your name can be there too :D.

This is a work in progress.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	The Zebra-machine (ZM) . . . . .	3
1.1.1	ZM Language ( $\mathbb{L}$ ) . . . . .	3
1.1.2	ZM Operations . . . . .	4
1.1.3	ZM Computation . . . . .	5
1.2	Computing Examples (implementation) . . . . .	6
1.2.1	Defining a not equal . . . . .	6
1.2.2	Defining a list . . . . .	8
1.3	Contributors . . . . .	9

# Chapter 1

## Introduction

This is the official book of Zebra-machine (ZM). Here you will find anything you need to understand in deep the ZM, the book covers both theoretical and practical definitions.

Zebra-machine (ZM) is a logical symbolic computation query system, given a set of computational definitions it will answer questions about them, therefore ZM is better suited for software validation and constrain satisfaction problems.

### 1.1 The Zebra-machine (ZM)

As mentioned before ZM is a logical symbolic computation query system, and it consists of two parts the definitions and the query, both parts share the same language of ZM terms, which is defined by a certain formal syntax, and a set of transformation rules.

#### 1.1.1 ZM Language ( $\mathbb{L}$ )

**The ZM language ( $\mathbb{L}$ ) of terms are defined as:**

1.  $\mathbb{L}_0$  and  $\mathbb{L}_{not}$  are sets of ZM terms.
2.  $c \in \mathbb{C}$ :  $\mathbb{C}$  is the set of terminal symbols called constants,  $c$  is a terminal symbol.  $c \in \mathbb{L}_0$ .
3. ' $p \in \mathbb{V}$ ':  $\mathbb{V}$  is the set of variables,  $p$  is a variable. ' $p \in \mathbb{L}_0$ '.
4.  $(p_0 \dots p_n) \in \mathbb{T}$ :  $\mathbb{T}$  is the set of tuples,  $(p_0 \dots p_n)$  its a n-tuple of ZM terms.  $(p_0 \dots p_n) \in \mathbb{L}_0$ .
5.  $\bar{p}$  can be any term that is not equal to  $p$ ,  $p \in \mathbb{L}_0$ .  $\bar{p} \in \mathbb{L}_{not}$ .
6. Nothing else is a ZM term.
7.  $\mathbb{L} = \mathbb{L}_0 \cup \mathbb{L}_{not}$ ,

### 1.1.2 ZM Operations

The powerset of set  $\mathbb{S}$  is expressed as  $P(\mathbb{S})$  .

**Unification ( $\otimes$ , binary operation)** defined as

$$\otimes : \mathbb{L} \times \mathbb{L} \rightarrow \mathbb{L}$$

and by the rules,

$$\otimes : \mathbb{L}_0 \times \mathbb{L}_0 \rightarrow \mathbb{L}_0$$

$$1. Q \otimes Q = Q$$

$Q$  unifies with itself, resulting on itself.

$$2. 'p \otimes Q = Q \iff 'p = Q$$

$$3. Q \otimes 'p = Q \iff 'p = Q$$

note that if  $Q \in \mathbb{V}$  then rule [2] will also apply, since both  $p$  and  $Q$  are the same the result is also the same.

$$4. (p_0 \dots p_n) \otimes (q_0 \dots q_n) \iff (p_0 \otimes q_0 \dots p_n \otimes q_n)$$

$(p_0 \dots p_n)$  z-tuple only unifies with other z-tuple if they have same size and all sub ZM terms unify.

$$\otimes : \mathbb{L}_0 \times \mathbb{L}_{not} \rightarrow \mathbb{L}_0$$

$$1. Q \otimes \bar{p} = Q \iff p \neq Q$$

$$\otimes : \mathbb{L}_{not} \times \mathbb{L}_0 \rightarrow \mathbb{L}_0$$

$$1. \bar{p} \otimes Q = Q \iff p \neq Q$$

$$\otimes : \mathbb{L}_{not} \times \mathbb{L}_{not} \rightarrow \mathbb{L}_0$$

$$1. \bar{P} \otimes \bar{Q} = 'p \iff 'p \otimes \bar{P} \wedge 'p \otimes \bar{Q}$$

where  $'p$  is a new variable.

note that if  $P = Q$ , then  $'p \neq (P \otimes Q)$ .

Anything else is not unifiable.

**Substitution ( $\mathcal{S}$ , function)** defined as:

$$\mathcal{S} : \mathbb{V} \times \mathbb{L} \times \mathbb{L} \rightarrow \mathbb{L} \quad (1.1)$$

$$\mathcal{S}(v, w, t) = \begin{cases} w & , \text{ if } t = v, \\ (\mathcal{S}(v, w, t_0) \dots \mathcal{S}(v, w, t_n)) & , \text{ if } t = (t_0 \dots t_n) \\ t & , \text{ otherwise} \end{cases} \quad (1.2)$$

**The notation  $\mathbb{T}_p$**  is used to define the set containing all of  $p$  inner tuples.

### 1.1.3 ZM Computation

**A ZM computation is expressed as 4-tuple  $(\sigma, \delta, q, \alpha)$  where:**

1.  $\sigma$  is a set of terminal symbols (constants),
2.  $\delta$  is a set of z-tuples (definitions),
3.  $q$  is a z-tuple (query),
4.  $\alpha$  is the set of possible computational answers to query  $q$  based on *delta* definitions.

**A definition** is a fact in the system. The inner tuples of a definition are considered and called queries, therefor for a definition to be true all of its inner tuples/queries must also be true.

**A query** is a question to the system that is true if and only if it unifies at least with one definition.

**Free and bound variables** on the context of a definition all definition variables are considered to be bound to the definition, on the context of queries all variables are free.

Bound variables will be expressed as:

$$([v_0 \dots v_n] p_0 \dots p_k)$$

where  $v_i, 0 \leq i \leq n$  are the tuple bound variables and  $p_j, 0 \leq j \leq k$  are the tuple terms.

ex:

$$([a \ b \ c] ('a \ 'b) 'c \ \text{yellow})$$

Bound information and notation is used on intermediate computation steps, therefor they are not included on the *ZM* language.

**Variable renaming** its necessary to ensure that distinct variables with same name are not handled as being the same.

ex:

$$([p] \text{ 'p 'q}) \otimes ([p \text{ q}] \text{ 'p 'q})$$

The first tuple has only 'p as bound variable and the second tuple as 'p and 'q declared as bound variables. To ensure that this variables keep their meaning, we rename the bound variables like this:

$$([p_0] \text{ 'p}_0 \text{ 'q}) \otimes ([p_1 \text{ q}_1] \text{ 'p}_1 \text{ 'q}_1)$$

After unification we get

$$([p_0 \text{ p}_1 \text{ q}_1] \text{ 'p}_0 \otimes \text{ 'p}_1 \text{ 'q} \otimes \text{ 'q}_1)$$

**A computation** is done with the following steps:

1.  $\delta_0 = \{p_0 \otimes \dots \otimes p_{|x|} : \forall x \in P(\delta) \wedge \forall p_i \in x, i = 0 \dots |x|\}$
2.  $\delta_1 = \{([v_0 \dots v_n] p_0 p_k) : \forall (p_0 \dots p_n) \in \delta_0\}$   
 where  $v_0 \dots v_n$  are all (bound) variables found on  $(p_0 \dots p_n)$  tuple.  
 $\delta_1$  is the set of definitions with bound variables.
3.  $\alpha = \{q \otimes p : \forall p \in \delta_1 \wedge \forall t \in \mathcal{T}_{q \otimes q} \rightarrow (\sigma, \delta, t, \alpha)\}$   
 the  $\alpha$  set is the result of the unification of the query  $q$  with all definitions  $p$  and the resulting inner tuples are also a valid computation.  
 a computational will stop if the set of inner tuples are empty.  
 a computational may not stop for some cases.

## 1.2 Computing Examples (implementation)

### 1.2.1 Defining a not equal

$$(\text{notEqual } \text{'p '}\overline{p})$$

Ex:

$$(\sigma, \delta, q, \alpha) = (\{\text{yellow, blue}\}, \{(\text{notEqual } \text{'p '}\overline{p})\}, q, \alpha)$$

1.  $q = (\text{notEqual } \text{'x 'x})$   
 $(\text{notEqual } \text{'x 'x}) \otimes ([p] \text{ notEqual } \text{'p '}\overline{p})$   
 $([p] \text{ notEqual} \otimes \text{notEqual } \text{'x} \otimes \text{'p 'x} \otimes \text{'}\overline{p})$   
 unify constants,  $([p] \text{ notEqual } \text{'x} \otimes \text{'p 'x} \otimes \text{'}\overline{p})$ ,  
 replace bound variables  $[p]$  by their unifying part, and dispose the variable.

in this case  $'p \rightarrow 'x$ ,  $(\text{notEqual } 'x'x \otimes \overline{'x})$ ,  
variable  $x$  can't be equal and not equal to  $x$  at same time,  
query fails to unify with any of the definitions,  
 $\alpha = \emptyset$ , no answers exist for this query.

2.  $q = (\text{notEqual yellow yellow})$   
 $(\text{notEqual yellow yellow}) \otimes ([p] \text{ notEqual } 'p \overline{'p})$   
 $([p] \text{ notEqual} \otimes \text{notEqual yellow} \otimes 'p \text{ yellow} \otimes \overline{'p})$   
unify constants,  $([p] \text{ notEqual yellow} \otimes 'p \text{ yellow} \otimes \overline{'p})$ ,  
replace bound variables  $[p]$  by their unifying part, and dispose the  
variable.

in this case  $'p \rightarrow \text{yellow}$ ,  $(\text{notEqual yellow yellow} \otimes \overline{\text{yellow}})$ ,  
yellow can't unify with not yellow,  
query fails to unify with any of the definitions,  
 $\alpha = \emptyset$ , no answers exist for this query.

3.  $q = (\text{notEqual blue yellow})$   
 $(\text{notEqual blue yellow}) \otimes ([p] \text{ notEqual } 'p \overline{'p})$   
 $([p] \text{ notEqual} \otimes \text{notEqual blue} \otimes 'p \text{ yellow} \otimes \overline{'p})$   
unify constants,  $([p] \text{ notEqual blue} \otimes 'p \text{ yellow} \otimes \overline{'p})$ ,  
replace bound variables  $[p]$  by their unifying part, and dispose the  
variable.

in this case  $'p \rightarrow \text{blue}$ ,  $(\text{notEqual blueyellow} \otimes \overline{\text{blue}})$ ,  
yellow, will never be unifiable with blue, so it can be solved to  
 $(\text{notEqual blue yellow})$ ,  
 $\alpha = \{(\text{notEqual blue yellow})\}$ .

4.  $q = (\text{notEqual } 'p \text{ yellow})$   
 $(\text{notEqual } 'p \text{ yellow}) \otimes ([p] \text{ notEqual } 'p \overline{'p})$   
rename bound variable,  $p \rightarrow x$ ,  
 $(\text{notEqual } 'p \text{ yellow}) \otimes ([x] \text{ notEqual } 'x \overline{'x})$   
 $([x] \text{ notEqual} \otimes \text{notEqual } 'p \otimes 'x \text{ yellow} \otimes \overline{'x})$   
unify constants,  $([x] \text{ notEqual } 'p \otimes 'x \text{ yellow} \otimes \overline{'x})$   
replace bound variables  $[x]$  by their unifying part, and dispose the  
variable.

in this case  $'x \rightarrow 'p$ ,  $(\text{notEqual } 'p \text{ yellow} \otimes \overline{'p})$   
 $\alpha = \{(\text{notEqual } 'p \text{ yellow} \otimes \overline{'p})\}$ .



### 1.2.2 Defining a list

1. (list)
2. (list 'item (list))
3. (list 'item (list 'i 'tail))

Ex:

$$(\sigma, \delta, q, \alpha) = (\{\text{yellow}, \text{blue}\}, \{(\text{list}), (\text{list 'item (list)}), (\text{list 'item (list 'i 'tail)})\}, q, \alpha)$$

1.  $q = (\text{list})$ , query a empty list.

$$(\text{list}) \otimes (\text{list}) = (\text{list} \otimes \text{list}) = (\text{list})$$

all other definitions fail to unify because the number of elements in remaining definitions don't match query tuple.

$$\alpha = \{(\text{list})\}$$

2.  $q = (\text{list yellow (list blue (list)))}$

Starting with definition (list),

$$(\text{list yellow (list blue (list)))} \otimes (\text{list}), \text{ fail.}$$

Next definition (list 'item (list)),

$$(\text{list yellow (list blue (list)))} \otimes ([\text{item}] \text{ list 'item (list)})$$

$$([\text{item}] \text{ list} \otimes \text{list yellow} \otimes \text{'item (list blue (list))} \otimes (\text{list}))$$

fail to unify (*list*) with (list blue (list)).

Next definition (list 'item (list 'i 'tail)),

$$(\text{list yellow (list blue (list)))} \otimes ([\text{item i tail}] \text{ list 'item (list 'i 'tail)})$$

$$([\text{item i tail}] \text{ list} \otimes \text{list yellow} \otimes \text{'item (list blue (list))} \otimes (\text{list 'i 'tail}))$$

unify constants,

$$([\text{item i tail}] \text{ list yellow} \otimes \text{'item (list blue (list))} \otimes (\text{list 'i 'tail}))$$

$$([\text{item i tail}] \text{ list yellow} \otimes \text{'item (list} \otimes \text{list blue} \otimes \text{'i (list)} \otimes \text{'tail}))$$

unify constants,

$$([\text{item i tail}] \text{ list yellow} \otimes \text{'item (list blue} \otimes \text{'i (list)} \otimes \text{'tail}))$$

—

$$(\text{list yellow (list blue (list)))}[\text{bound : item i tail, constrains : 'item = yellow 'i = blue 'tail = (list)}]$$

the next step is to unify all sub-tuples that are not yet checked with definitions, this are: (list blue (list)) and (list),

$$(\text{list blue (list)}) \otimes (\text{list}), \text{ fail.}$$

$(\text{list blue (list)}) \otimes (\text{list 'item (list)})[\text{bound : item}]$   
 $(\text{list} \otimes \text{list blue} \otimes \text{'item (list)} \otimes (\text{list}))[\text{bound : item}]$   
 $(\text{listblue(list)})[\text{bound : item, constrains : 'item = blue}], \text{ succeed.}$   
 $(\text{list blue (list)}) \otimes (\text{list 'item (list 'i 'tail)}), \text{ fail.}$

Finally  $(\text{list})$  will only unify with  $(\text{list})$  definition, ending the process and resulting on  $\alpha = \{(\text{list yellow (list blue (list))})\}$

### 1.3 Contributors

- Filipe Vieira, <https://github.com/fsvieira>