The Great Book of Zebra

The Zebra Project

September 29, 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

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, therefor ZMis 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. $(p_0 \ldots p_n) \in \mathbb{T}$: $\mathbb{T}$ is the set of tuples, $(p_0 \ldots p_n)$ its a n-tuple of ZM terms. $(p_0 \ldots p_n) \in \mathbb{L}_0$.

5. $\overline{p}$ can be any term that is not equal to p, $p \in \mathbb{L}_0$. $\overline{p} \in \mathbb{L}_{not}$.

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

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

and by the rules,

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

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. $(p_0 \ldots p_n) \otimes (q_0 \ldots q_n) \iff (p_0 \otimes q_0 \ldots p_n \otimes q_n)$

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

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

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

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

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

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

1. $\overline{P} \otimes \overline{Q} = `p \iff `p \otimes \overline{P} \wedge `p \otimes \overline{Q}$

    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:

$$\mathcal{S} : \mathbb{V} \times \mathbb{L} \times \mathbb{L} \to \mathbb{L} \tag{1.1}$$

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

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. σ is a set of terminal symbols (constants),

2. δ is a set of z-tuples (definitions),

3. q is a z-tuple (query),

4. α 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:

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

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

ex:

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

Bound information and notation is used on intermidiate 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:

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

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:

$$([p_0] \text{ `}p_0 \text{ `}q) \otimes ([p_1 \ q_1] \text{ `}p_1 \text{ `}q_1)$$

After unification we get

$$([p_0 \ p_1 \ q_1] \text{ `}p_0 \otimes \text{ `}p_1 \text{ `}q \otimes \text{ `}q_1)$$

A computation is done with the following steps:

1. $\delta_1 = ([v_0 \ldots v_n] \ p_0 \ p_k) : \forall (p_0 \ldots p_n) \in \delta)$

   where $v_0 \ldots v_n$ are all (bound) variables found on $(p_0 \ldots p_n)$ tuple.

   δ_1 is the set of definitions with bound variables.

2. $\alpha = \{q \otimes p : \forall p \in \delta_1 \ \wedge \forall t \in \mathcal{T}_{q \otimes q} \to (\sigma, \delta, t, \alpha)\}$

   the α set is the result of the unification of the query q with all definitions p and the resulting innter 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

1.2.1 Defining a not equal

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

Ex:

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

1. $q = (\text{notEqual `}x \text{ `}x)$

   $(\text{notEqual `}x \text{ `}x) \otimes ([p] \ \text{notEqual `}p \ \overline{\text{`}p})$

   $([p] \ \text{notEqual} \otimes \text{notEqual `}x \otimes \text{`}p \text{ `}x \otimes \overline{\text{`}p})$

   $([p] \ \text{notEqual} \otimes \text{notEqual `}x \otimes \text{`}p \text{ `}x \otimes \overline{\text{`}p})$

   $([p] \ \text{notEqual `}x \text{ `}x), \text{constrains} : \text{`}x = \text{`}p \wedge \text{`}x \neq \text{`}p$

   variable x can't be equal and not equal to `p at same time,

   query fails to unify with any of the definitions,

   $\alpha = \emptyset$, no awnsers existe for this query.

2. $q = $ (notEqual yellow yellow)

   ($[p]$ notEqual yellow yellow) $\otimes$ (notEqual 'p $\overline{'p}$)

   ($[p]$ notEqual $\otimes$ notEqual yellow $\otimes$ 'p yellow $\otimes$ $\overline{'p}$)

   ($[p]$ notEqualyellowyellow), constrains : yellow $=$ 'p $\wedge$ yellow $\neq$ 'p

   variable p can't be equal and not equal to yellow vablue at same time,

   query fails to unify with any of the definitions,

   $\alpha = \emptyset$, no answers exist for this query.

3. $q = $ (notEqual blue yellow)

   ($[p]$ notEqual blue yellow) $\otimes$ (notEqual 'p $\overline{'p}$)

   ($[p]$ notEqual $\otimes$ notEqual blue $\otimes$ 'p yellow $\otimes$ $\overline{'p}$)

   ($[p]$ notEqualblueyellow), constrains : blue $=$ 'p $\wedge$ yellow $\neq$ 'p $\wedge$ blue $\neq$ yellow

   $\alpha = \{$(notEqual blue yellow)$\}$.

4. $q = $ (notEqual 'p yellow)

   ($[p]$ notEqual 'p yellow) $\otimes$ (notEqual 'p $\overline{'p}$)

   rename bound variable, $p \rightarrow x$,

   ($[p]$ notEqual 'p yellow) $\otimes$ (notEqual 'x $\overline{'x}$)

   ($[p]$ notEqual $\otimes$ notEqual 'p $\otimes$ 'x yellow $\otimes$ $\overline{'x}$)

   ($[p]$ notEqualpyellow), constrains : p $=$ 'x $\wedge$ yellow $\neq$ 'x

   $\alpha = \{$(notEqual 'p yellow), constrains : '$p \neq$ yellow$\}$.

1.2.2 Defining a list

1. (list)

2. (list '$item$ (list))

3. (list '$item$ (list 'i '$tail$))

   Ex:

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

1. $q = $ (list), query a empty list.

   (list) $\otimes$ (list) $=$ (list)

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

   $\alpha = $ (list)

2. $q = $ (list yellow (list blue (list)))

Starting with definition list,

(list yellow (list blue (list))) $\otimes$ (list), fail.

Next definition (list '*item* (list)),

(list yellow (list blue (list))) $\otimes$ (list '*item* (list))[bound : item]

(list $\otimes$ list yellow $\otimes$ '*item* (list blue (list)) $\otimes$ (list))[bound : item]

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

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

(list yellow (list blue (list))) $\otimes$ (list '*item* (list '*i* '*tail*))[bound : item i tail]

(list $\otimes$ list yellow $\otimes$ '*item* (list blue (list)) $\otimes$ (list '*i* '*tail*))[bound : item i tail]

(list yellow(list$\otimes$list blue$\otimes$'*i* (list)$\otimes$'*tail*))[bound : item i tail, constrains : '*item* = yellow]

(list yellow(list blue (list)))[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)),

(list blue (list)) $\otimes$ (*list*), fail.

(list blue (list)) $\otimes$ (list '*item* (list))[bound : *item*]

(list $\otimes$ list blue $\otimes$ '*item* (list) $\otimes$ (list))[bound : *item*]

(listblue(list)))[bound : *item*, constrains : '*item* = blue], succeed.

(list blue (list)) $\otimes$ (list '*item* (list '*i* '*tail*)), fail.

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

1.2.3   What john likes?

A simple example of querying facts.

We start with the definitions/facts:

1. (mary likes wine '*p*)

   mary likes wine.

2. (mary likes john '*p*)

   mary likes john.

3. (peter likes peter '*p*)

   peter likes himself.

4. (john likes '$stuff$ (mary likes '$stuff$ 'p))
   john likes everything that mary likes.

5. (john likes '$stuff$ (mary likes '$stuff$ 'p))
   john likes everything that mary likes.

6. (john likes '$person$ ('$person$ likes wine 'p))
   john likes anyone that likes wine.

7.

   (john likes '$person$ (list ('$person$ likes '$person$ 'p)(list (notEqual '$person$john) (list))))

   john likes anyone that likes themselves.

1.3   Contributors

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