fwo Research Foundation Flanders
Opening new horizons

Multiverse Debugging

Carmen Torres Lopez, Elisa Gonzalez Boix, Stefan Marr, Robbert Gurdeep Sigh, Christophe Scholliers

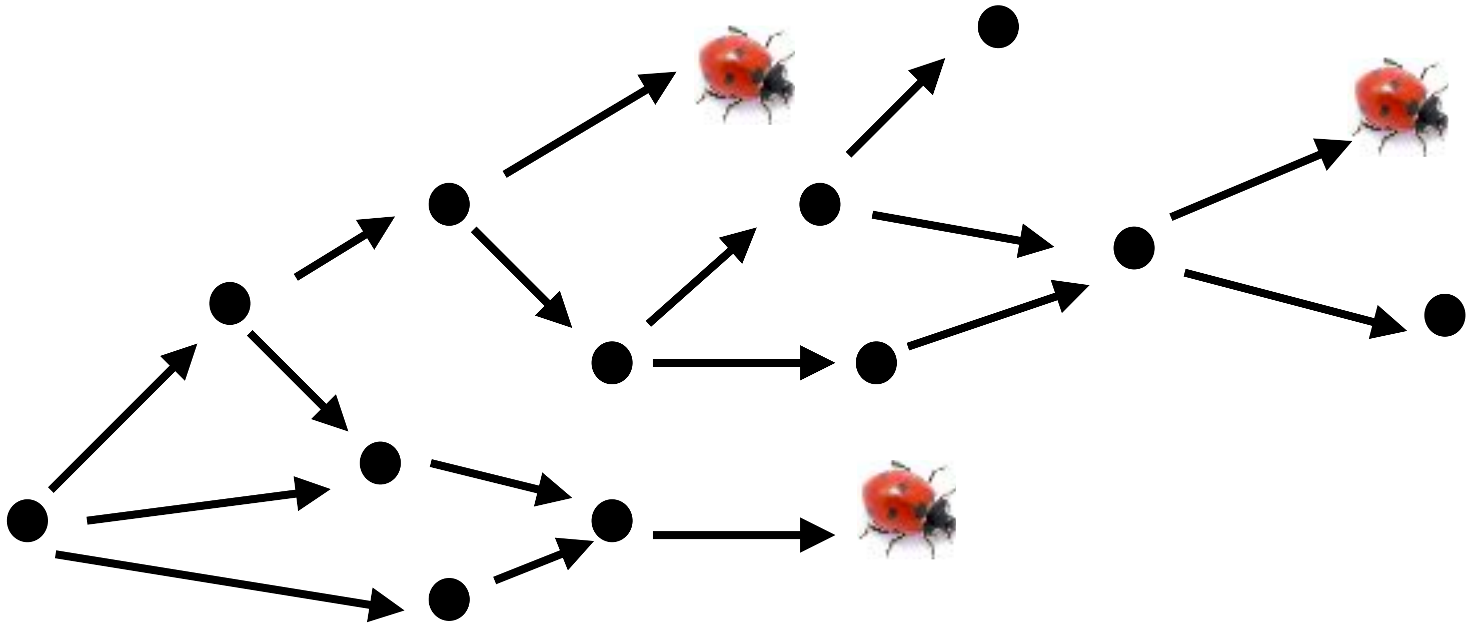
VUB VRIJE UNIVERSITEIT BRUSSEL

University of Kent

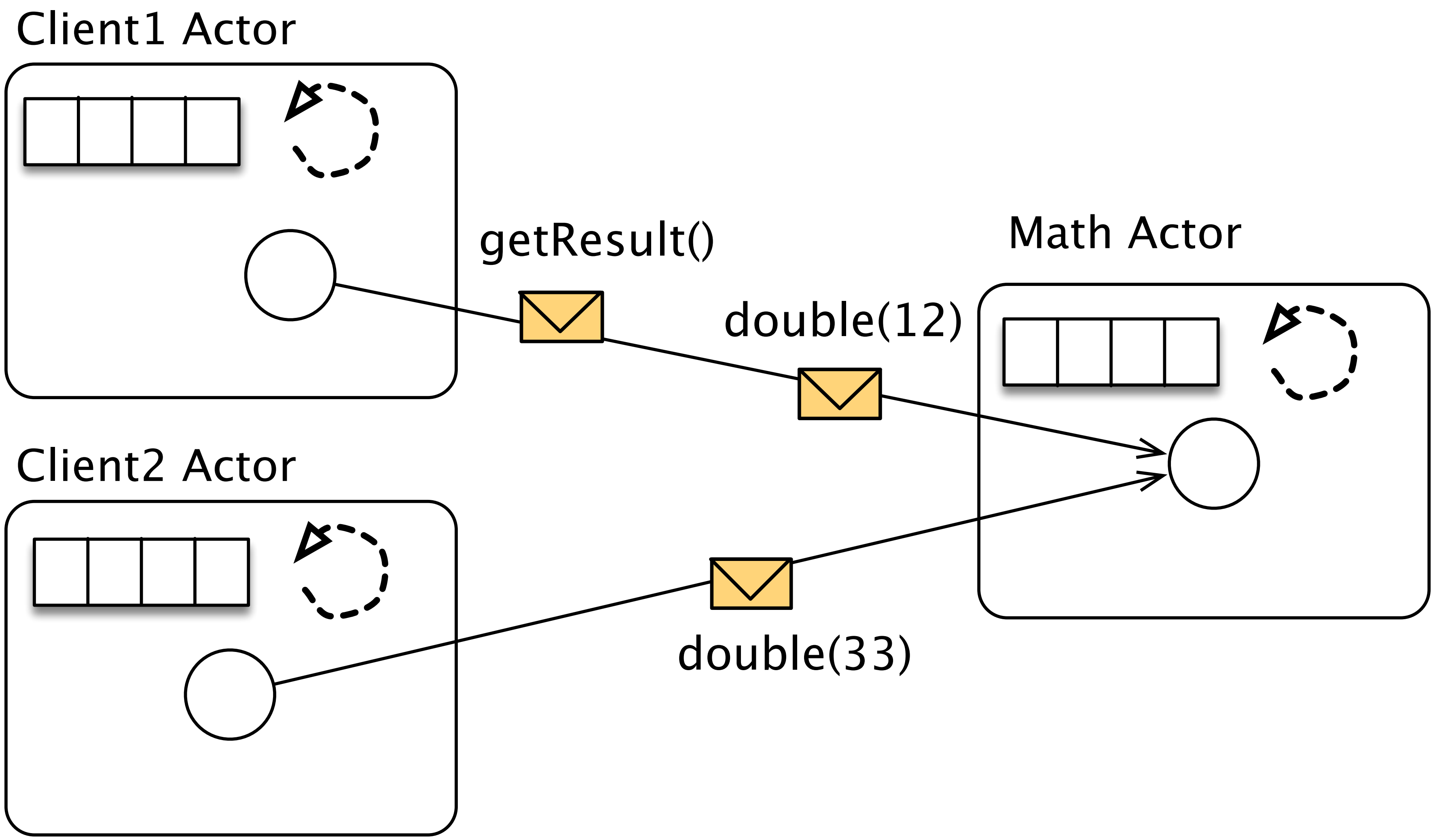
UNIVERSITEIT GENT

Goal

- To build a debugger for **non-deterministic programs** that
  - is probe-effect free, and
  - is able to explore the space of all possible bugs



Running Example



Running Example in AmbientTalk

```
def makeMath() {
    actor:{
        def result := 0;
        def double(x){result := x+x};
        def getResult(){result};
    }
};
def makeClient1(math){
    actor:{ |math|
        def start(){
                math<-double(12);
                when: math<-getResult()@FutureMessage becomes: {|res|
                   system.println(res);
                }}}
};
def makeClient2(math){
    actor: { |math|
        def start(){ math<-double(33) }}
};
def math := makeMath();
def client1 := makeClient1(math);
def client2 := makeClient2(math);
client1<-start();
client2<-start();
```

Bad Message Interleaving

```
def makeMath() {
    actor:{
        def result := 0;
        def double(x){result := x+x};
        def getResult(){result};
    }
};
```

Faulty Interleaving	Correct Interleaving	Correct Interleaving
client 1 - double(12)	client 1 - double(12)	client 2 - double(33)
client 2 - double(33)	client 1 - getResult() -> 24	client 1 - double(12)
client 1 - getResult() -> 66	client 2 - double(33)	client 1 - getResult() -> 24

time ↓

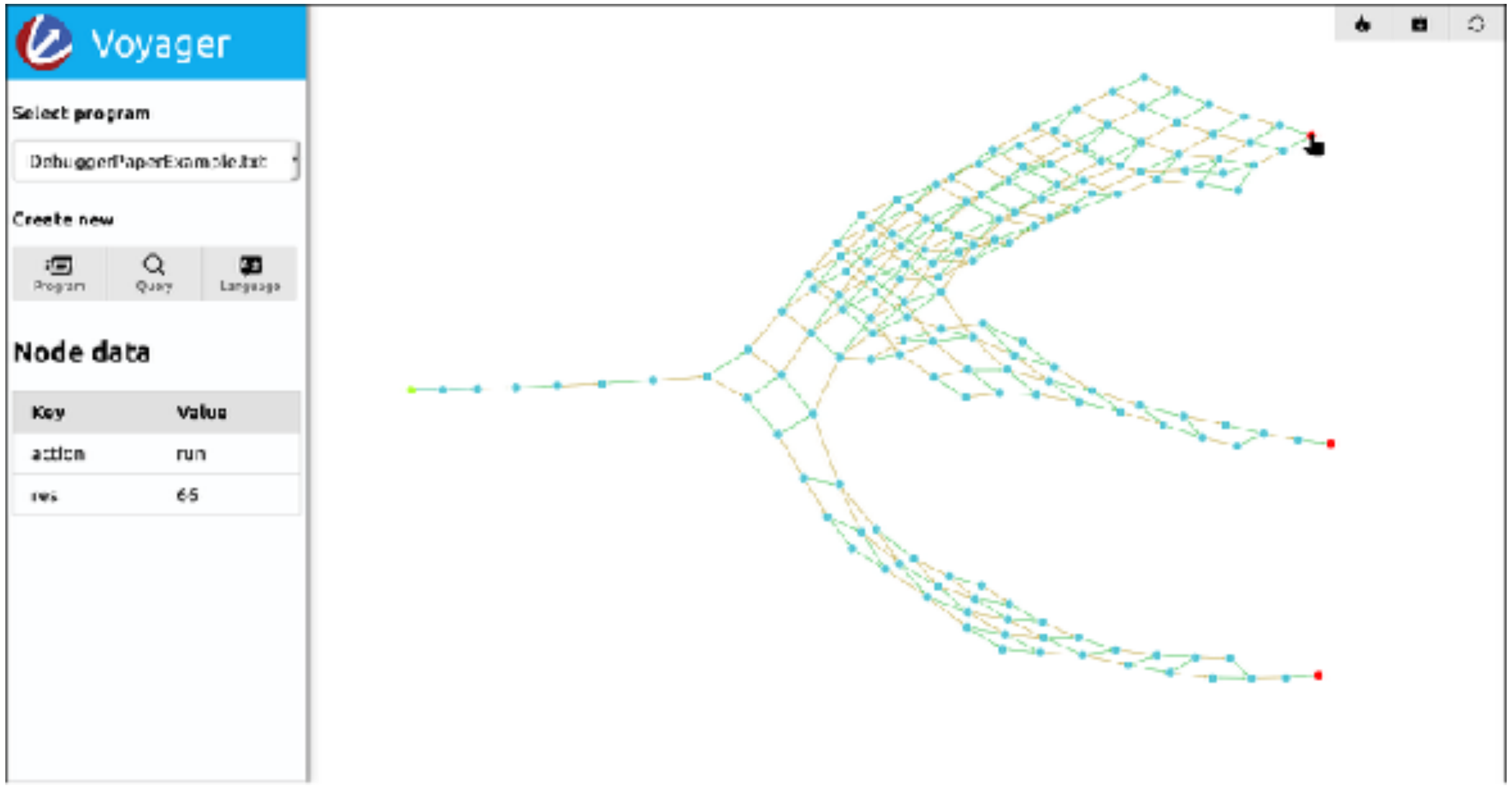
Carmen Torres Lopez, Stefan Marr, Elisa Gonzalez Boix, Hanspeter Mössenböck. A Study of Concurrency Bugs and Advanced Development Support for Actor-based Programs. Programming with Actors 2018: 155-185

Multiverse Debugging

Properties:

1. Observe *all* possible paths of the program execution

2. One step leads to a possible set of *universes*, i.e. paths of execution



Select program

DebuggerPaperExample.txt

Node data

Key	Value
action	run
res	65

Carmen Torres Lopez, Robbert Gurdeep Sigh, Stefan Marr, Elisa Gonzalez Boix, Christophe Scholliers. Multiverse Debugging: Non-deterministic Debugging for Non-deterministic Programs. To Appear in ECOOP 2019

Multiverse Debugging Recipe

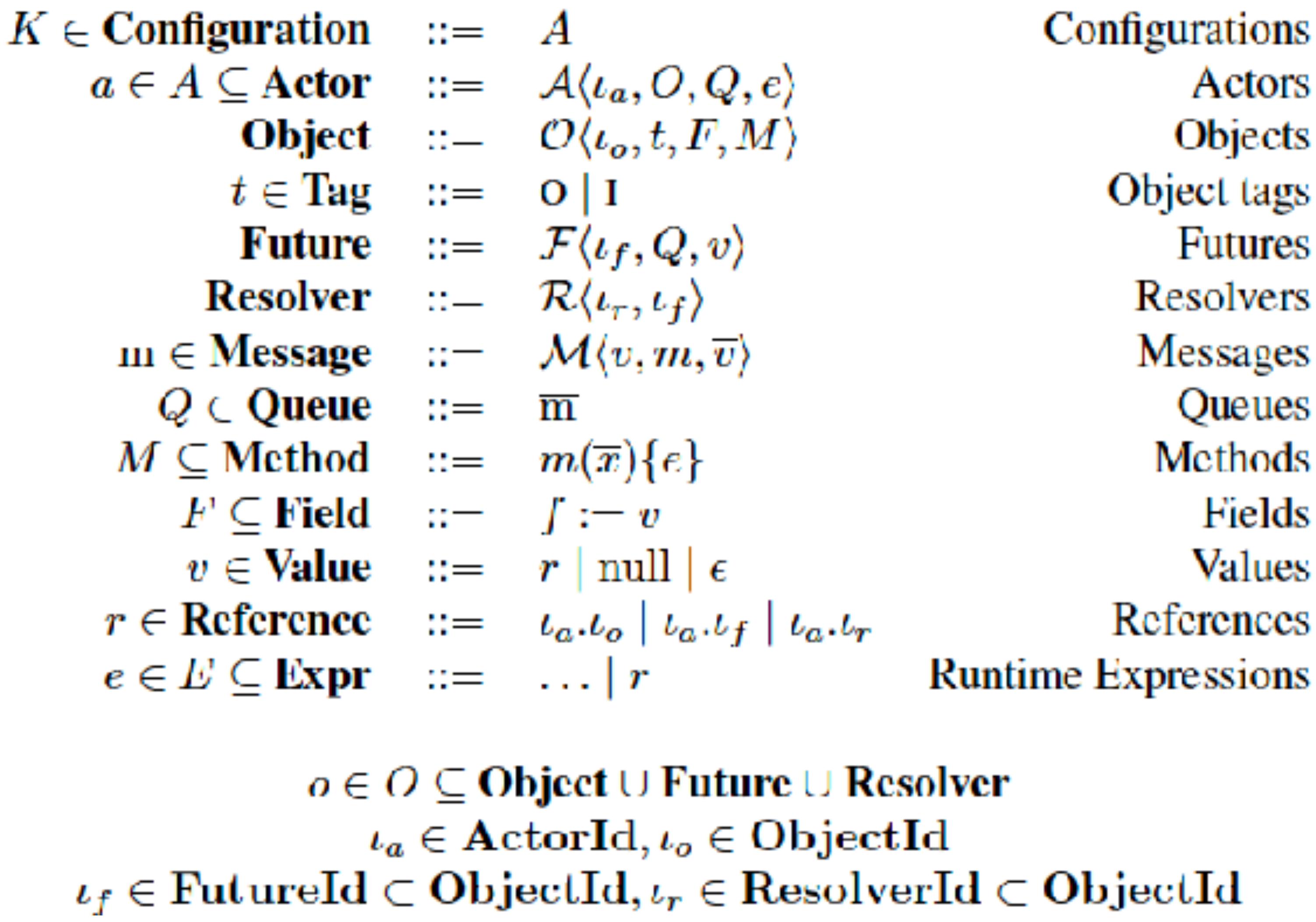
1. Operational semantics of the **non-deterministic base language**.

2. Operational semantics of the **debugger** in terms of the base-level semantics.

- **Configuration** that a debugger needs to keep track of in order to debug a base level program (including the base level semantics).
- **Debugging operations** of the debugger, in terms of the base-level semantics.

Step 1

Featherweight AmbientTalk



$K \in \mathbf{Configuration}$	$::=$	A	Configurations
$a \in A \subseteq \mathbf{Actor}$	$::=$	$\mathcal{A}\langle \iota_a, O, Q, e \rangle$	Actors
$\mathbf{Object}$	$::=$	$\mathcal{O}\langle \iota_o, t, F, M \rangle$	Objects
$t \in \mathbf{Tag}$	$::=$	$\textsc{o} \mid \textsc{i}$	Object tags
$\mathbf{Future}$	$::=$	$\mathcal{F}\langle \iota_f, Q, v \rangle$	Futures
$\mathbf{Resolver}$	$::=$	$\mathcal{R}\langle \iota_r, \iota_f \rangle$	Resolvers
$m \in \mathbf{Message}$	$::=$	$\mathcal{M}\langle v, m, \overline{v} \rangle$	Messages
$Q \subseteq \mathbf{Queue}$	$::=$	$\overline{m}$	Queues
$M \subseteq \mathbf{Method}$	$::=$	$m(\overline{x})\{e\}$	Methods
$F \subseteq \mathbf{Field}$	$::=$	$f := v$	Fields
$v \in \mathbf{Value}$	$::=$	$r \mid \mathtt{null} \mid \epsilon$	Values
$r \in \mathbf{Reference}$	$::=$	$\iota_a.\iota_o \mid \iota_a.\iota_f \mid \iota_a.\iota_r$	References
$e \in E \subseteq \mathbf{Expr}$	$::=$	$\ldots \mid r$	Runtime Expressions

$$o \in O \subseteq \mathbf{Object} \cup \mathbf{Future} \cup \mathbf{Resolver}$$

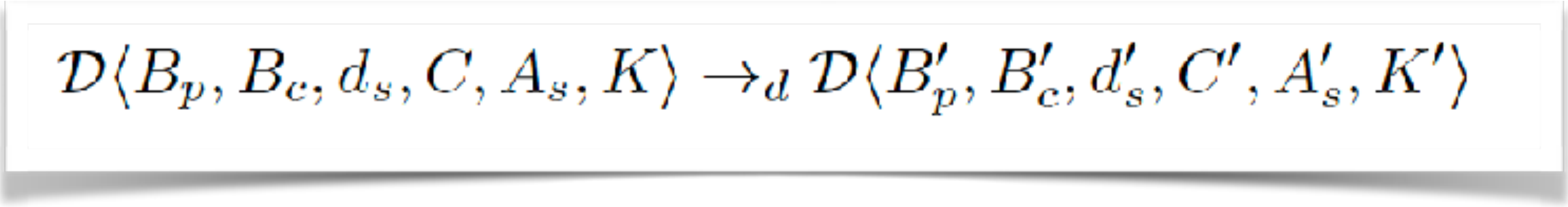
$$\iota_a \in \mathbf{ActorId}, \iota_o \in \mathbf{ObjectId}$$

$$\iota_f \in \mathbf{FutureId} \subset \mathbf{ObjectId}, \iota_r \in \mathbf{ResolverId} \subset \mathbf{ObjectId}$$

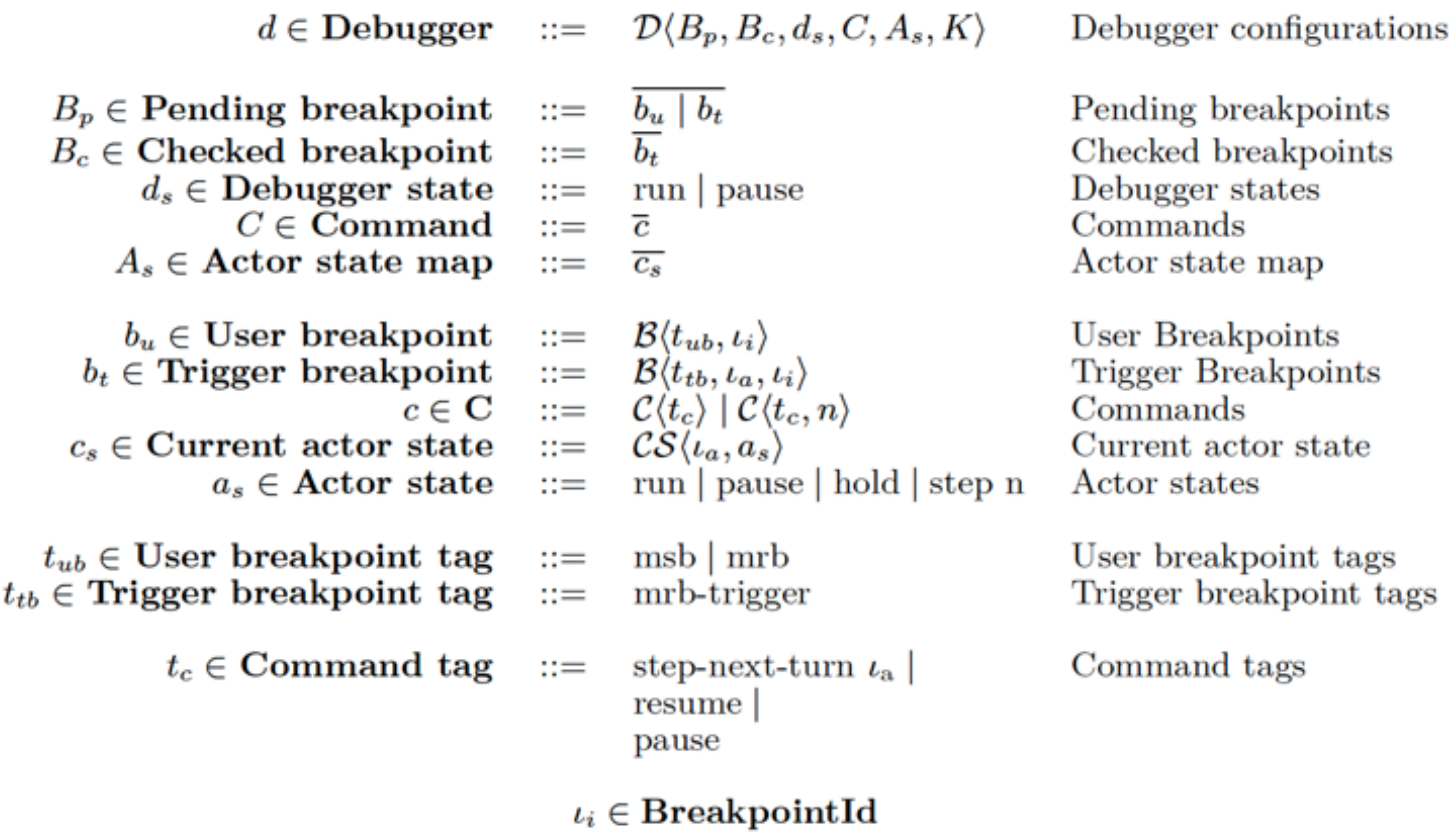
Van Cutsem T., Gonzalez Boix E., Scholliers C, Lombide Carreton A., Harnie D., Pinte K., and De Meuter W., 2014. *AmbientTalk: programming responsive mobile peer-to-peer applications with actors*. Computer Languages, Systems and Structures 40, 3–4 (2014), 112–136.

Step 2

Voyager



$$\mathcal{D}\langle B_p, B_c, d_s, C, A_s, K\rangle \rightarrow_d \mathcal{D}\langle B_p', B_c', d_s', C', A_s', K'\rangle$$



$d \in$ **Debugger**	::=	$\mathcal{D}\langle B_p, B_c, d_s, C, A_s, K\rangle$	Debugger configurations
$B_p \in$ **Pending breakpoint**	::=	$\overline{b_u \mid b_t}$	Pending breakpoints
$B_c \in$ **Checked breakpoint**	::=	$\overline{b_t}$	Checked breakpoints
$d_s \in$ **Debugger state**	::=	run | pause	Debugger states
$C \in$ **Command**	::=	$\overline{c}$	Commands
$A_s \in$ **Actor state map**	::=	$\overline{c_s}$	Actor state map
$b_u \in$ **User breakpoint**	::=	$\mathcal{B}\langle t_{ub}, \iota_i\rangle$	User Breakpoints
$b_t \in$ **Trigger breakpoint**	::=	$\mathcal{B}\langle t_{tb}, \iota_a, \iota_i\rangle$	Trigger Breakpoints
$c \in$ **C**	::=	$\mathcal{C}\langle t_c\rangle \mid \mathcal{C}\langle t_c, n\rangle$	Commands
$c_s \in$ **Current actor state**	::=	$\mathcal{CS}\langle \iota_a, a_s\rangle$	Current actor state
$a_s \in$ **Actor state**	::=	run | pause | hold | step n	Actor states
$t_{ub} \in$ **User breakpoint tag**	::=	msb | mrb	User breakpoint tags
$t_{tb} \in$ **Trigger breakpoint tag**	::=	mrb-trigger	Trigger breakpoint tags
$t_c \in$ **Command tag**	::=	step-next-turn ι_a | resume | pause	Command tags

$\iota_i \in$ **BreakpointId**

Running Example in Voyager

```
(
  ()                                    ; Pending breakpoints
  ()                                    ; Checked breakpoints
  run                                   ; Debugger state
  ()
  ()                                    ; Commands (user interaction)
  ((client1 run))                       ; Actor map
  ((actor                               ; Base language term
    client1
    ()
    ()
    (let (math (actor
         (field result 0)
         (method double x (set! (this $ result) (+ x x)))
         (method result p (this $ result))))
    in
    (let (client2 (actor
            (method start math (send math double (33) c2-double-to-math))))
    in
    (let (a (send client2 start (math) c1-start-to-c2)) in
    (let (b  (send math double (12) c1-double-to-math)) in
    (let (x_f x_r) future in
    (let (x_l (
          let (some-var 5) in
          (object (method apply x ((x_r $ resolve-mu) x))))
         ) in
    (let (var (send (let (x_f1 x_r1) future in
    (let (var (send math result (0 x_r1) c1-result-to-math)) in x_f1))
                        register-mu (x_l) c1-result-to-math)) in x_f)))))))
  ))
)
```

Demo

https://github.com/chscholl/GraphRedex/blob/artefact/ECOOP2019/Publications/ECOOP2019/README.md

Voyager available at https://redex.ugent.be/debugger.html

Conclusion

"Non-deterministic programs require non-deterministic tools"

- A new exploration path in debugging with many challenges to tackle:
    - state explosion
    - multiverse breakpoint & stepping operations
    - novel visualisations