Practical Multiverse Debugging through User-defined Reductions:
Application to UML Models

Matthias Pasquier[1–2], Ciprian Teodorov[2], Frédéric Jouault[3], Matthias Brun[3], Luka Le Roux[2], Loïc Lagadec[2]

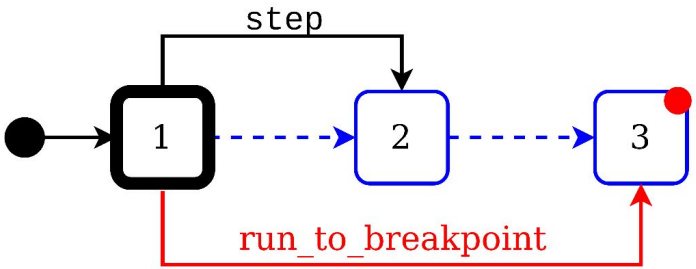
[1] ERTOSGENER - Angers, France
[2] Lab-STICC CNRS UMR 6285 ENSTA Bretagne - Brest, France
[3] ESEO - Angers, France

Models 2022 - Montréal, Canada - October 23-28 2022

Context: Debugging Specification Languages

- Debugging is useful
  - Useful early in the development lifecycle to fix problems as soon as possible
- We need to adapt debuggers to specification languages specificities
  - A specification language is non-deterministic
- Non-deterministic debugging, or multiverse debugging allows us to debug such systems, but face a scalability issue when searching for breakpoints
- We propose to extend multiverse debugging by allowing the user to define reduction policies, creating under-approximations during the breakpoint search

Evolution of debugging: Step debugging



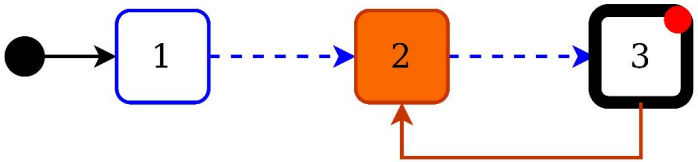
Debug Actions:

- ➔ step
- ➔ run_to_breakpoint

```
x := 1;
x := 2;
●x := 3;
```

Evolution of debugging: Step debugging

```
x := 1;
x := 2;
●x := 3;
```

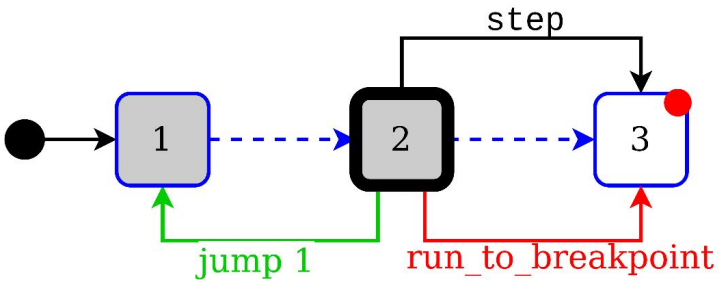


Debug Actions:

- ➔ step
- ➔ run_to_breakpoint

Evolution of debugging: Omniscient debugging

```
x := 1;
x := 2;
●x := 3;
```



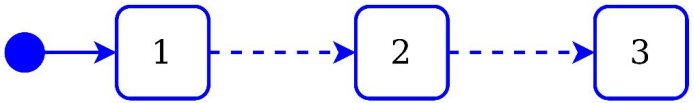
Debug Actions:

- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

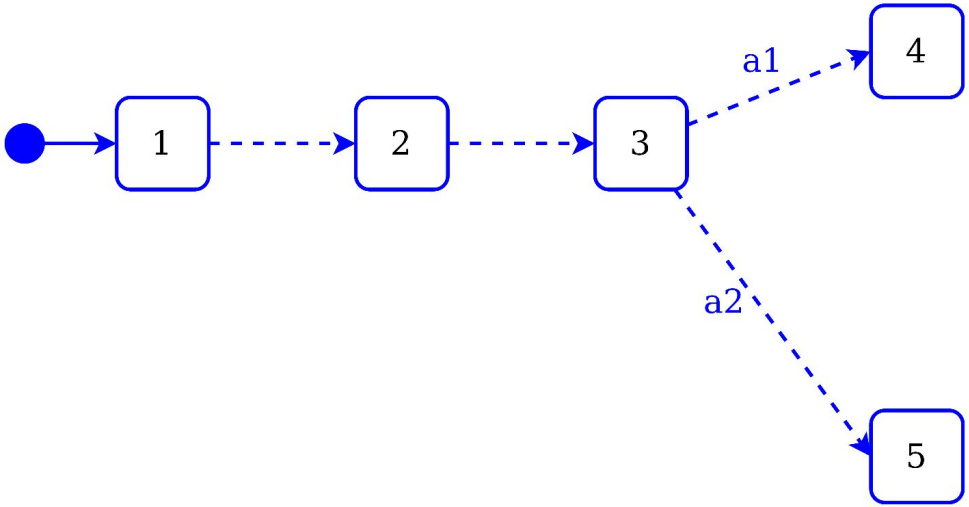
Bil Lewis. Computing Research Repository 2003. *“Debugging Backwards in Time.”*

Specification Language:
How to handle non determinism ?

```
x := 1;
x := 2;
x := 3;
```

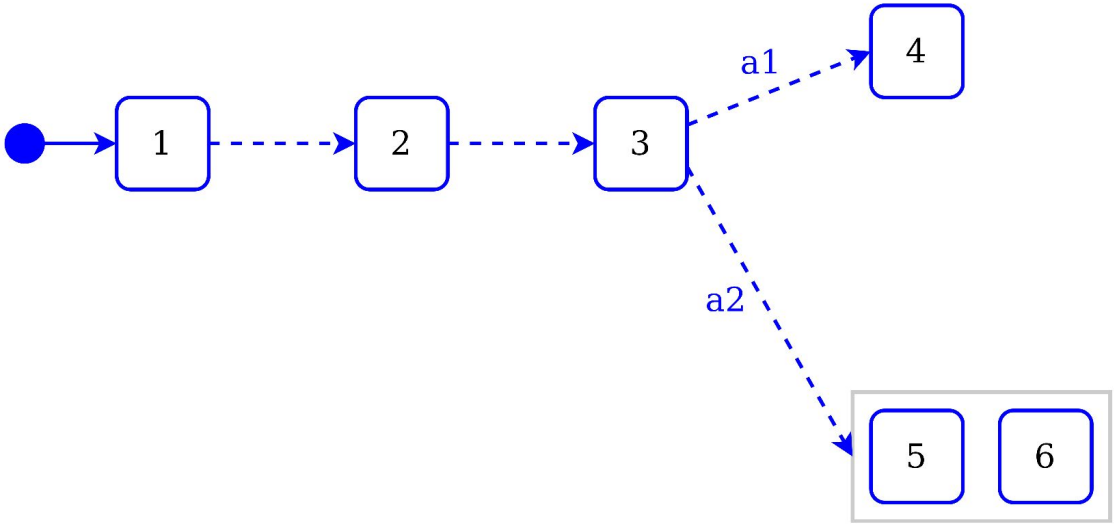


Specification Language:
How to handle non determinism ?



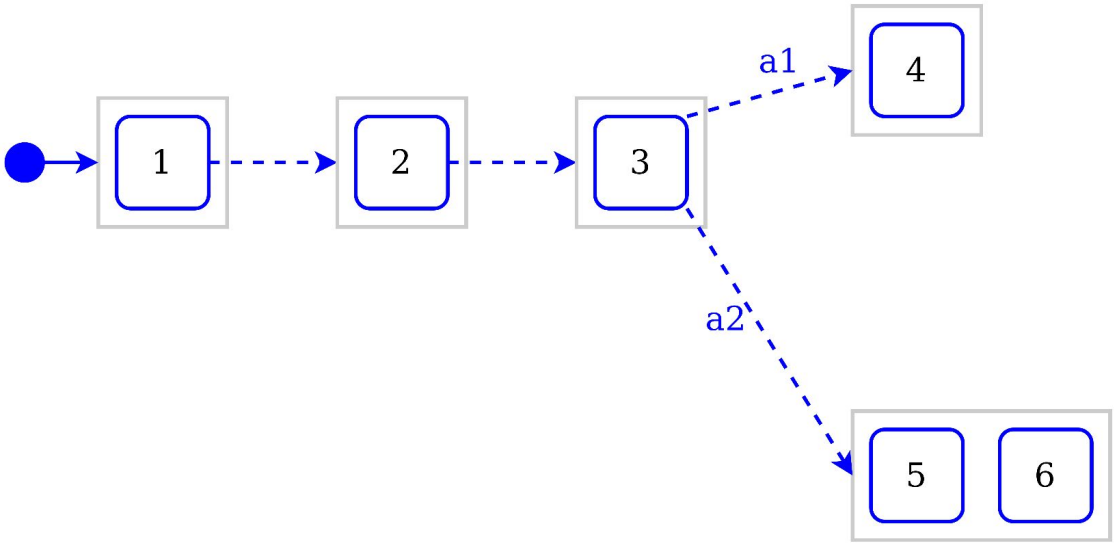

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
    }
    a2 -> {
        x := 5;
    }
}
```

Specification Language:
How to handle non determinism ?



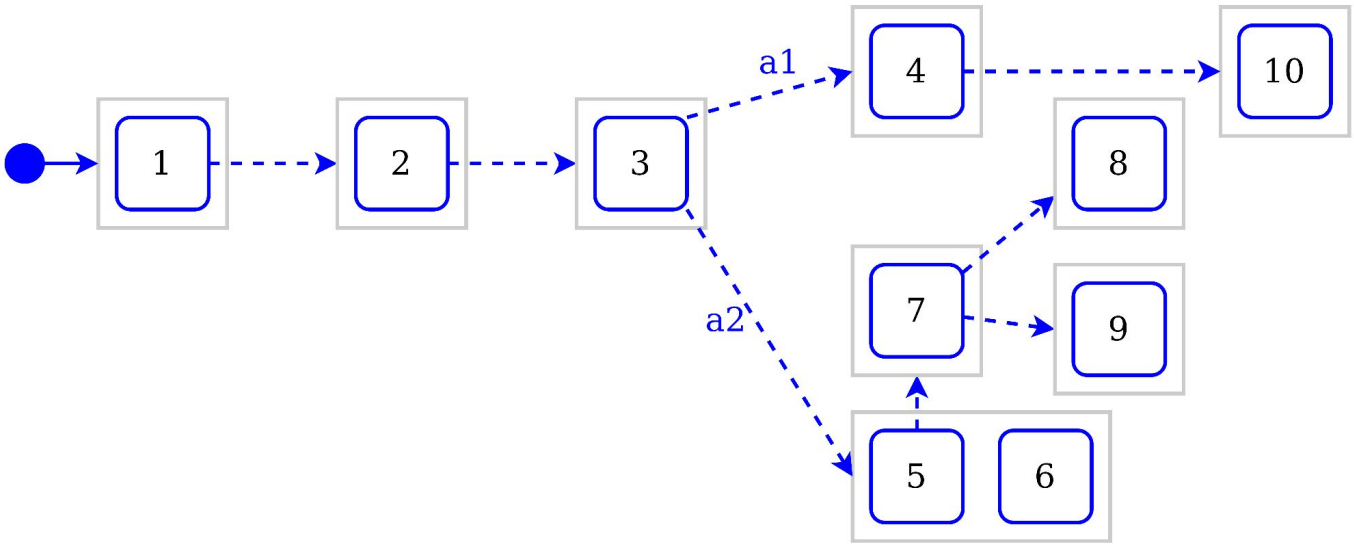

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
    }
    a2 -> {
        x := any of {5,6};
    }
}
```

Specification Language:
How to handle non determinism ?



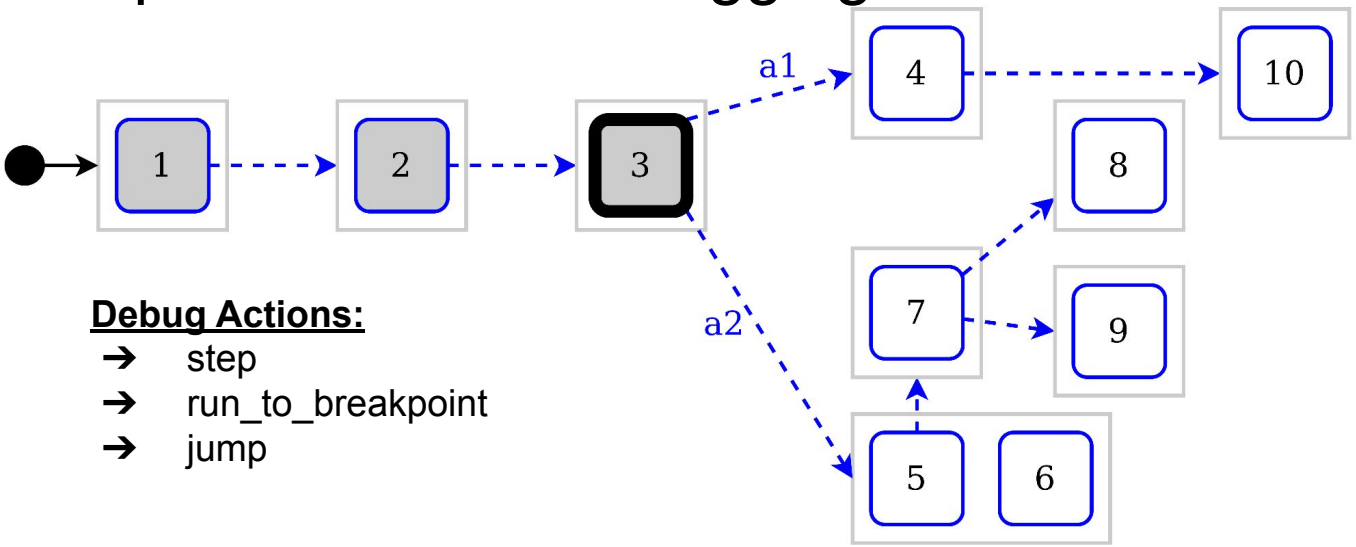

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
    }
    a2 -> {
        x := any of {5,6};
    }
}
```

Specification Language:
How to handle non determinism ?




```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle non-determinism requirements for debugging?



Debug Actions:

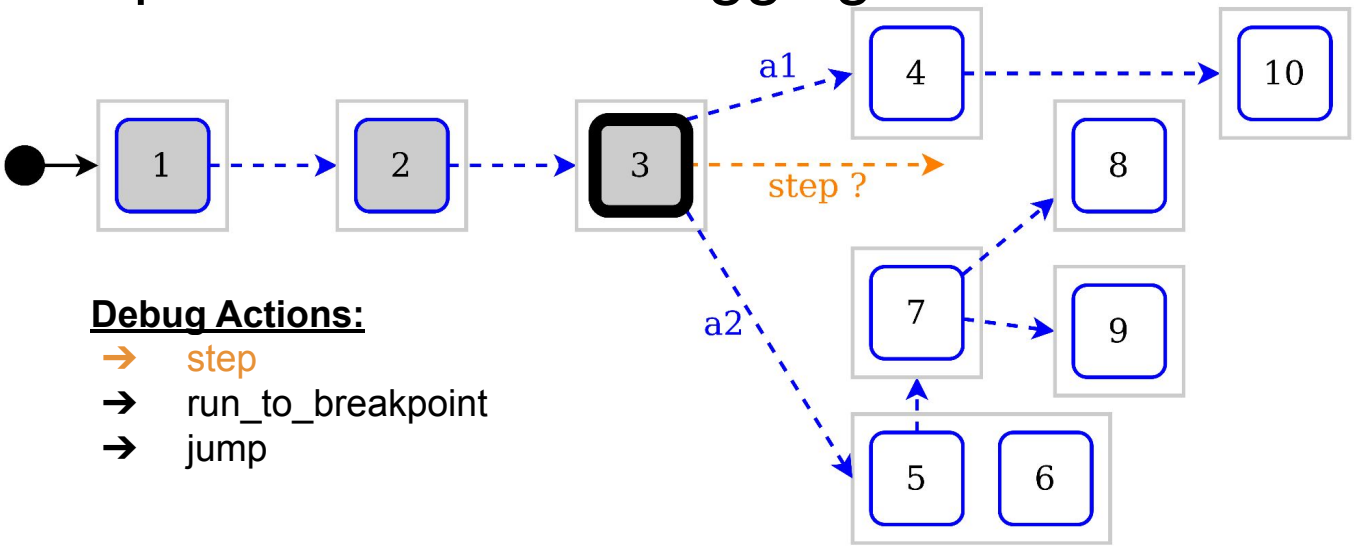
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```


Carmen Torres et Al. 33rd European Conference on OOP 2019. *"Multiverse Debugging: Non-deterministic Debugging for Non-deterministic Programs"*

Valentin Besnard. Ph. D. Dissertation 2020. *“EMI: An approach to unify analysis and embedded execution with a controllable model interpreter.”*

How to handle non-determinism requirements for debugging?



Debug Actions:

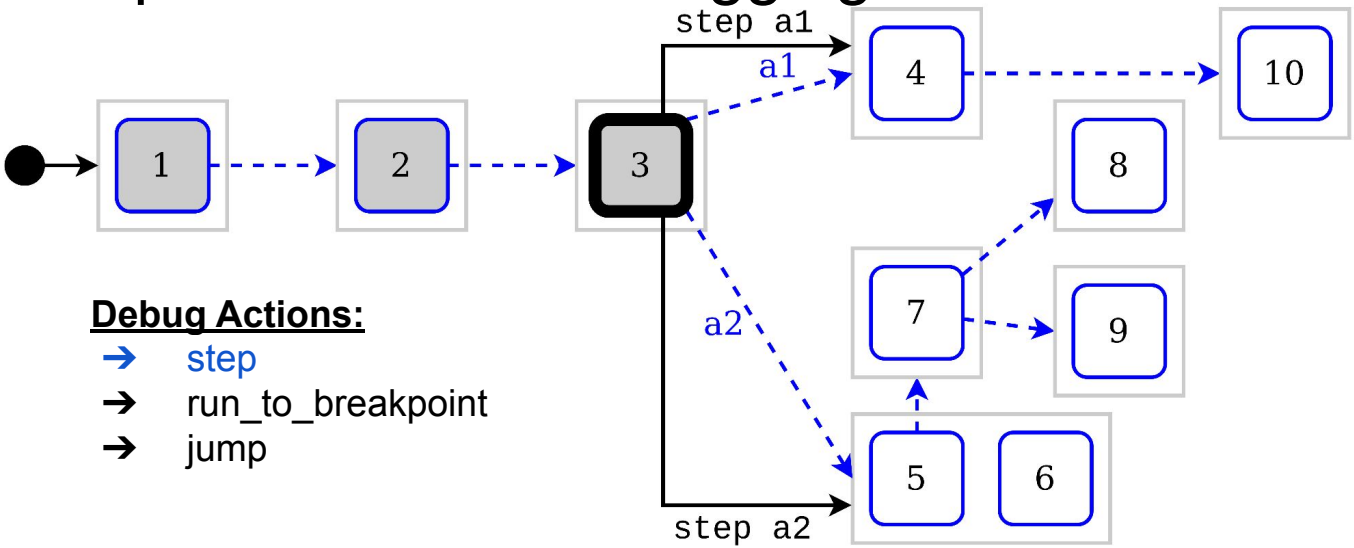
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

Solving problems with non determinism:

- ➔ step: Navigating first case of non determinism
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle non-determinism requirements for debugging?



Debug Actions:

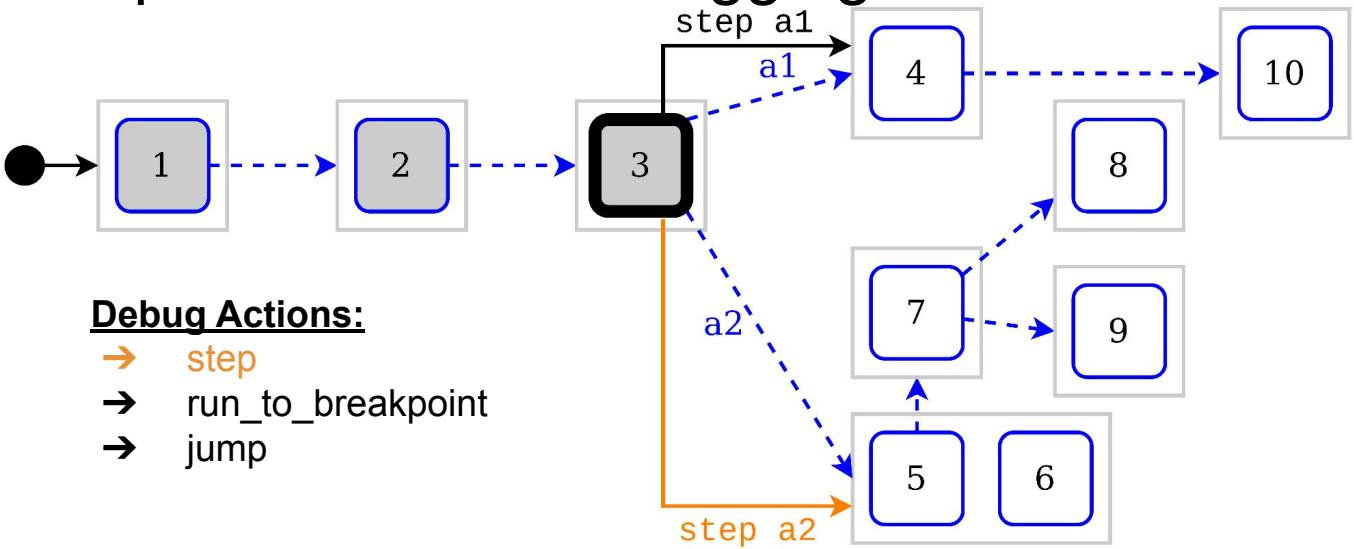
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle non-determinism requirements for debugging?



Debug Actions:

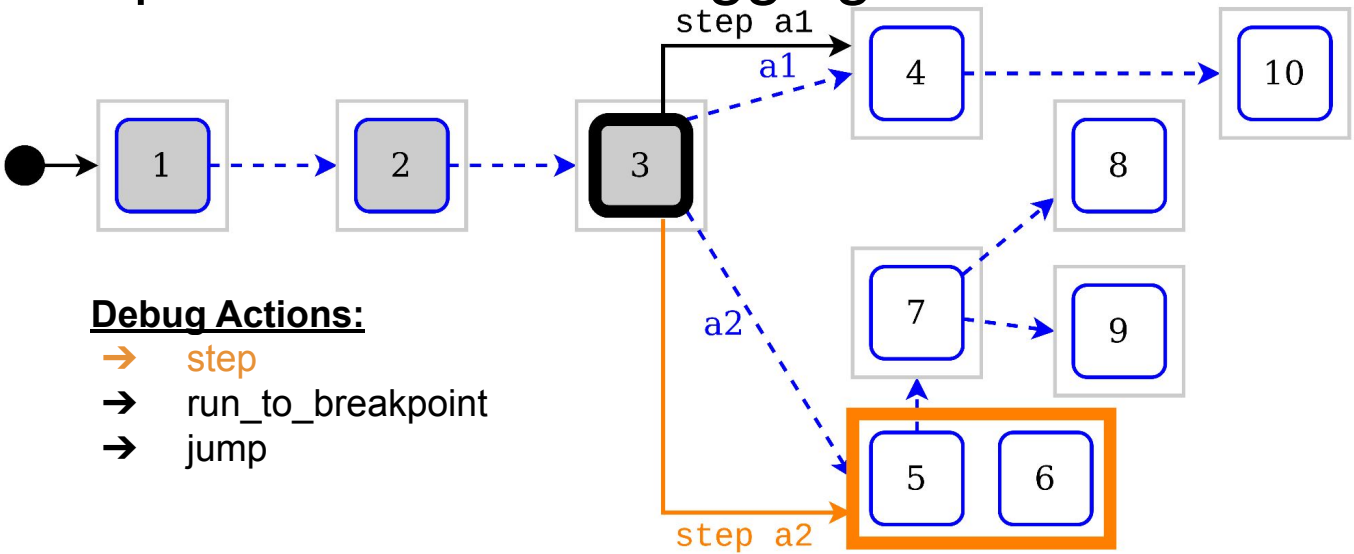
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; Navigating second case of ND
- ➔ run_to_breakpoint
- ➔ jump

How to handle non-determinism requirements for debugging?



Debug Actions:

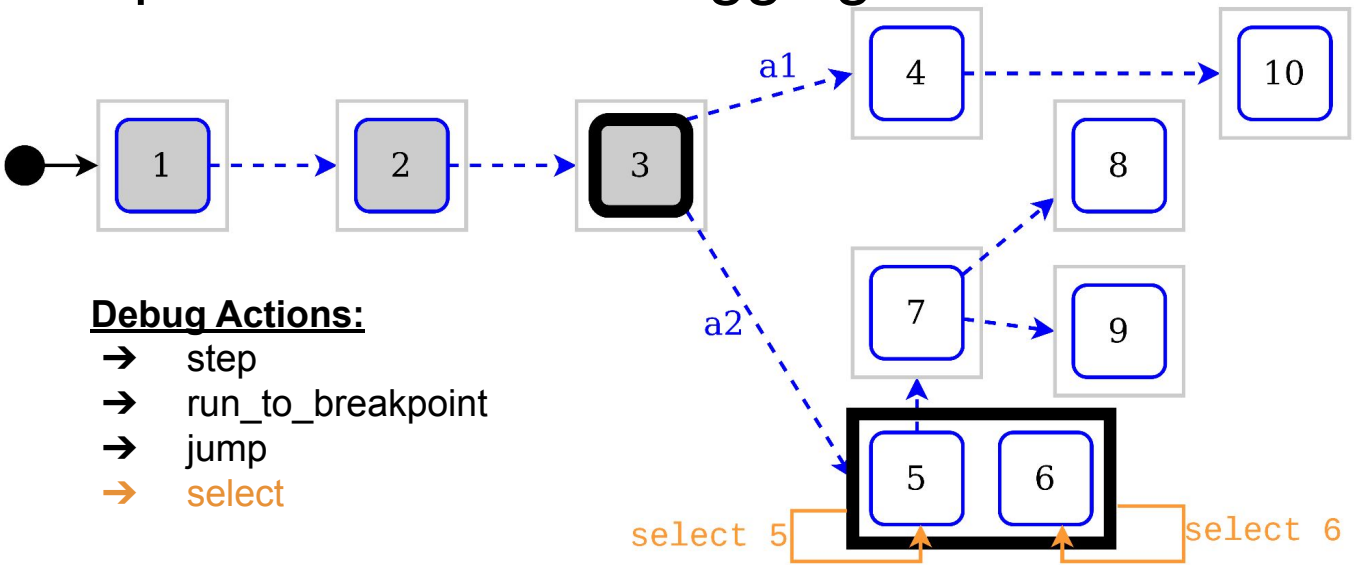
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; Navigating second case of ND
- ➔ run_to_breakpoint
- ➔ jump

How to handle non-determinism requirements for debugging?



Debug Actions:

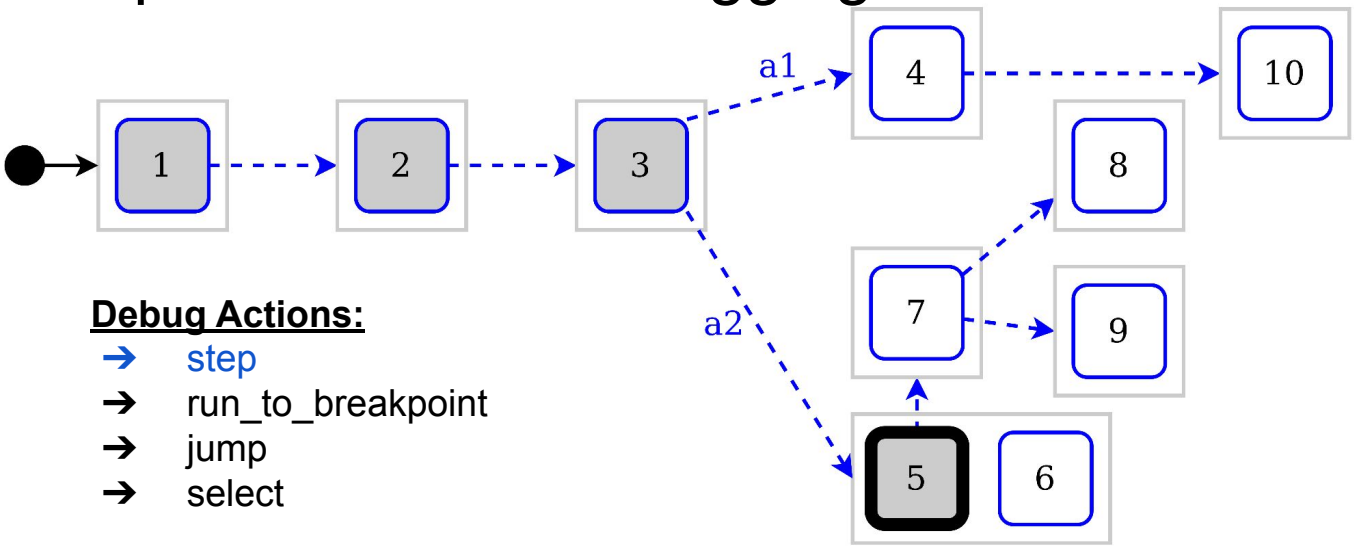
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle non-determinism requirements for debugging?



Debug Actions:

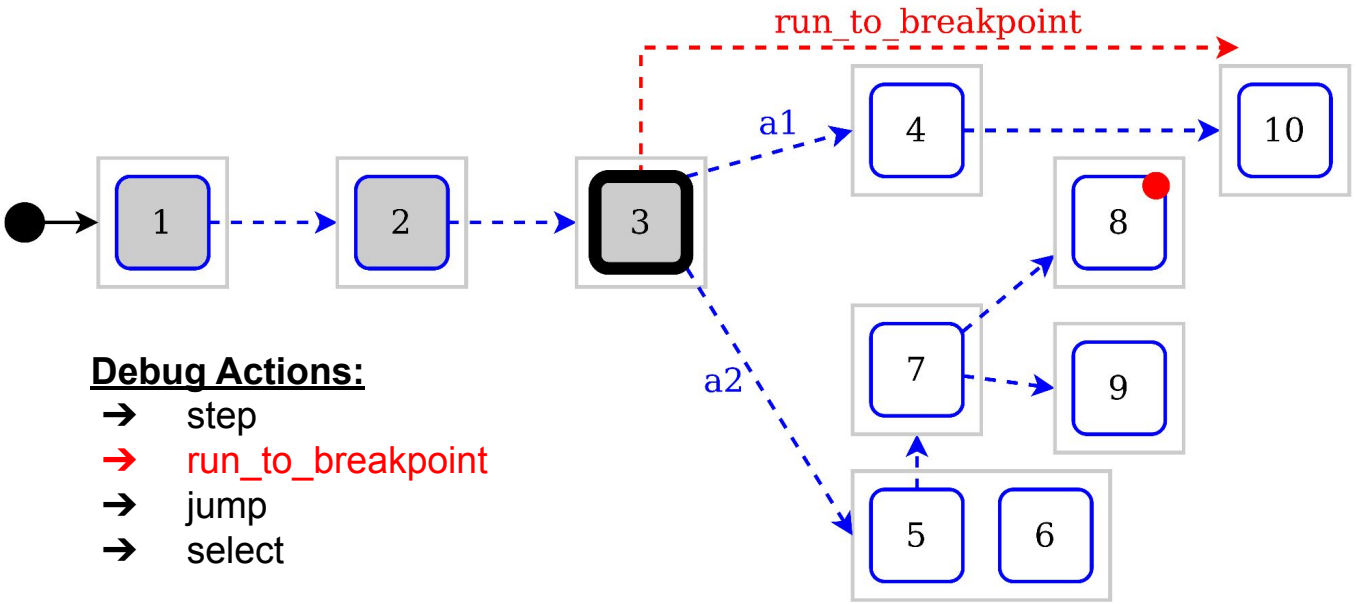
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle breakpoints in a ND system?



Debug Actions:

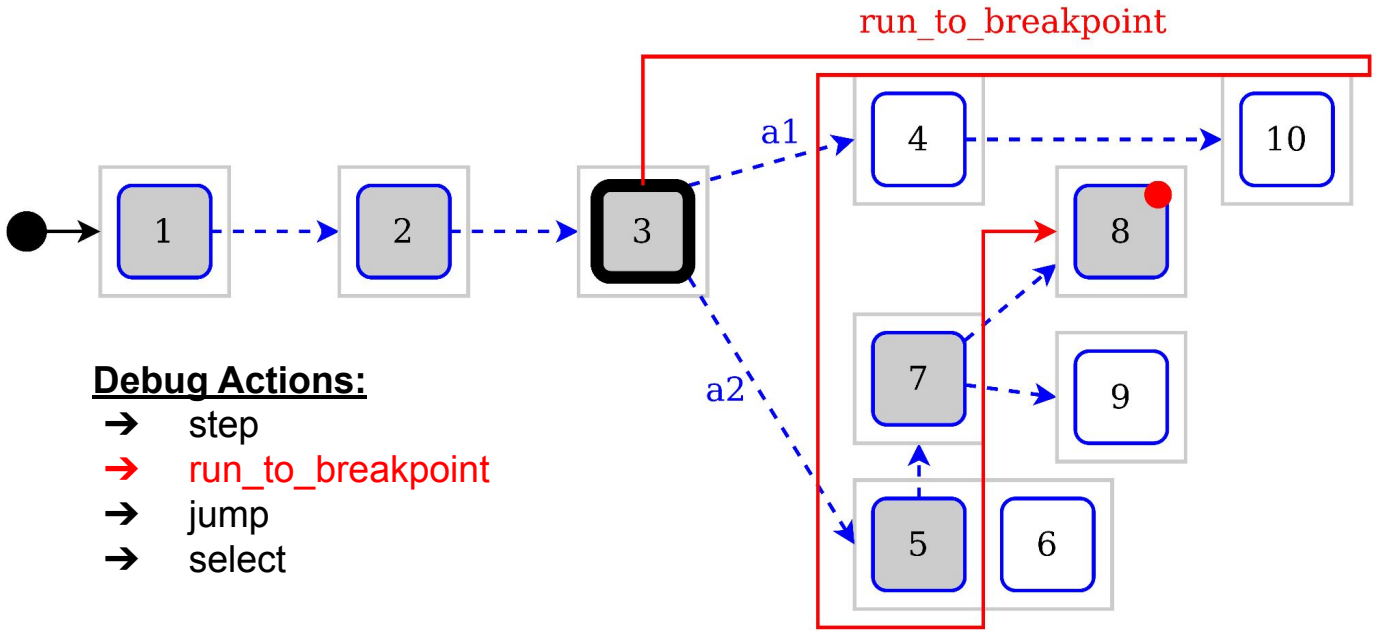
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Missing breakpoints when exploring a single branch
- ➔ jump

How to handle breakpoints in a ND system?



Debug Actions:

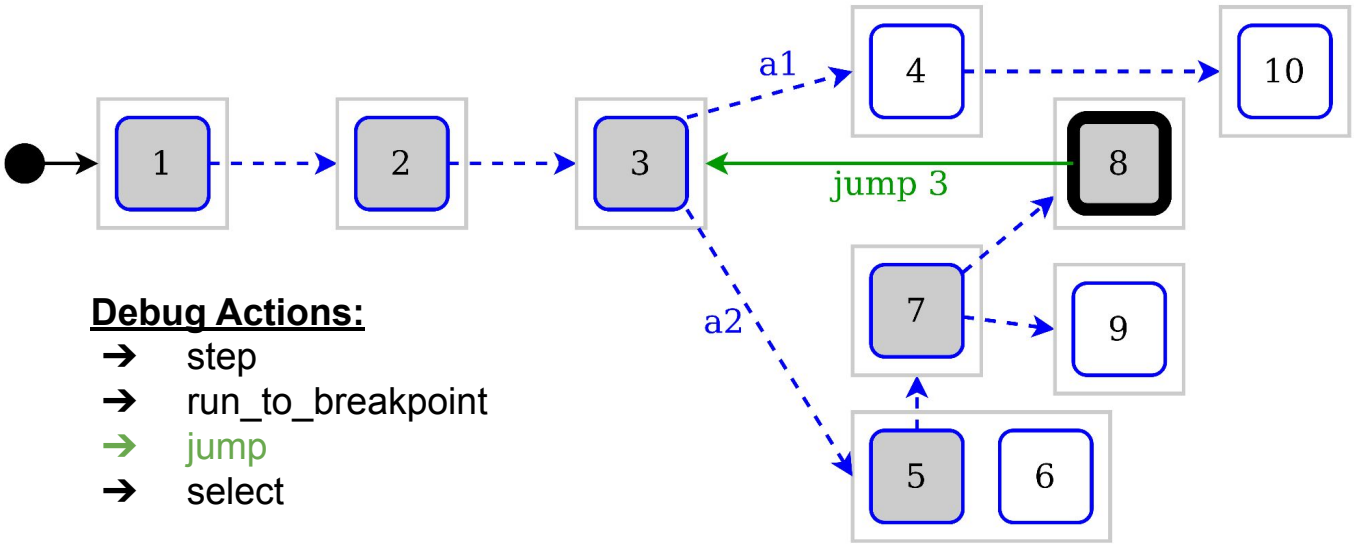
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump

How to handle jumps in a ND system?



Debug Actions:

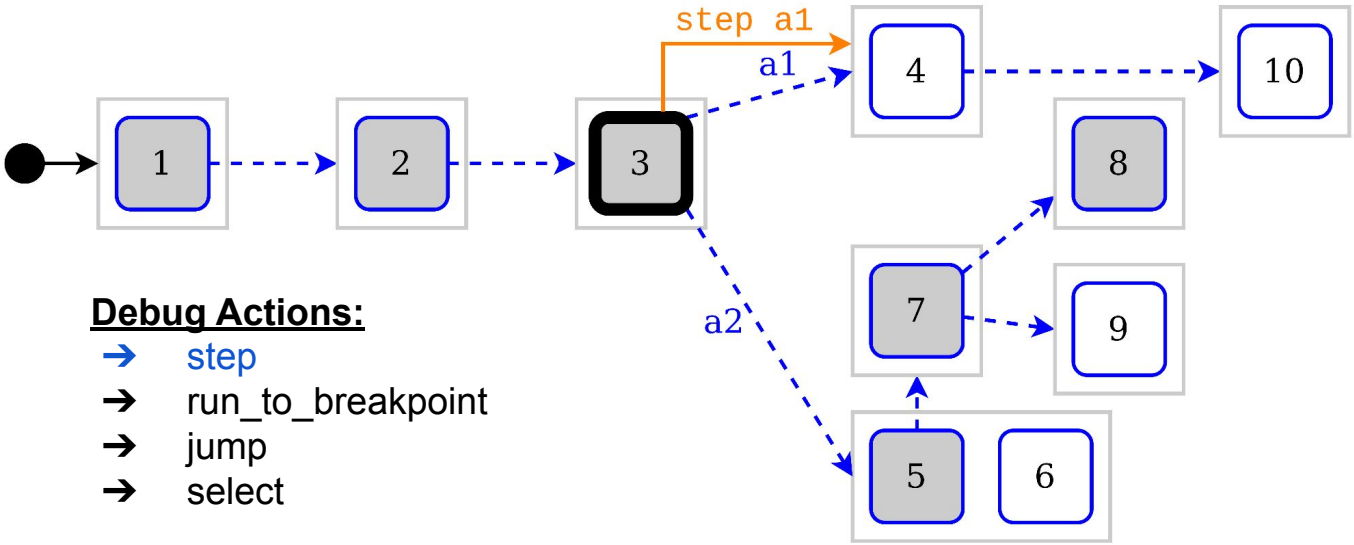
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

How to handle jumps in a ND system?



Debug Actions:

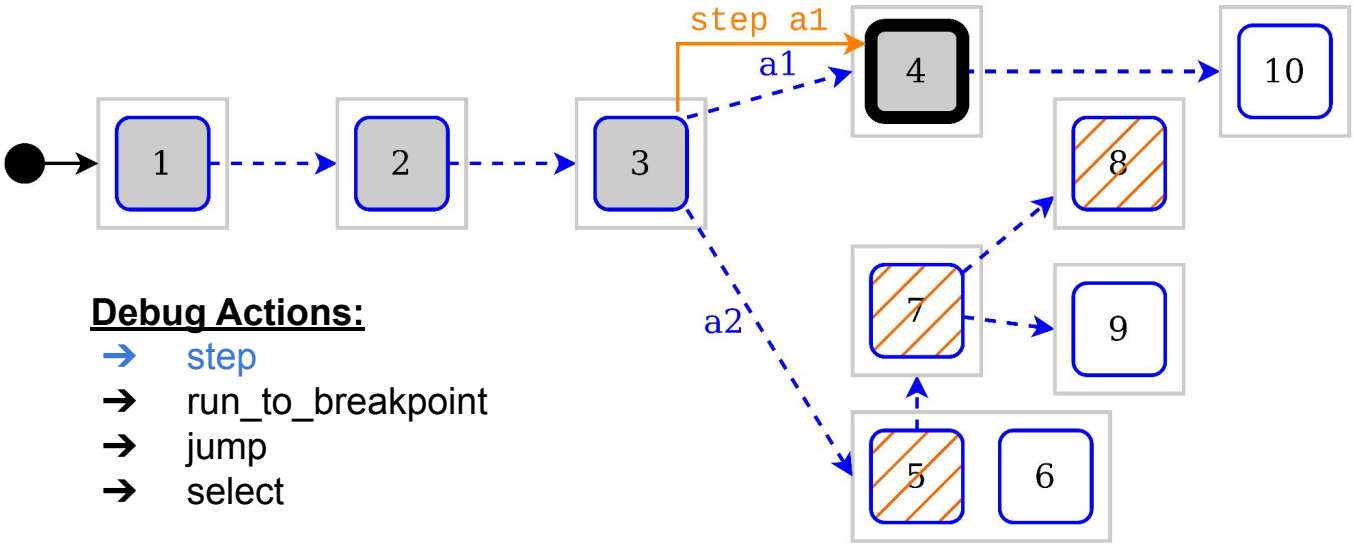
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump: History is lost when switching to another end-to-end branch

How to handle jumps in a ND system?



Debug Actions:

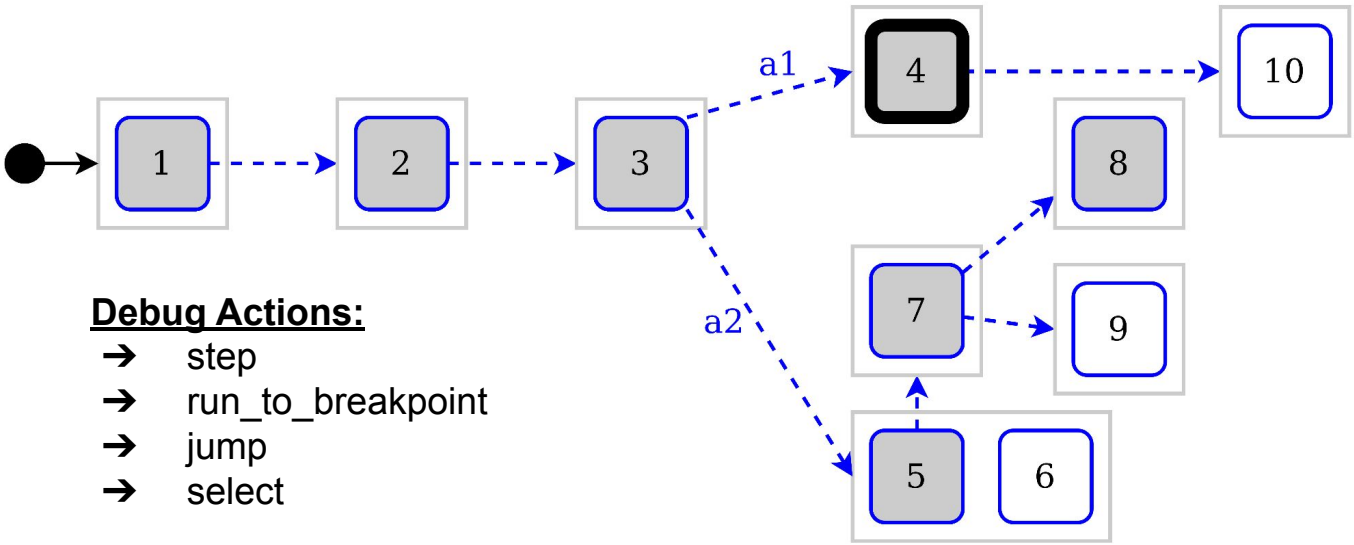
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump: History is lost when switching to another end-to-end branch

How to handle jumps in a ND system?



Debug Actions:

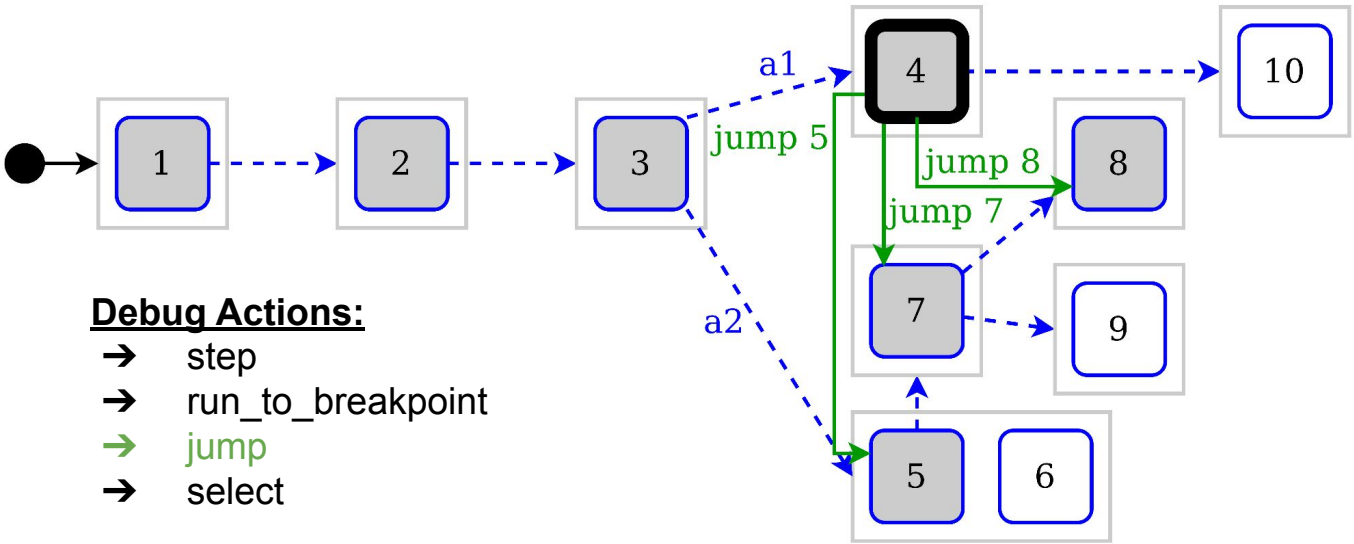
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump: Storing the trace as a tree

How to handle jumps in a ND system?



Debug Actions:

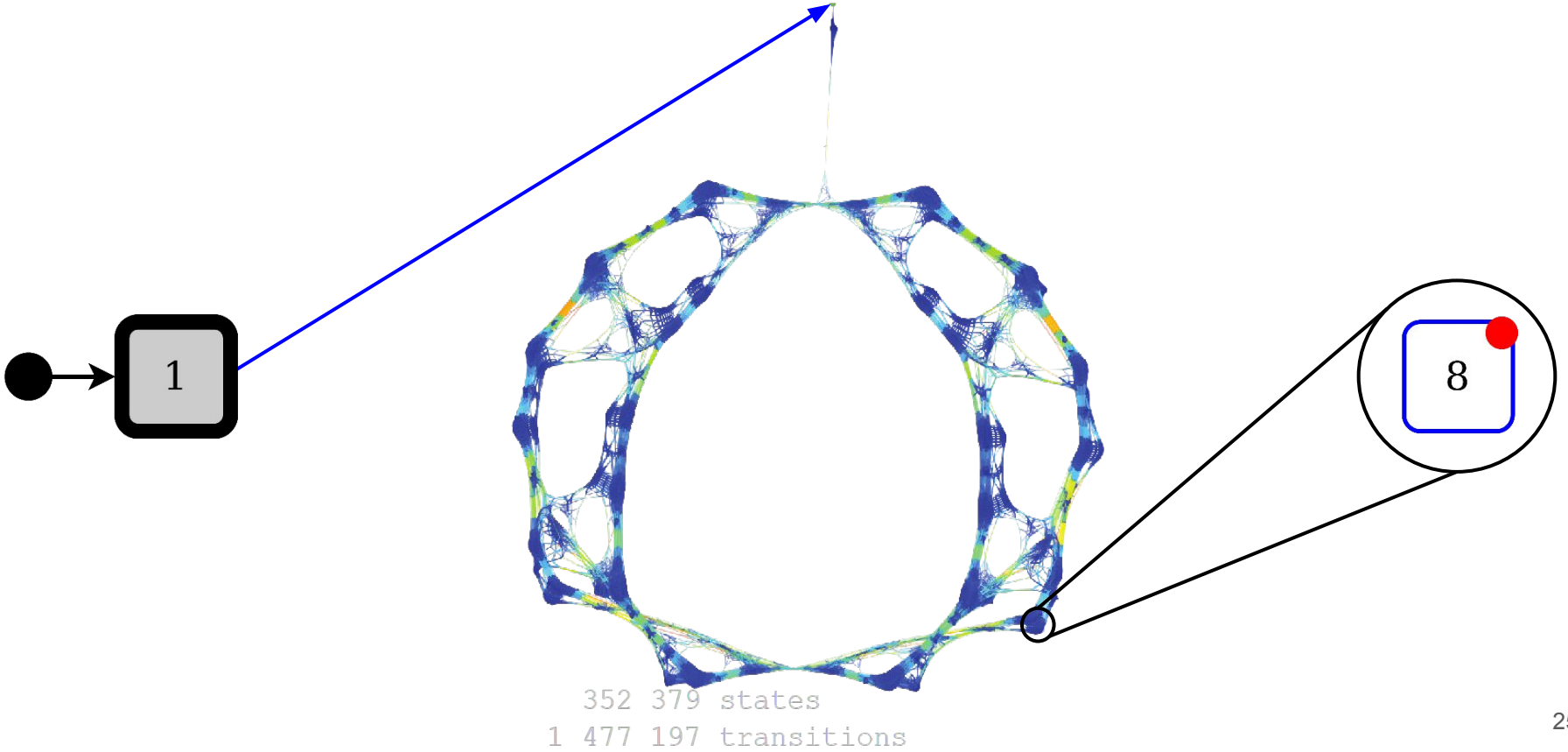
- ➔ step
- ➔ run_to_breakpoint
- ➔ jump
- ➔ select

Solving problems with non determinism:

- ➔ step: Exposing multiple possible steps; New select debug action
- ➔ run_to_breakpoint: Reachability query over the system
- ➔ jump: Storing the trace as a tree

```
x := 1;
x := 2;
x := 3;
actions {
    a1 -> {
        x := 4;
        x := 10;
    }
    a2 -> {
        x := any of {5,6};
        if (x == 5) {
            x := 7;
            actions {
                -> x := 8;
                -> x := 9;
            }
        }
    }
}
```

Problems of Multiverse debugging



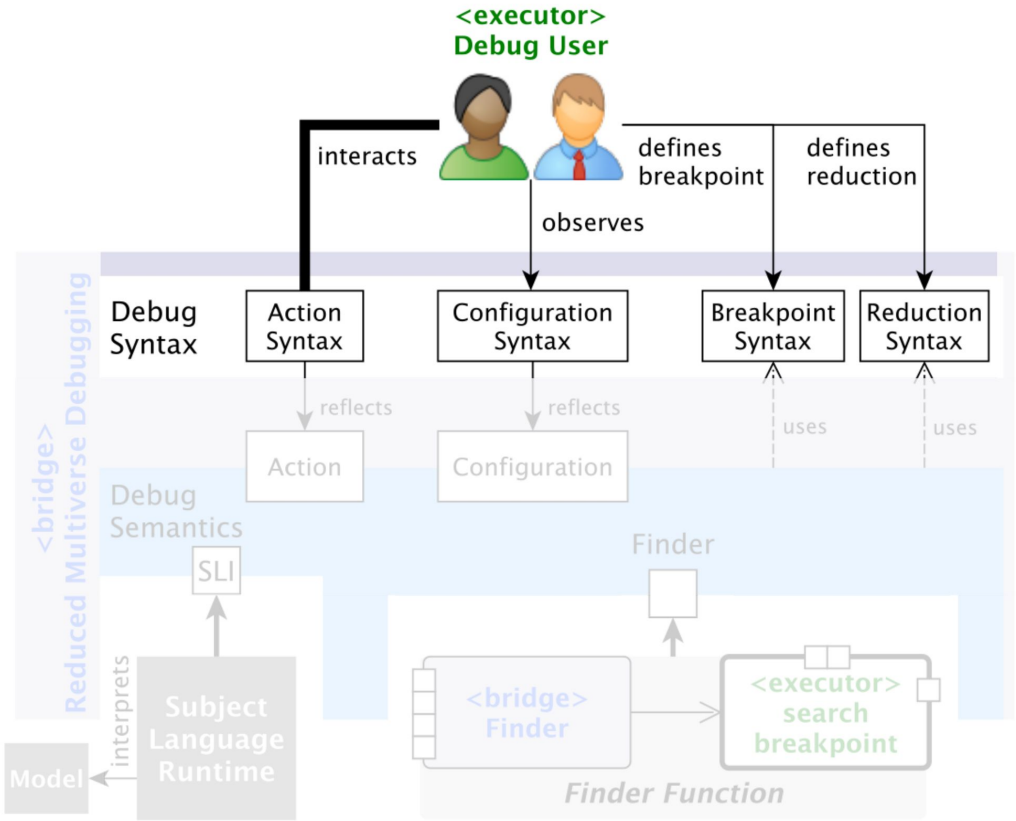
Our contribution: Reduced Multiverse Debugging (RMD)

- Keeping the useful aspects of multiverse debugging[1] for specification languages
  - Explicit non-deterministic debug steps
  - Storing the trace as a tree
  - Searching for breakpoints on all branches
- While making this approach scalable to real-size projects
  - By using reduction policies during breakpoint search
- Debug semantics formalization[2]
  - Making it easily adaptable to new target languages
- Application to UML

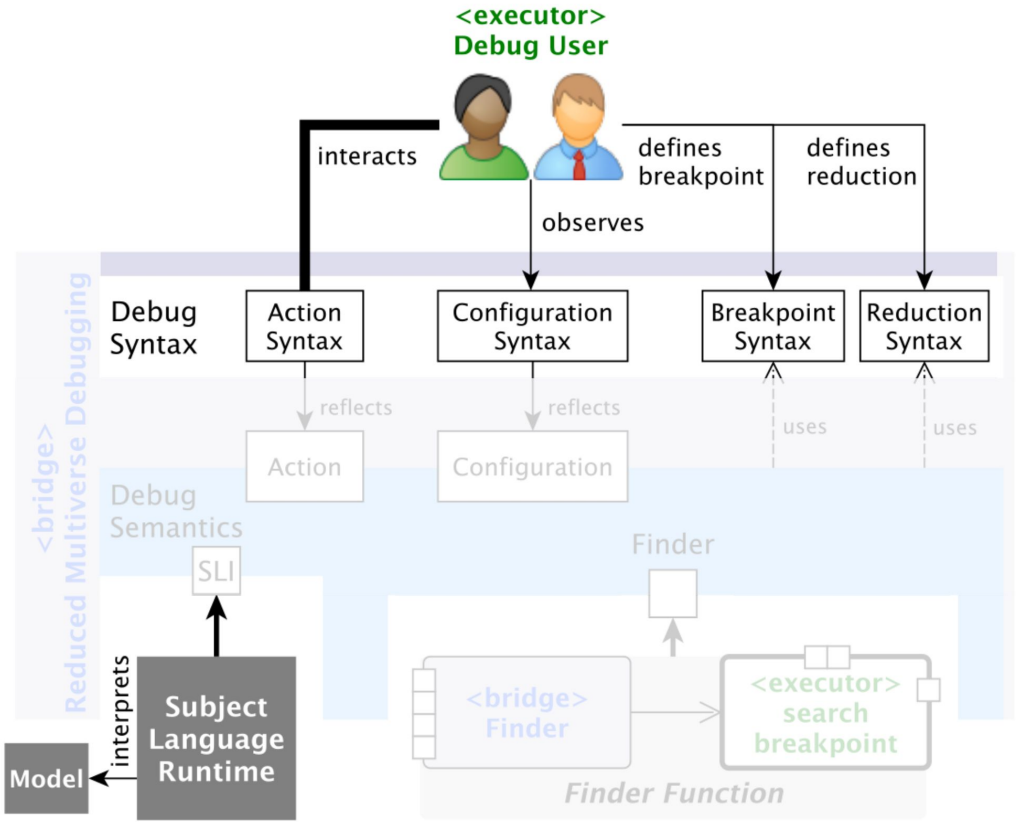
[1] Carmen Torres et Al. 33rd European Conference on OOP 2019. *"Multiverse Debugging: Non-deterministic Debugging for Non-deterministic Programs"*

[2] Valentin Besnard. Ph. D. Dissertation 2020. *“EMI: An approach to unify analysis and embedded execution with a controllable model interpreter.”*

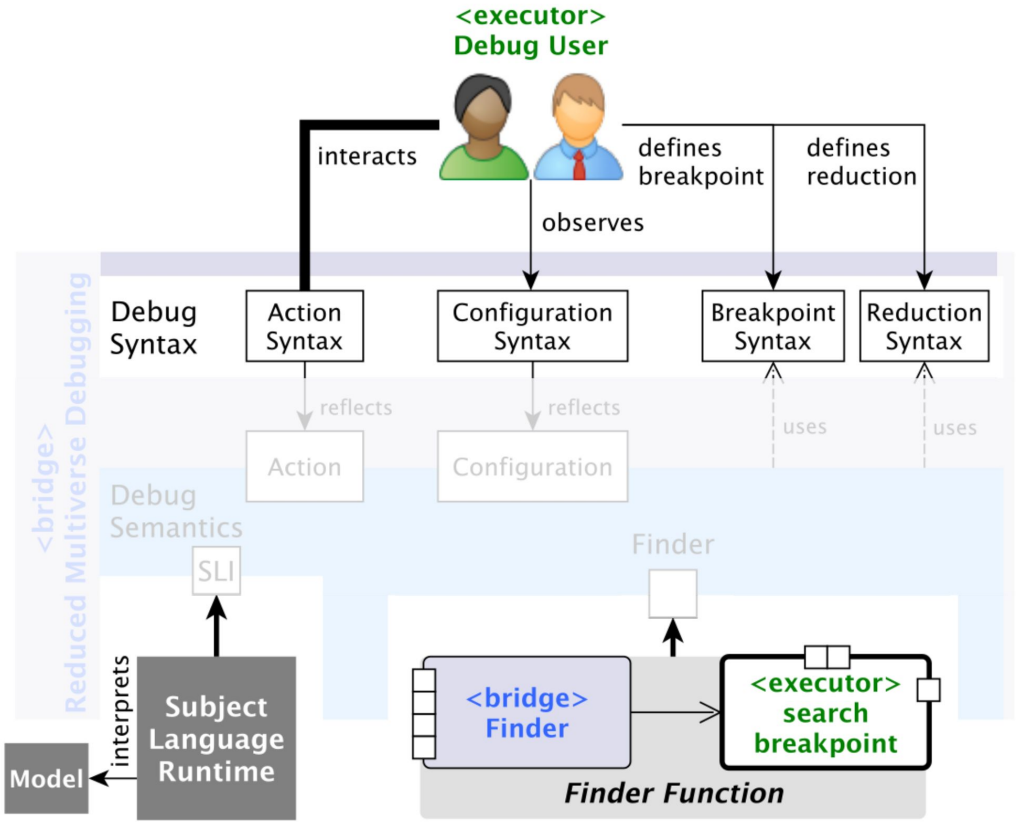
Architecture of Reduced Multiverse Debugging (RMD)



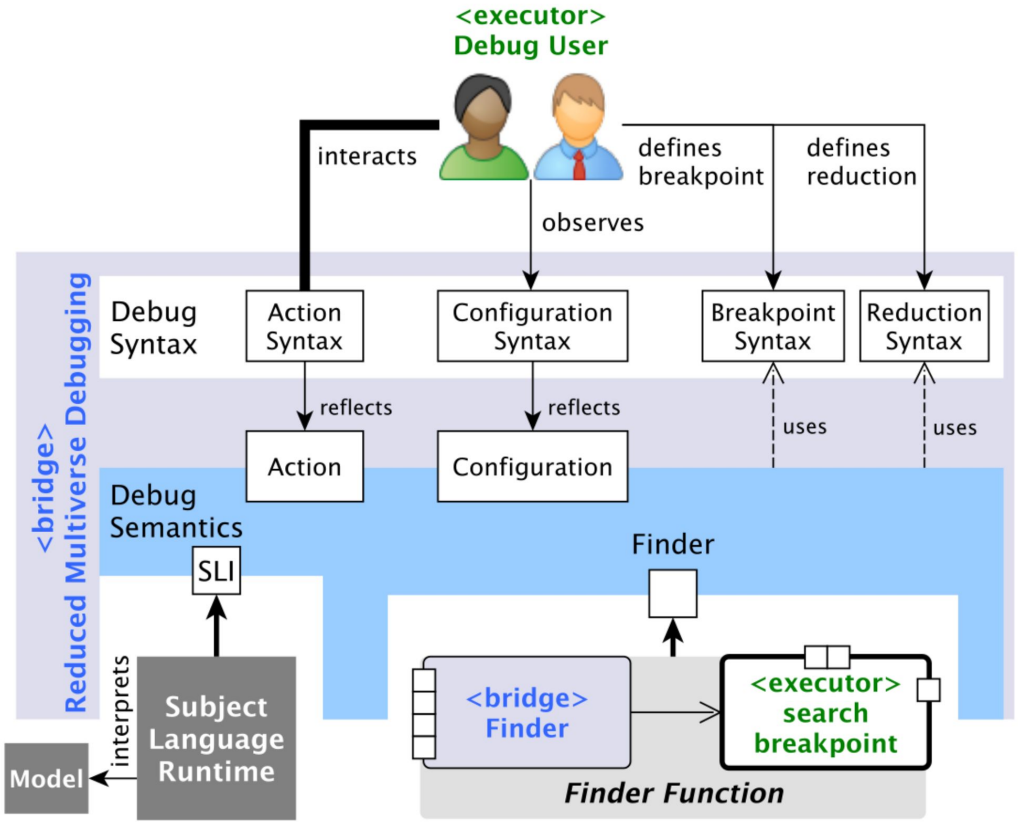
Architecture of Reduced Multiverse Debugging (RMD)



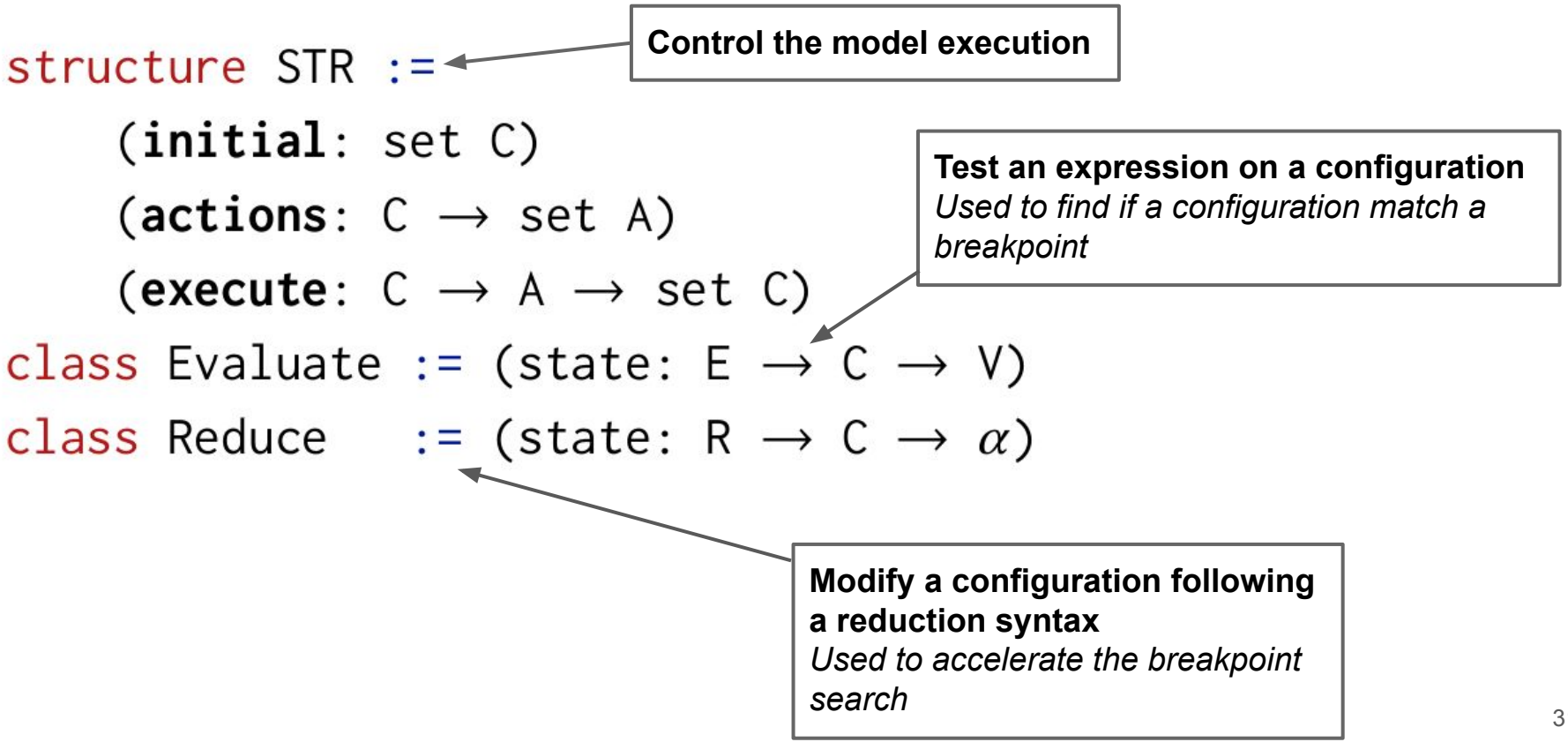
Architecture of Reduced Multiverse Debugging (RMD)



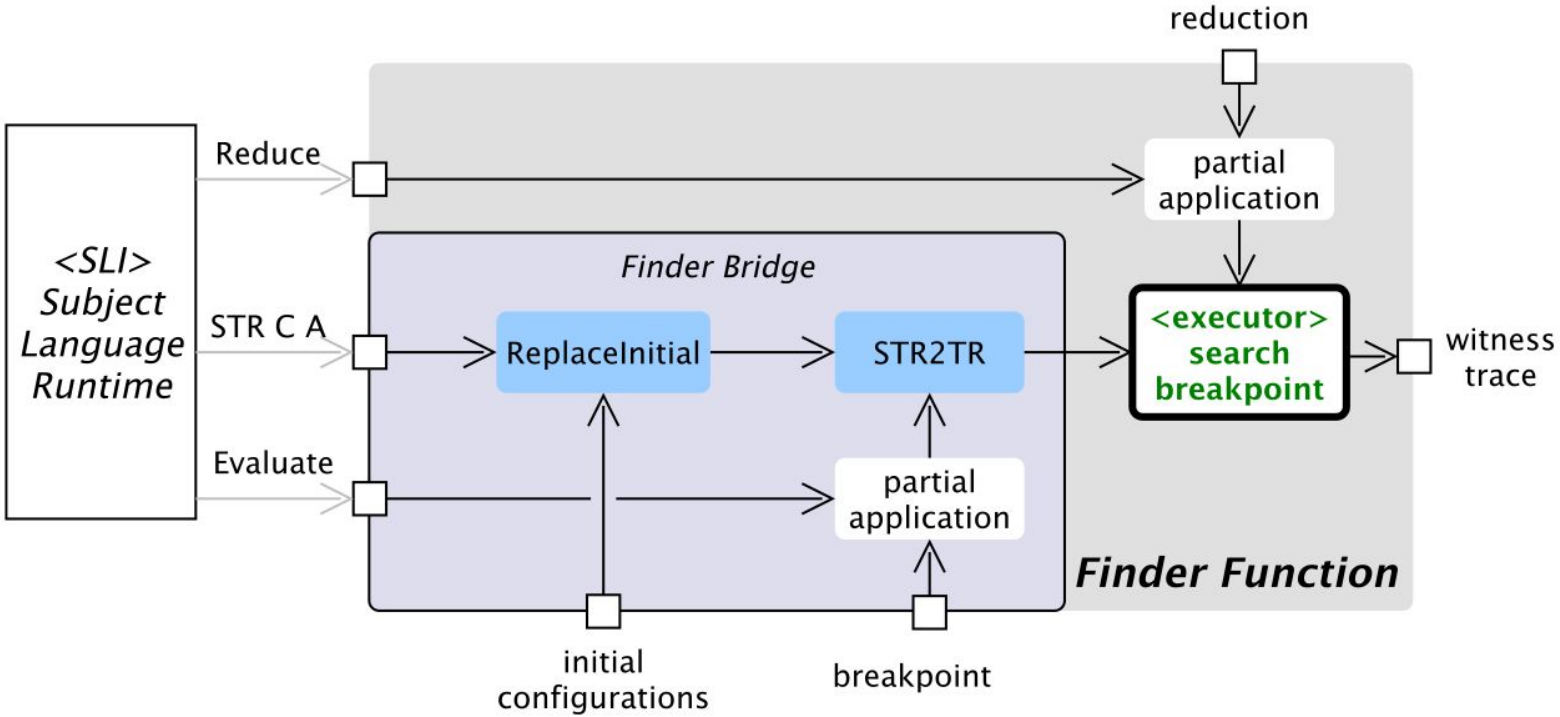
Architecture of Reduced Multiverse Debugging (RMD)



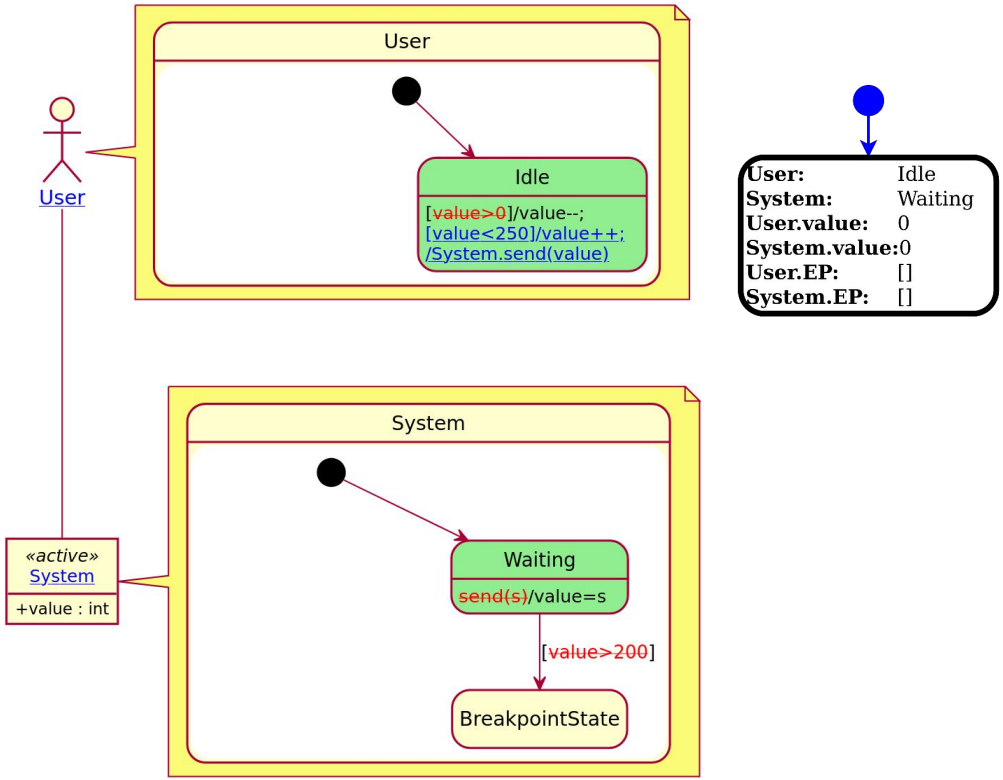
Semantic Language Interface (SLI)



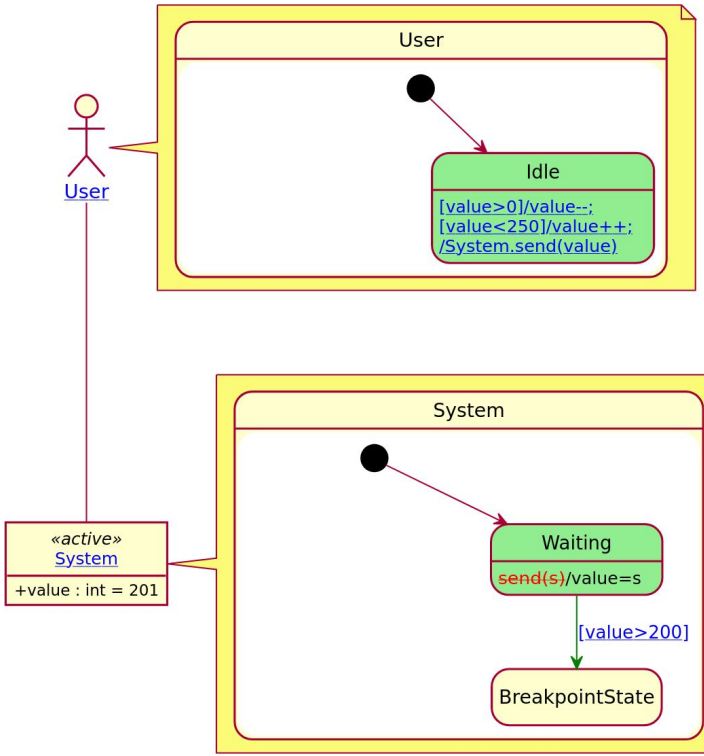
The Finder Function

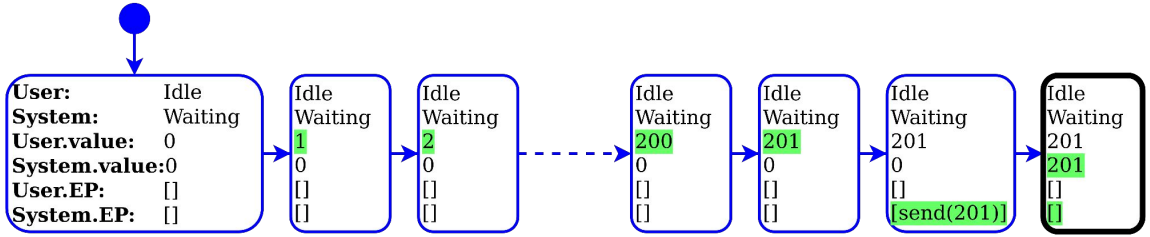


Experimentation: Benefits of the reduction function

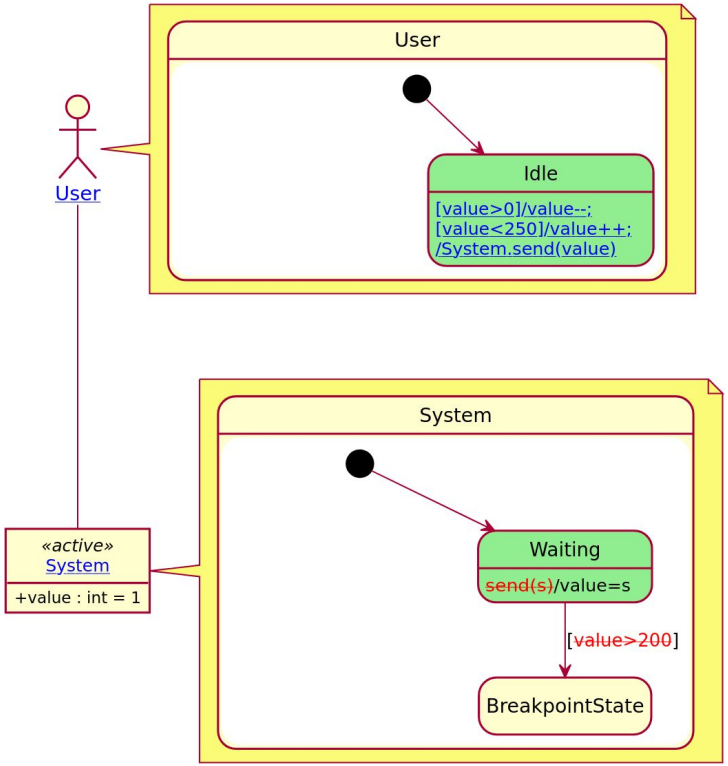


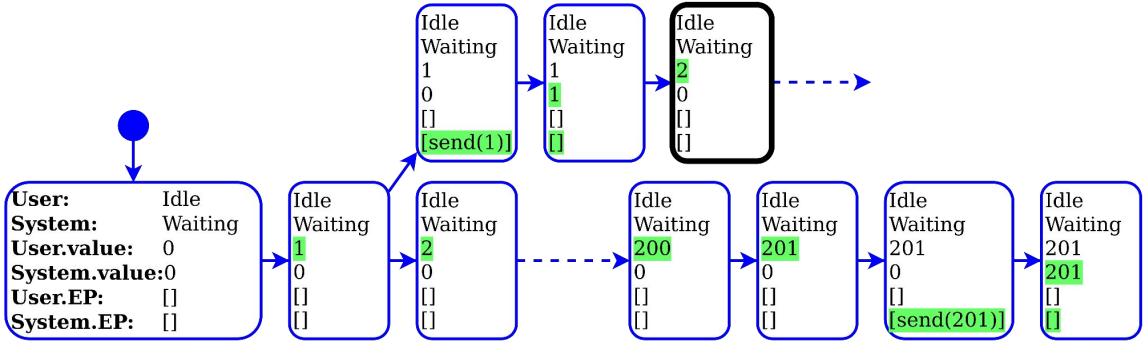
Experimentation: Benefits of the reduction function



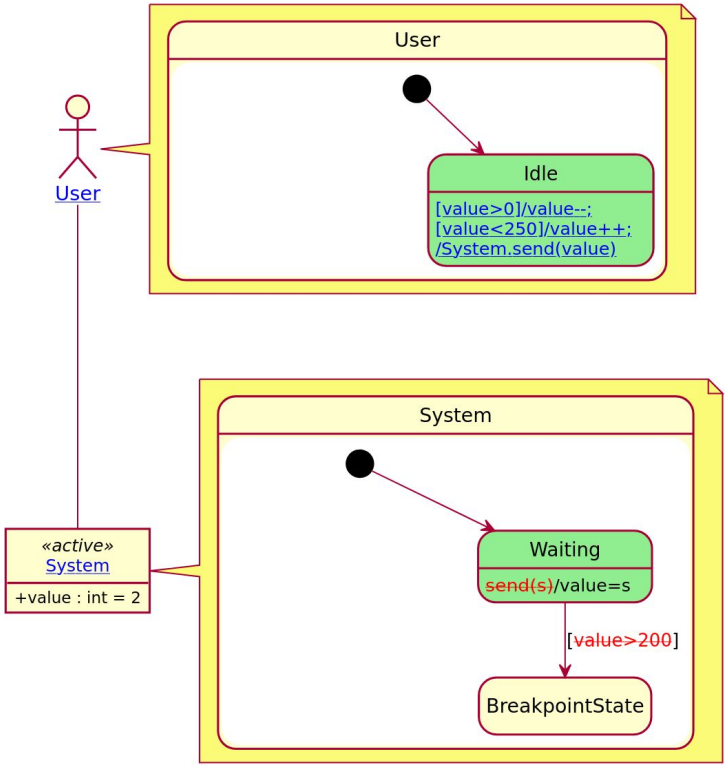


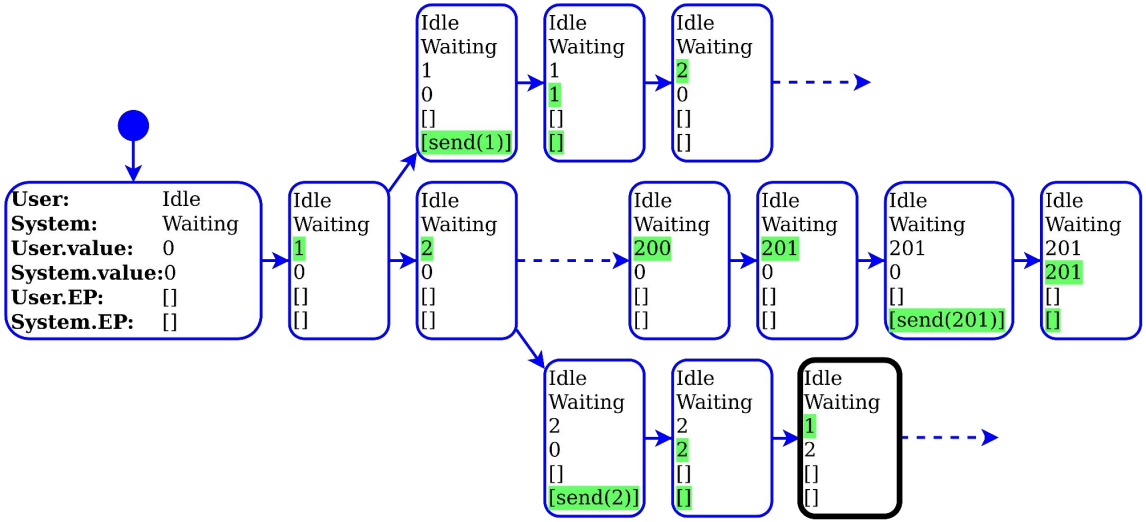
Experimentation: Benefits of the reduction function



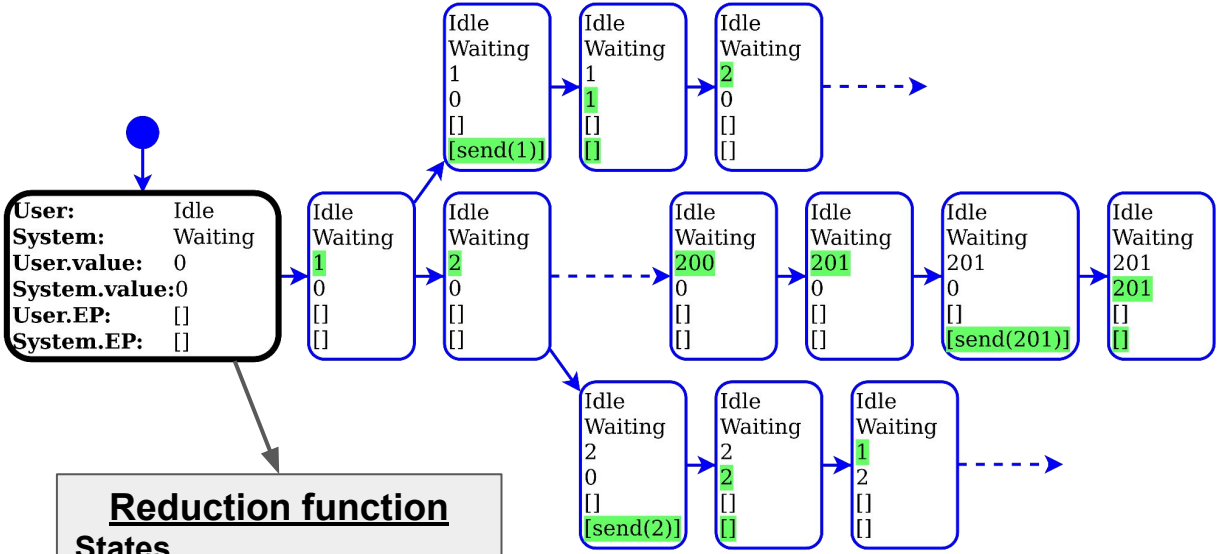


Experimentation: Benefits of the reduction function





Applying a reduction function



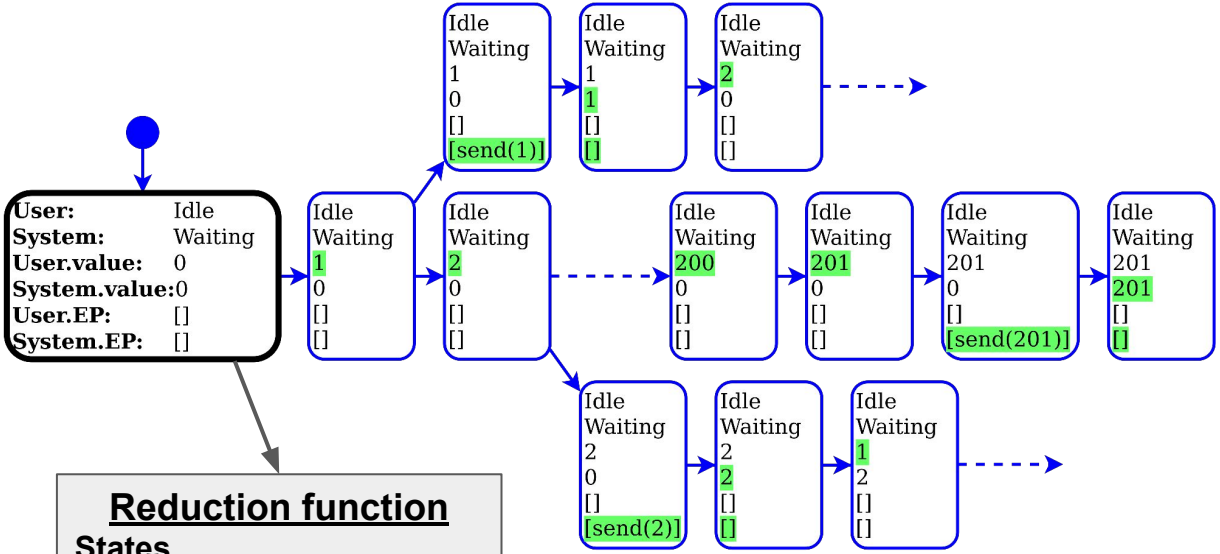
Reduction function

States
- User
- System

Values
- User.value
- System.value

EventPools
- User.EP
- System.EP

Applying a reduction function



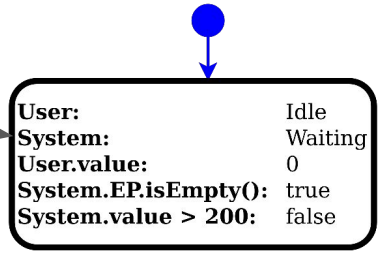
Reduction function

States
- User
- System

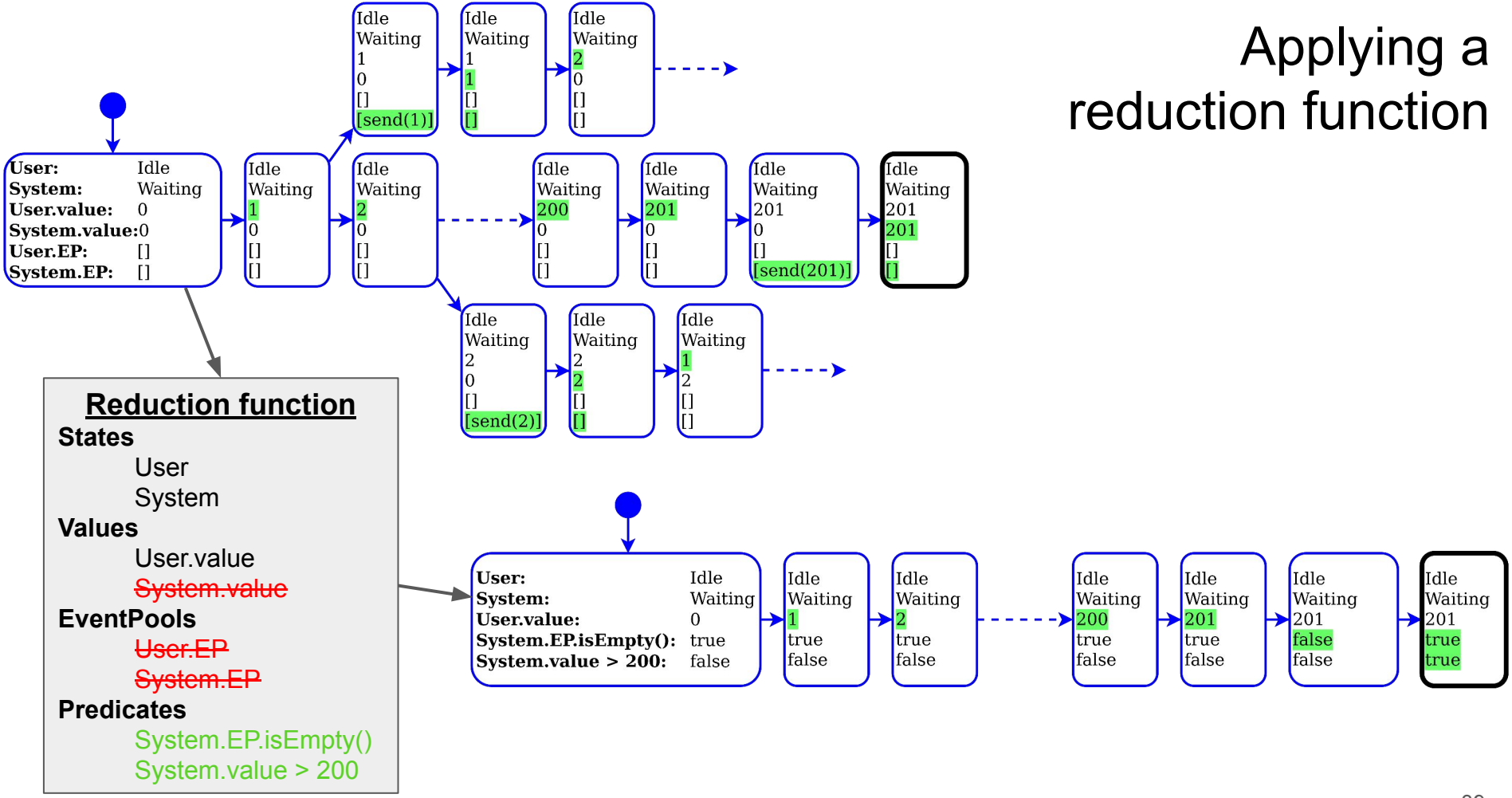
Values
- User.value
- ~~System.value~~

EventPools
- ~~User.EP~~
- ~~System.EP~~

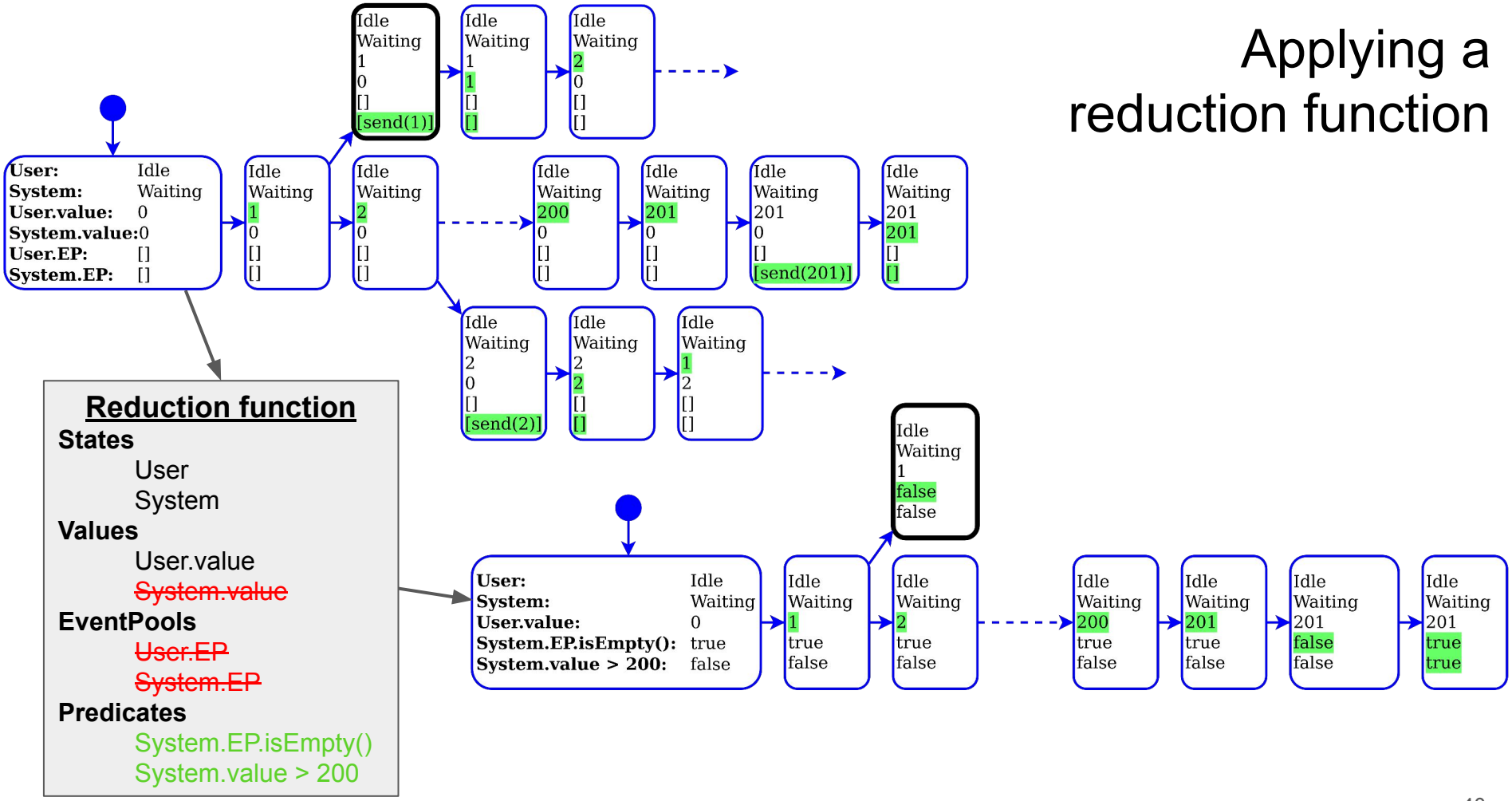
Predicates
- System.EP.isEmpty()
- System.value > 200



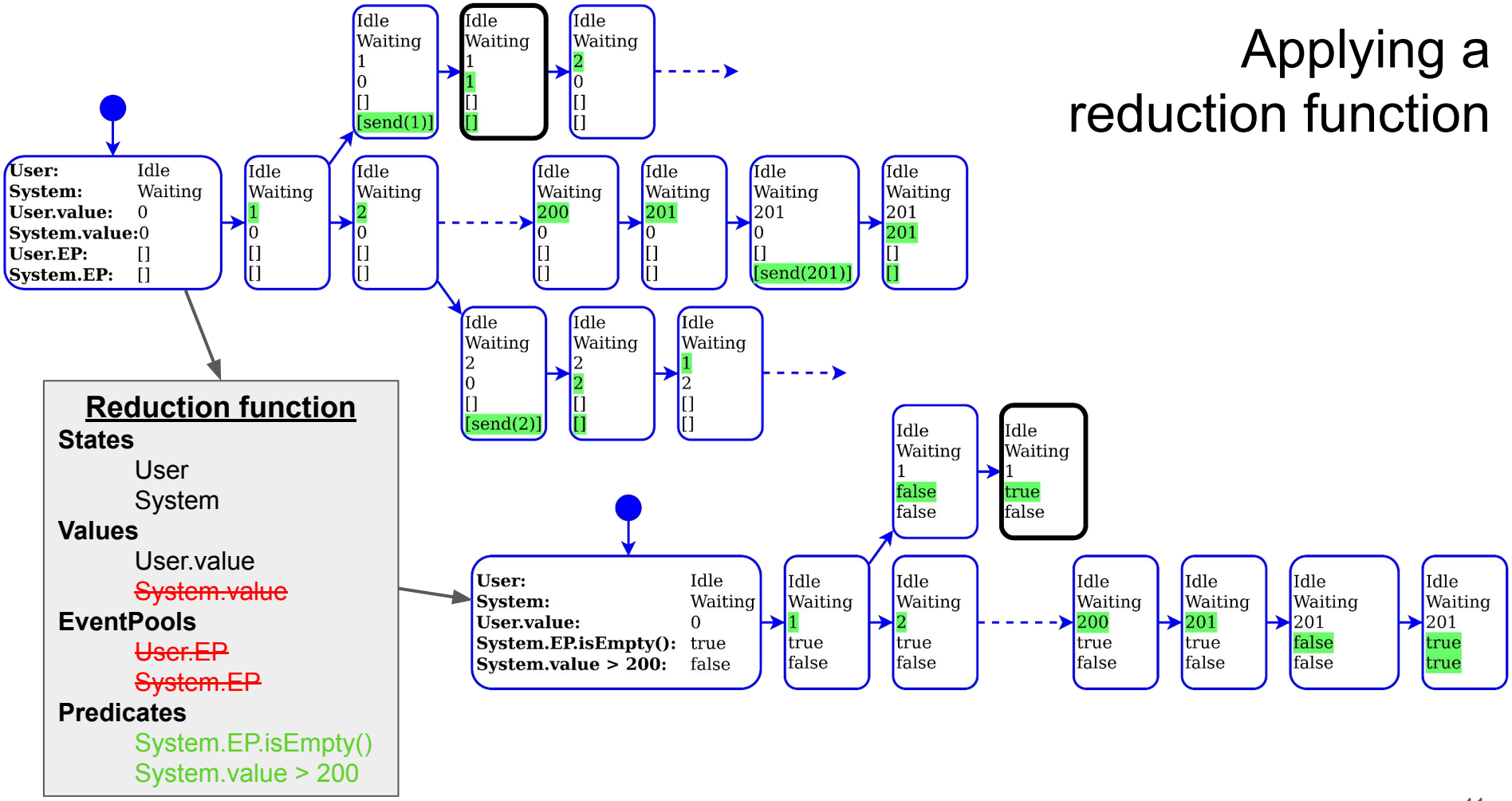
Applying a reduction function



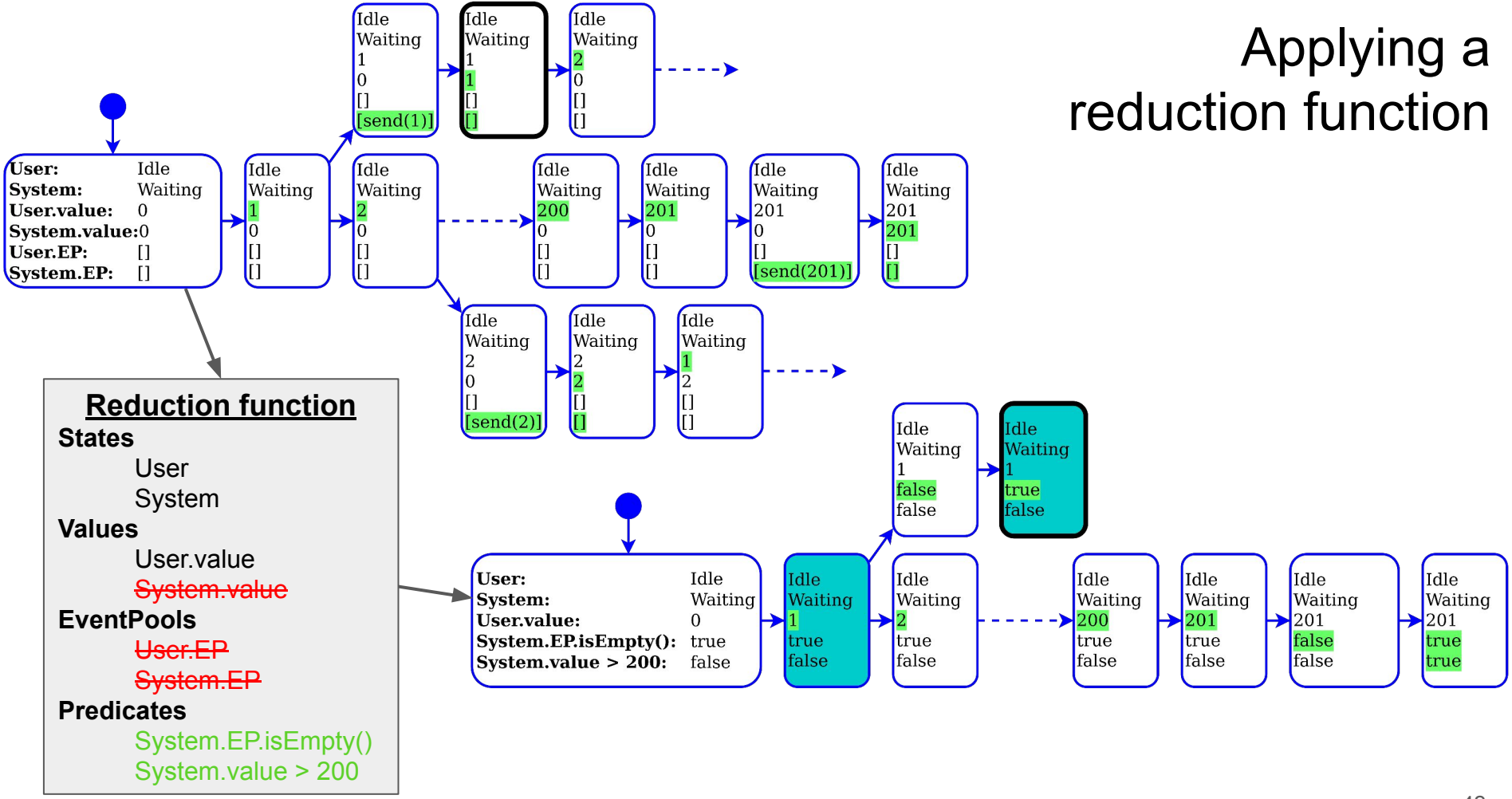
Applying a reduction function



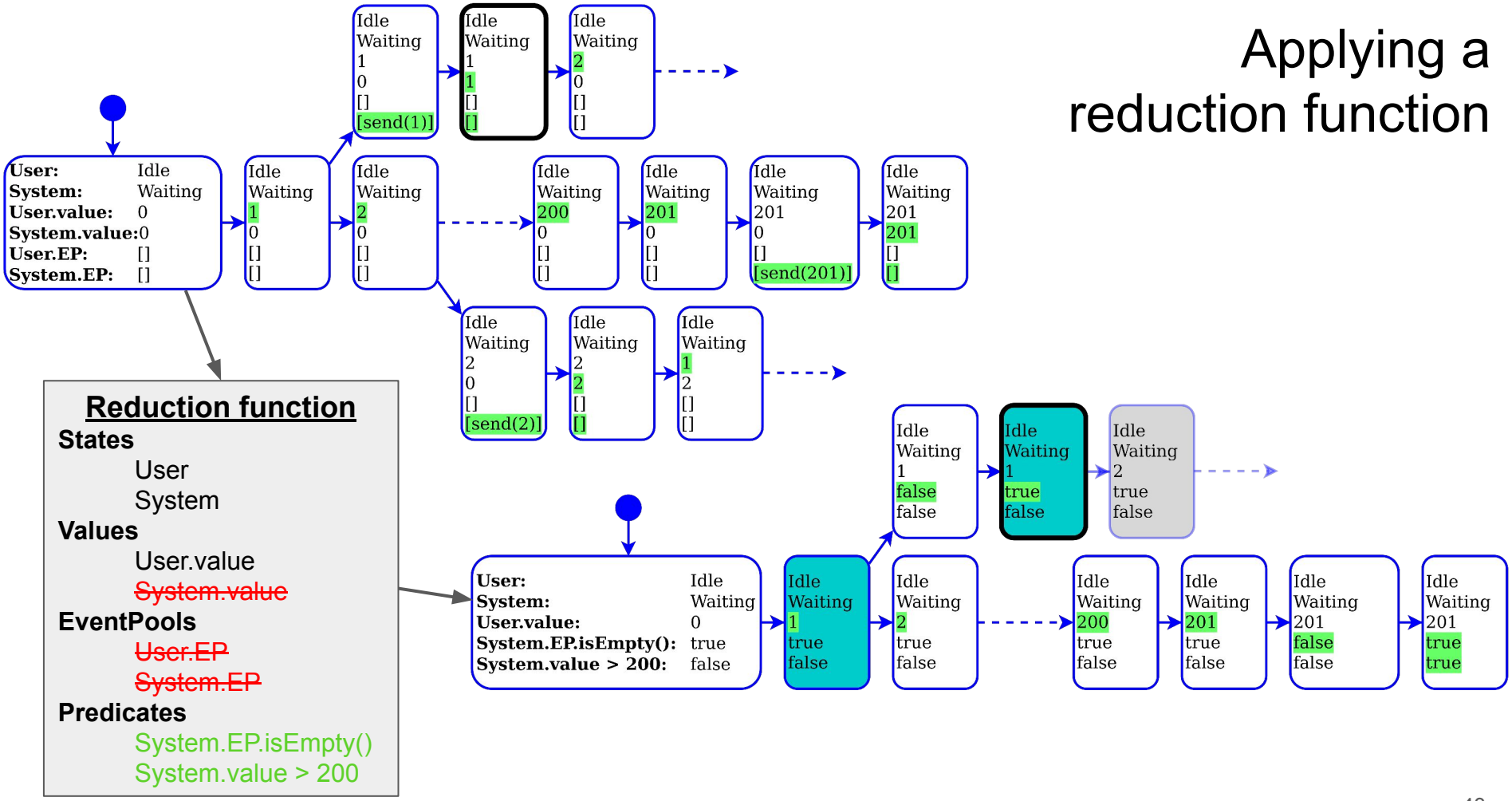
Applying a reduction function



Applying a reduction function

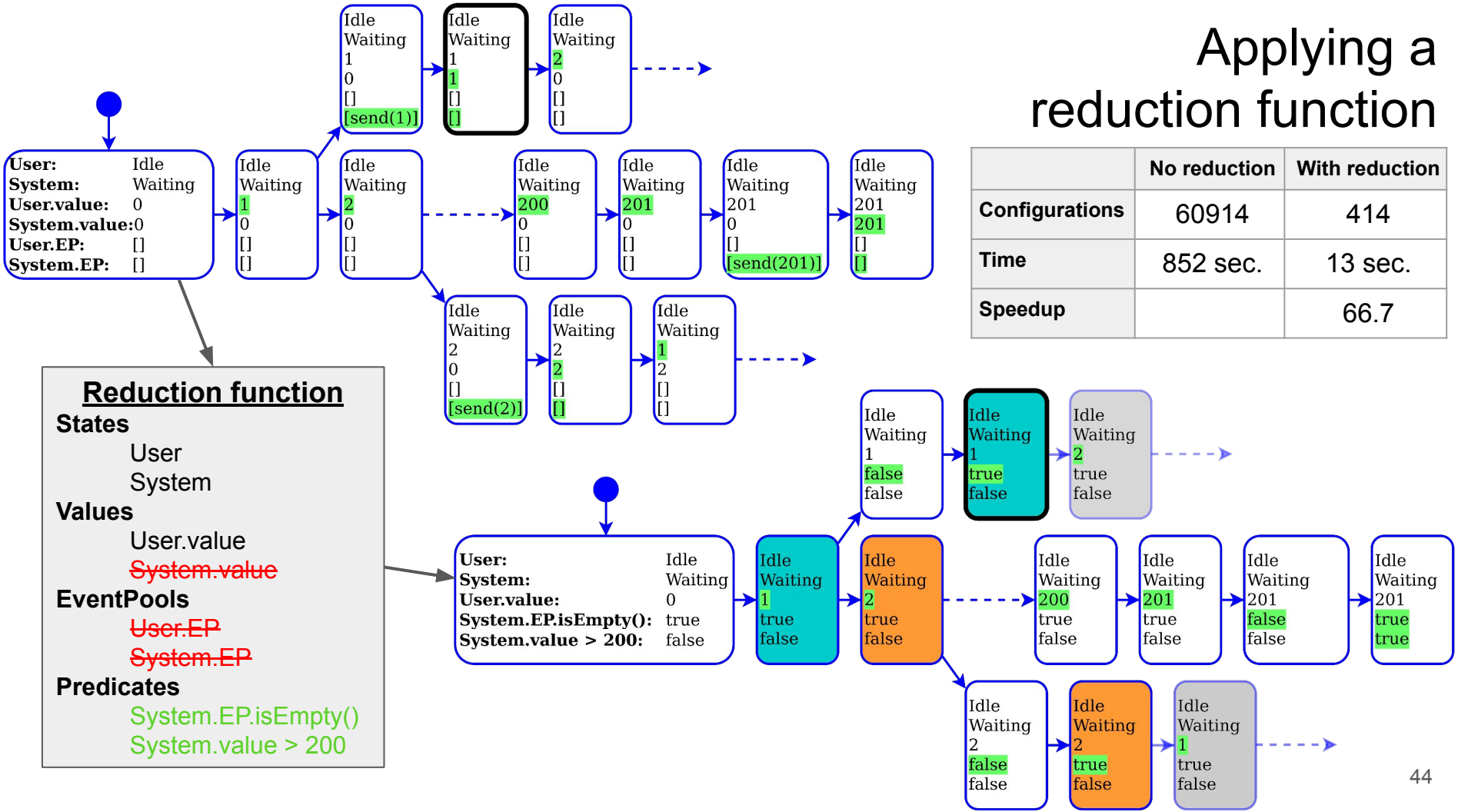


Applying a reduction function



Applying a reduction function

	No reduction	With reduction
Configurations	60914	414
Time	852 sec.	13 sec.
Speedup		66.7



More reduction functions

This technique allows us to use some common reduction techniques during our debugging session, for exemple:

- Bitstate hashing[1]
- Dead variable reduction[2]
- Symmetry reduction[2]
- Predicate abstraction[3]

[1] Gerard J. Holzmann. Springer US 1996. *“An analysis of bitstate hashing.”*
[2] Kelly Androutsopoulos et Al. ACM Comput. Surv. 2013. *“State-Based Model Slicing: A Survey.”*
[3] Naoki Kobayashi et Al. PLDI 2011. *“Predicate Abstraction and CEGAR for Higher-Order Model Checking”*

Conclusion

- Multiverse debugging is useful for working with non-deterministic specification languages.
- Reduced Multiverse Debugging is a way to make multiverse debugging scalable to system with bigger, or even infinite state-space.

Perspectives

- Adapting the breakpoints expression to the non-deterministic behaviour of the system.
- Finding appropriate reduction functions, at a general level, a language level or model level.
- Evaluating the usability of these features when used in a debugging session.