École polytechnique de Louvain

Observing the detailed behaviour of large distributed applications in real time using $\triangle$QSD

Author: **Francesco Nieri**
Supervisor: **Peter Van Roy**
Readers: **Tom Barbette, Peer Stritzinger, Neil Davies**
Academic year 2024–2025
Master [120] in Computer Science

Contents

Chapter 1

Defining a grammar for an outcome diagram

1.1 A

1.1.1 Primitives

Causal link

A causal link between two components can be defined by a right arrow from `component_i` to `component_j`

```
component_i -> component_j
```

1.1.2 Components' names

Components' names must follow the following regex pattern: `/[a-zA-Z0-9_]+/`

1.1.3 Observables

We define two observable parts in our system, probes and outcomes.

Probes

A probe is an observable of the system, represented by a start and end event, measuring the delay of an execution of ....

A probe can contain one component or a sequence of causally linked components.

The user can define as many probes as they want, they have to be declared as follows:

```
probe = component [-> component2];
probe2 = newComponent -> anotherComponent;
```

Probes will not be parsed after the system has been defined, in the case below, an error will be thrown.

```
probe = ...;
probe2 = ...;
system = ...;
probe3 = ...;
```

Proes can be reused in other probes or in the system by adding a s: before they are used.

```
probe3 = s:probe -> s:probe2;
```

Outcome

Outcomes are the smallest observable unit in the system, an outcome can be followed by another component, indicating a causal link between the two.
In the definition of the system or a probe, an outcome is defined with its name.

```
probe = outcomeName;
```

1.1.4 Operators

All operators must contain at least two components, this is because the operations to calculate the DeltaQ of these operators rely on using the CDF of the components that define the operator.

All components inside an operator must be separated by a comma.

All-to-finish operator

An all-to-finish operator needs to be defined as follows:

```
a:operator(component1, component2...)
```

First-to-finish operator

A first-to-finish operator needs to be defined as follows.

```
f:operator(component1, component2...)
```

Probabilistic choice operator

A probabilistic choice operator needs to be defined as follows:

```
p:operator[probability_1, probability_2, ...
 ↪  probability_i](component_1, component_2, ..., component_i)
```

In addition to being comma separated, the number of probabilities inside the brackets must match the number of components inside the parentheses.

1.1.5 Parsing and syntax

We perform parsing using Lark in python, the grammar is based on an ENBF syntax (add citation here). The grammar we defined in this section can be summarised by the following syntax.

```
\%ignore WS


start: definition* system?  //Subsystem first, system second
definition: IDENTIFIER "=" component+ ";"

system: "system" "=" component+

component: outcome ("->" component)?
        | BEHAVIOR_TYPE ":" IDENTIFIER "(" component_list ")" ("->"
        ↪  component)?
        | probe ("->" component)?

outcome: IDENTIFIER
probe: PROBE_IDENTIFIER ":" IDENTIFIER
component_list: component ("," component)+

PROBE_IDENTIFIER: "s"
BEHAVIOR_TYPE: "f" | "a" | "p"

IDENTIFIER: /[a-zA-Z0-9_]+/
```