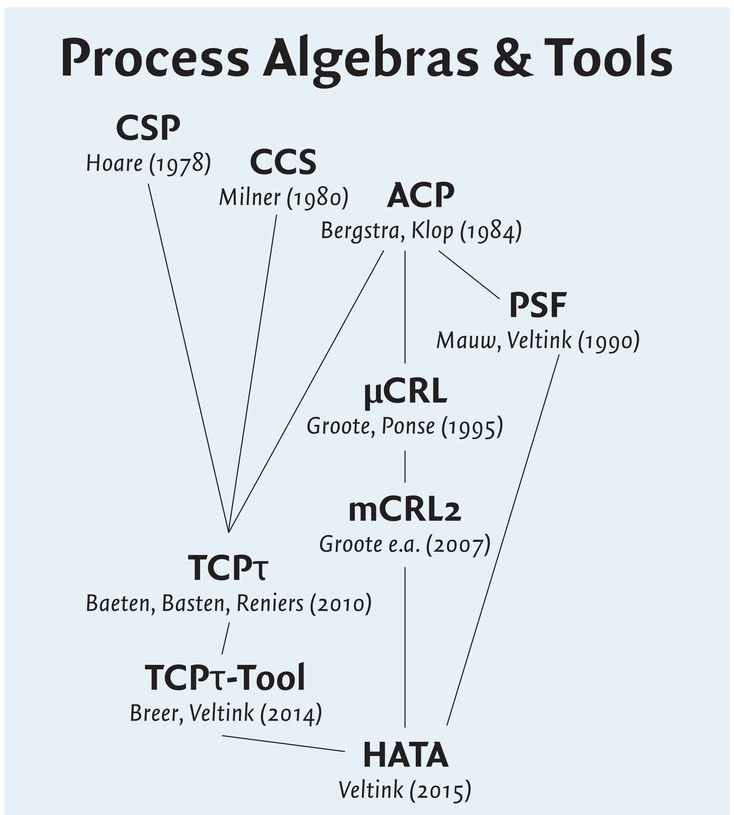
University of Applied Sciences
HOCHSCHULE EMDEN·LEER

HATA

Process Algebra in HATA • 1

The formal theory driving HATA specifications - Sequential Processes

Goals
HATA offers an extension to the programming language Java that allows for describing the behaviour of concurrent processes by means of an implementation of a process algebra. This process algebra is mostly based on ACP, but also uses aspects from mCRL2 and TCPτ. However, the HATA notation differs from the original formal process theories in order to appeal to the intuition of programmers familiar with current day program languages.
The theoretical context is shown in the following image.



Actions
The basic unit of execution is an action, an indivisible event. In HATA an action is modelled by a call to a Java method. An action can be parametrized by data. In HATA the data types can be any Java reference type. The combination of the name of the action and the types of its parameters is called its *signature*.
An example of a parameterized action is inserting a coin into a vending machine:

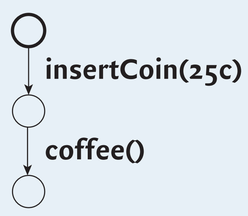
```
insertCoin(25c)
```

Process Operators
Actions can be combined into larger *process terms* by applying process operators. The two essential operators are introduced below.

Sequential Composition
This is the most common operator. It is used to define an order between actions. We can express that an action *b* can only happen after action *a* has finished as follows: *a* ; *b*. Here the semicolon is used as an operator on the two operands: *a* and *b*.
A vending machine that serves coffee for 25 cents could be modelled as:

```
insertCoin(25c); coffee()
```

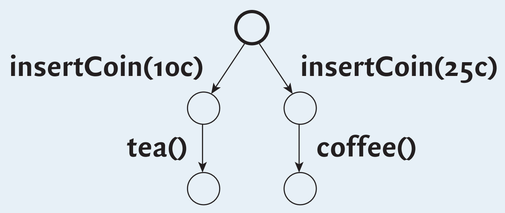
The resulting transition system is given below:



Alternative Composition
The next operator implements a choice between two possible continuation paths. We can express that either the process term *x* or the process term *y* can happen (but not both!) as follows: *x* | *y*.
Now we can extend the example of the vending machine with the possibility to serve tea for 10 cents:

```
insertCoin(25c); coffee()
| insertCoin(10c); tea()
```

The resulting transition system shows the two possible execution paths:



Non-deterministic choice
By chaining a number of alternative compositions, we can essentially specify a number of next possible options for the current state. If there is more than one possible continuation, one of the possible next steps is chosen non-deterministically.

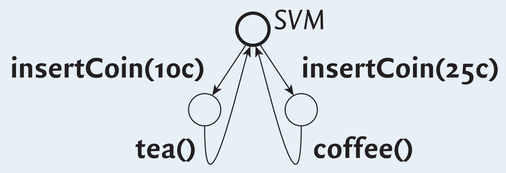
Processes
Sofar, we have only defined process terms. Processes are defined by relating a process (variable) to a process term. Process identifiers typically start with an uppercase character.
We can define a *Simple Vending Machine* (*SVM*) using the process term from the previous example.

```
SVM = insertCoin(25c); coffee()
| insertCoin(10c); tea()
```

Recursion
There is no explicit operator for repeating behaviour (cf. loops). Recursion is used instead. The vending machine from the previous example has a serious drawback, it can only serve one cup of either tea or coffee. A more realistic model would allow for serving multiple drinks. This is achieved by incorporating a process variable into a process term.

```
SVM = (insertCoin(25c); coffee()
| insertCoin(10c); tea()) ; SVM
```

After serving a drink the machine returns to its initial state. The resulting transition system is shown below:



The final figure shows, that the process name (*SVM*) can be used as a reference to the initial state of the transition diagram.

Process Algebra vs. Programming Languages

Process algebras traditionally use a different notation for the process operators than the one used in HATA. The sequential composition and action-prefix are traditionally written as: "." or "·", the alternative composition as "+". As HATA explicitly aims at appealing to the intuition of programmers, a different notation has been chosen. The semicolon is used for separating program statements and implicitly defining an order. The use of "|" meaning "*or*" is common in boolean expressions.

Conclusion

The formal theory used in HATA to specify sequential process behaviour has been introduced above. We have shown how basic atomic actions are related to method calls in Java. We have shown how actions can be transformed into process terms by using the sequential and alternative compositions, and how processes are defined by attaching naming identifiers to process terms. Finally we have shown how loops are modelled through recursion on processes.

Prof. Dr. Gerrit Jan Veltink
gert.veltink@hs-emden-leer.de

Hochschule Emden/Leer
Constantiaplatz 4
26723 Emden