The NC-maude user manual

Marc Boyer
DTIM-ONERA

July 28, 2014

Abstract

This report presents the NC-maude tool, a sand-box to teach, play with and experiments network calculus [3].

1 Introduction

NC-maude tool is a sand-box to teach, play with and experiments network calculus [3]. It is based on the Maude rewriting tool [4].

Once loaded the NC-maude libraries in the Maude interpreter, the user can ask to the Maude kernel to reduce some NC related expression, to compute sum of functions, convolutions, etc.

2 Tutorial

2.1 Warning

Since NC-maude is an open tool, based on rewriting logic, there is no clear difference between user and developer, and you can not use it without any knowledge on its implementation.

2.2 Some first steps

A general-purpose file exists: `startNC.maude`. Just load it at the beginning. Then, you can play. To begin, just a few syntax. A flow is defined by its name and its service curve. At first step, just ask Maude to parse a flow definition with name `"R"` and arrival curve $\gamma_{5,3}$. Second, do the same with a network element of name `"S"` and service curve $\beta_{9,2}$. Then, you can now ask to compute (ie reduce) the delay observed by this flow in this network element. The answer is $\frac{7}{3}$ ([1]). What is the output flow? It's name is `"R|S"`, and its arrival curve is $\gamma_{5,13}$.

[1] You may get a different number of rewrites, since this number is related to internal implementation, and this documentation and the code are not automatically synchronised.

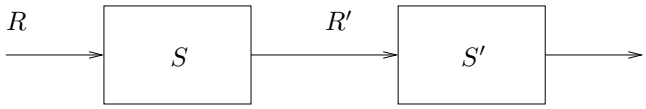

Figure 1: A simple architecture

```
boyer-laptop% maude startNC.maude -no-advise
     \||||||||||||||||||/
   --- Welcome to Maude ---
     /||||||||||||||||||\
    Maude 2.4 built: Nov  6 2008 17:14:50
    Copyright 1997-2008 SRI International
   Fri Aug 21 13:29:56 2009
Maude> parse flow("R",Gamma 5 3) .
Flow: flow("R",Gamma 5 3)
Maude> parse  netElem("S",Beta 9 2) .
NetworkElement: netElem("S",Beta 9 2)
Maude> red delay( flow( "R" , Gamma 5 3 ) ,  netElem( "S" , Beta 9 2 )  ) .
reduce in START-NC : delay(flow("R",Gamma 5 3),netElem("S",Beta 9 2)) .
rewrites: 84
result PosRat: 7/3
Maude> red flow( "R" , Gamma 5 3 ) |  netElem( "S" , Beta 9 2 ) .
reduce in START-NC : flow("R",Gamma 5 3) | netElem("S",Beta 9 2) .
rewrites: 68
result Flow: flow("R|S",Gamma 5 13,netElem("S",Beta 9 2))
```

But in fact, writing interactive network topologies is q bit boring. The normal way to use it is to write a module defining your topology, and to compute things about it. Here is a simple example that you can fin in file `examples/simpleExample.maude`. It describes the architecture presented in Figure 1 where a flow R goes in sequence through two servers S and S'.

```
load ../startNC.maude
mod SIMPLE-EX is
  protecting START-NC .

  ops r r' : -> Flow .
  ops s s' : -> NetworkElement .

  eq r = flow( "R", Gamma 5 4 ) .
  eq r' = r | s .
  eq s = netElem( "S" , Beta 10 2 ) .
  eq s' = netElem( "S'" , Beta 20 1 ).
endm
red r' .
red r | s | s' .
```

```
red delay( r , s ) .
red delay( r' , s' ) .
red delay( r , s ; s' ) .
red float( delay( r , s ) ) .
red float( delay( r' , s' ) ) .
red float( delay( r , s ; s' ) ) .
```

We are going to define four objects: `r`, `r'`, `r'`, `s`, and `s'`, corresponding respectively to flows R and R' and network elements S and S'. Notice that there are no constant in `Maude`, it just exist operators without parameters. You can load this file with the `Maude` command `load`, or simply give it as a command line parameter.

Once this is done, you can ask `Maude` to compute R', the delay $d(R, S)$ and so one.

Notice that `NC-maude` works with rationals, not floating point numbers. But you can convert from one to the other by the operator `float`.

2.3 Main network constructors

Here is a list of the main operators to build a network.

- `flow( _ , _ ) : String ArrivalCurve -> Flow` Build a flow from its name and its (maximal) arrival curve.
- `netElem( _ , _ ) : String ServiceCurve -> NetworkElement` Build a network element from its name and its (minimal) service curve.
- `netElem( _ , m: _ s: _ M: _ ) : String ServiceCurve ServiceCurve ServiceCurve -> Ne` a network element from its name, its minimal (`m:`), strict (`s:`) and maximal (`M:`) service curves.
- `netElem-strict( _ , _ ) : String ServiceCurve -> NetworkElement` Build a network element from its name and its strict service curve. Could be seen as a shortcut for `netElem( _ , m: Zero s: _ M: Delta 0 )`.
- `_ | _ : Flow NetworkElement -> Flow` Computes the output flow going through a network element .
- `sharedNE( _ , _ : _ ) : String ServiceCurve Set{Flow} -> SharedNetworkElement` Build a shared network element from its name, its (minimal) service curve and its inputs.
- `sharedNE( _ , m: _ s:_ M: _ : _ ) : String ServiceCurve ServiceCurve ServiceCurve SetFlow -> SharedNetworkElement` Like `sharedNE` and general `netElem`.
- `sharedNE-strict( _ , _ : _ ) : String ServiceCurve Set{Flow} -> SharedNetworkElemen` Like `sharedNE-strict` and general `netElem`.

- `_ ; _ : Path Path -> Path` Build a path of network elements.
- `shaper( _ , _ ) : String ShapingCurve -> Shaper` Build a shaper from its name and its shaper curve.
- `greedyShaper( _ , _ ) : String ShapingCurve -> GreedyShaper` Build a greedy shaper from its name and its shaper curve.
- `sharedGreedyShaper( _ , _ : _ ) : String ShapingCurve Set{Flow} -> SharedGreedySha` Build a shared greedy shared its name, its shaper curve and its input flows.

The main interesting bounds are

- `delay(_,_) : Flow NetworkElement -> Rat` Computes an upper bound on the delay of a flow through a network element.
- `buffer(_,_) : Flow NetworkElement -> Rat` Computes an upper bound on the buffer size used by a flow in a network element.
- `delay( _ ) : Path -> Rat` ,
  `sne-delay( _ ) : Path -> Rat` ,
  `g-delay( _ , _ ) : Flow Path -> Rat`,
  `g-sne-delay( _ , _ ) : Flow Path -> Rat`
  Different ways to compute the delay of a flow through a path.

2.4 Ploting

Ploting is based on the `gnuplot` tool. To plot curves, run a `gnuplot` shell and load the `min-plus.gnuplot` file[2]. In a `maude` shell, load the `gnuplot.maude` file, and use the function `gnuplot` on the expression. It returns an expression. You can copy/paste this expression into the `gnuplot` shell.

Here is a small example of interaction in the `maude` shell.

```
mboyer-laptop|~/NC-maude> maude -no-advise -no-banner startNC.maude
Maude> load  gnuplot.maude
Maude> red gnuplot( Gamma 3 4 , Beta 5 2 ) .
reduce in GNUPLOT : gnuplot(Gamma 3 4,Beta 5 2) .
rewrites: 44
result String: "plot [0:] Gamma( 3,4, x ) title 'gamma(3,4)' with lp ,
    Beta( 5,2, x ) title 'beta(5,2)' with lp"
Maude> red gnuplot( ( Gamma 1/4 8 /\ Gamma 2 1) ,
             Beta 5 1 ,
      [ ( Beta 5 1) - ( Gamma 1/4 8 /\ Gamma 2 1) ]+) .
reduce in GNUPLOT : gnuplot((Gamma 2 1 /\ Gamma 1/4 8),Beta 5 1,[Beta 5 1 - Gamma 2 1 /\ Gamma 1/4 8]+) .
rewrites: 148
result String: "plot [0:] min( Gamma( 2,1, x ),Gamma( 1/4,8, x )) title 'gamma(2,1) ^ gamma(1/4,8)'
   with lp , Beta( 5,1, x ) title 'beta(5,1)' with lp , max( Beta(5,1, x ) - min( Gamma( 2,1, x ),
   Gamma( 1/4,8, x )),Beta( 0,0, x )) title '[ beta(5,1) - gamma(2,1) ^ gamma(1/4,8) ]+' with lp"
```

The results can copied/pasted into a `gnuplot` shell, leading to plotting like the one of Figures 2.

[2]Or just run `gnuplot` with the command line `gnuplot min-plus.gnuplot -`.

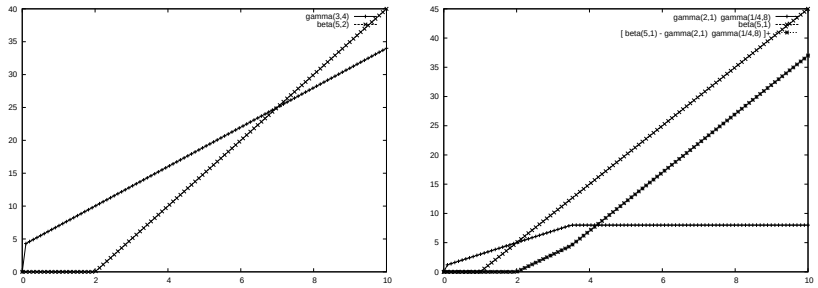

Figure 2: Example of plotting outputs

3 Others tools

They are several NC related tools, with different purposes.

The Real-Time Calculus (RTC) Toolbox [6] is a Matlab toolbox which allow the user to manipulate a certain kind of curve, and to make Modular Performance Analysis, a variation of Network Calculus.

Compared to RTC Toolbox, NC-maude is free (Matlab is not), open (you can manipulate your own kind of curve), and handle the classical network calculus, as described in [3].

The COINC software manipulates a wide class of piece-wise linear functions, and efficiently computes the common min-plus operations. The data representations and algorithms are presented in [**?**].

Compared to COINC software, NC-maude handles less functions, but is not limited to min-plus manipulation, and allow to handle topologies, to apply specific network calculus results. In a few word, the `MinPlus*` files does less things than COINC, but in a simplest way, and the `NC.maude` is original.

4 Why rewriting, and why `Maude` ?

I chose rewriting mainly because rewriting code is often very close to the mathematical definition of objects[3] and then, you can more easily trust the code[4].

But `Maude` in not the only rewriting tools: they are at least cafeOBJ, ELAN and TOM.

I chose `Maude` for some reasons:

1. `Maude` was recommended to me by colleagues,

2. `Maude` is efficient,

[3] Subtyping is really set inclusion, you can define mix-fix operators to write expressions like in mathematics, and computing is applying elementary rules, very often easy to verify one per one.

[4] And in fact, the mains bugs I had during development were term not reducing. In general, once passed the first unitary tests, I had rarely wrong results.

3. `Maude` comes with an interpreter, what exempt me for writing a parser,

4. `Maude` allows to define a sort in some module/file and to add new terms and subsorts in other one, a feature really important to develop and decompose a little large code (a few KLOC),

The drawbacks I have to face to are:

1. there is not `let` mechanism, to allow to locally define values and/or operators, and this is a drawback when code size increases;

2. rewriting is a very low level system, without any "modern" aspects like encapsulation (public interface/private implementation),

3. rewriting must be confluent, then, priorities between rules have no sense. Nevertheless, from efficiency point of view, it could have...

5 Naming convention

Most *logical* modules `M` are decomposed into two *maude* modules, one `M-SORT`, defining the sorts, the operators, and `M-IMPL` containing the equations.

6 Error handling

The noblest part of programming is designing algorithm, but the most common (and usefull) is error handling.

In imperative langages, error handling can be done with `assert`-like instuctions. But there is no instruction in maude, just typed expression, that get reduced, or not. And when there is a bug, in general, the expression does not get reduced.

Let us give an example: division by 0.

```
Maude> red 2 / 4 .
reduce in CONVERSION : 2/4 .
result PosRat: 1/2
Maude> red 2 / 0 .
reduce in CONVERSION : 2 / 0 .
result [Rat,FindResult]: 2 / 0
```

But, as in classical programming imperative langage, I find some bugs on some simples tests, all seems to be right, and when trying to reduce a real example, I get a 24-lines long term, and I try, with my tired eyes, to find which sub-term should have been reduced, and have not been.

So, my solution is to write code to detect pathological cases. And my solution is to have some dumy error functions. Such a function is an expression of the associated type, but including an error message. In the (minimal) following example, the division by 0 reduces into a "message" `error-Nat "Division by`

0". Without the rule div-by-0, and expression 1 / 0 just does not reduce. But now, it reduces into error-Rat "Division by 0".

```
fmod RAT-ERROR is
  protecting RAT .
  protecting STRING .
  op error-Rat _ : String -> Rat .
  eq [div-by-0] : n::Rat / 0 = error-Rat "Division by 0" .
endfm
```

This kind of mechanism is a way to encode exception in pure functional language, as presented in [5, § 14]. Of course, we could add a few more code to have more information on the context, like error-Rat "Division of 6 by 0".

A simple grep on expression error in the source file will show you real examples off this pattern.

7 Floating point version

NC-maude was designed to use infinite precision rationals (*i.e.* the sort Rat of Maude, since the first objective of NC-maude is truth and result confidence. Nevertheless, for performances reasons, we also tried to have a floating point version.

As the rational version is the main objective of the project, the floating point version is generated from the rational version, using scripts (sed mainly). The rational version have been modified as less as possible to allow this generation. The main difference is the introduction of a comparison operator = =, which is reduced into internal equality operator == in the rational version, and into a relative comparison in the floating point version, defined by $x = \ = x' \iff |x - x'| \leq \epsilon \max\{|x|, |x'|\}$ for rationals, and specific definitions for functions.

There also are some specific comments, ----FloatOnly to add some code in the floating point version, and couples ---NoFloatTest/---EndNoFloatTest to avoid some unitary tests in the floating pint version.

To generate the floating point version, use the script makeReal.sh. It generates a sub-directory Maude-Float with the floating point version.

8 Encapsulation and late-binding in Maude

Rewriting is a very powerfull but low-level programming langage. There is no "modern" feature like encapsulation, late-binding, popular in object-oriented languages.

There is, of course, some "Object-Based Model": the module CONFIGURATION in the prelude of Maude), with sorts Object, Oid, Cid) considering object-oriented programming as a set of objects exchanging messages in an undeterministic concurrent way (see [2, Chapter 8]).

Its is not the scope of this paper to say what is real object-oriented model. It focuses on two features absent of the `Maude` object model: encapsulation and late binding.

The first question is: did we need such features? My personal experience, after about 4.5K lines of `Maude` is that they are.

8.1 Encapsulation

First, encapsulation is an important feature since the representation of some object can change during the project life. A common example is the two dimentionnal point. A first implantation could be to have simply an `x` and `y` value. In C langage, a thing like `struct Point { float x,y; }`. In `Maude`, a constructor `point(_,_): Float Float -> Point`. But when project grows, the need of being able to manipulate polar coordinates could arise. In class based object oriented langages (like the populars C++ and Java), it is often done with an abstract interface, just defining the public methods of the class, and some private internal representation (with some `implements` or `extends` keyword). In `Maude`, we can do the same. The interface is the type `Point` and a set of methods like `getX(_) : Point -> Float`, `setX(_,_): Point Float -> Point`. (Tanks to the very free syntax of `Maude`, even some popular postfix notation `_.x : Point -> Float` can be used.) Then, an implementation of such a Point class can be done as follow.

```
fmod POINT is
  protecting FLOAT .
  protecting STRING .
  protecting CONVERSION .

  sort Point .

  op point( _ , _ ) : Float Float -> Point [ctor] .
  ops _.x _.y : Point -> Float .
  op print( _ ) : Point -> String .

  vars i j : Float .
  eq [pt-x] : point( i , j ).x = i .
  eq [pt-y] : point( i , j ).y = j .

  var pt : Point .
  eq [pt-print] :
     print( pt ) = "(x:" + string( (pt).x  )
                 + ",y:" + string( (pt).y  )  + ")" .
endmf
```

Then, if the need of mixed representation cartesian and polar of point is needed, the canonical representation is no more unique, some new constructor `op polPoint(_,_) : Float Float -> Point` can be added, some meth-

ods have to be changed, but the one based on the interface (like the `print`, are unchanged.

You may thing this never happen. My experience with `NC-maude` is different. In one version, I have added a source to each flow, to be able to group flows going from a common server. And some revisions after, I decide to remove it. The same, I have to add minimal frame size to flows. When I had to make these changes, I would have to update all equations involving the server sort, and I only have to update the one not using the interface methods.

8.2 Dynamic binding

The second important feature from the object-oriented world is the "dynamic binding" (also known as "late binding" or "open recursion" [5]) as is related to sub-typing (also known as "inheritance" in OOP).

Sub-typing is integrated in `Maude`. A sub-type is just a sub-sort. Dynamic binding is the feature that allow a same method to have different implementations for a type and a sub-type, *and* to select the more specific one.

Did we need it? Yes for `NC-maude`. We could have different rules to compute some upper bound on a system, and the more information we have (ie the more precise sub-type we manipulate), the more precise results we get.

To come back to our `Point` example, we could have a `NamedPoint`, which is a `Point` with a name, and we want the method `print` to print the coordinates *and* the name.

Such a subsort could be encoded as follow.

```
fmod NAMED-POINT is
  protecting POINT .
  sort NamedPoint .
  subsort NamedPoint < Point .

  op namedPoint( _ , _ , _) : Float Float String -> NamedPoint [ctor] .
  op super( _ ) : NamedPoint -> Point .
  op _.name : NamedPoint -> String .

  vars i j : Float .
  var n : String .
  var npt : NamedPoint .

  eq [npt-super] : super( namedPoint( i , j , n) ) = point( i , j ) .
  eq [npt-x] : (npt).x = super( npt ).x .
  eq [npt-y] : (npt).y = super( npt ).y .
  eq [npt-name] : namedPoint( i , j , n ).name = n .

  eq [npt-print] : print( npt ) = "[" + (npt).name + "]" +  print( super(npt) ) .
endfm
```

The use of the operator `super` allows to quickly write the methods `_.x` and `_.y`. Nevertheless, if any `setX` method exists, its overloading it not so simple.

With this implementation, we are able to manipulate a `NamedPoint` in equations written for `Point`, but your theory is no more confluent (equations `pt-name` and `npt-name` can be applied to a `NamedPoint`, with different results.

Let us first have a look on what works: the use of a `NamedPoint` in place of a `Point`. Let us consider a simple module which uses `Point`: the module `DISTANCE` which compute the distance between two point and also print it.

```
fmod DISTANCE is
  protecting POINT .

  op distance( _ , _ ) : Point Point -> Float .
  op printDist( _ , _ ) : Point Point -> String .

  vars pt pt' : Point .
  eq [dist] :
     distance(  pt , pt' ) =
      sqrt( ( ( (pt).x - (pt').x ) ^ 2.0 )
          + ( ( (pt).y - (pt').y ) ^ 2.0 ) ) .

  eq npt = namedPoint( 2.0 , 2.0 , "npt" ) .

  eq npt = namedPoint( 2.0 , 2.0 , "npt" ) .
  eq [print-dist] :
    printDist( pt , pt' ) =
     "The distance from "  +
     print( pt ) + " to " +
     print( pt' ) + " is " +
     string( distance( pt , pt' ) ) .
endfm
```

Then, we can compute the distance between a `Point` and a `NamedPoint`, even if the `DISTANCE` module does not know anything about the `NamedPoint` sort. Let us consider the constants `p1`,`p2` and `npt` defined in the test module `TEST-PT`.

Then, you can ask `Maude` to reduce the term `distance(p1,npt)` and also between `p1` and `p2`, with `p2` defined as a `Point` and being a `NamedPoint`.

```
fmod TEST-PT is
  protecting NAMED-POINT .
  protecting DISTANCE .

  ops p1 p2 : -> Point .
  op npt : -> NamedPoint .

  eq p1 = point( 1.0 , 1.0 ) .
```

```
  eq npt = namedPoint( 2.0 , 2.0 , "npt" ) .
  eq p2 = namedPoint( 0.0 , 0.0 , "p2" ) .
endfm
```

Now, let us turn to the dynamic binding. When printing the distance between a Point and a NamedPoint, the "method" `print` is called, that is to say, the term `print(npt)` have to be reduced. And two equations could apply, `pt-print` and `npt-print`, with different results. Our theory is not confluent, and the `Maude` tool will chose one or the other (depending on some internal implementation choice).

Now, we would like to have dynamic binding, ie to force the equation `npt-print` to be used.

The first idea could be to forbid the applying of the equation when the argument is a `NamedPoint`, and write the rewrite the `pt-print` equation as follow:

```
  ceq [pt-late-print] :
      print( pt ) = "(x:" + string( (pt).x  )
                  + ",y:" + string( (pt).y  ) + ")"
 if not( pt :: NamedPoint)  .
```

But it would imply to know, when writing the `POINT` module, all its sub-sorts. Another trick could be to use the `owise` attribute, to allow to use this equation only if no other (ie more specific) matches. But this would only works with a two level hierarchy[5].

Our solution is to introduce an "intermediary" sort, `SubPoint`, and to ask sort/class that want to "inherit" from `Point` (and to benefit of the dynamic binding) to be a sub-sort of this sort, and to rewrite `pt-late-print` with the test `pt ::  SubPoint` .

The sorts definitions in module `POINT` becomes

```
  sorts Point  SubPoint .
  subsort SubPoint < Point .
```

The correct implantation of the dynamic binding on method `print` is:

```
  ceq [pt-late-print] :
      print( pt ) = "(x:" + string( (pt).x  )
                  + ",y:" + string( (pt).y  ) + ")"
      if not( pt :: SubPoint)  .
```

We can now ask to print the distance from `p1` to `p2` and gets the result

```
result String: "The distance from (x:1.0,y:1.0) to [p2](x:0.0,y:0.0)
  is 1.4142135623730951"
```

[5]It works when adding a third sort/class `ColoredPoint`, but not when adding a forth sort/class `NamedColoredPoint`.

9 Modules and files

File RatInf.maude

RAT-INF Definition of the rational numbers union infinites. Natural overloading of `+`, `-`, `min`, `max`.

File RealFct.maude

REAL-FCT Define the sort `RealFct`, the set of functions from $\mathbb{R}$ to $\mathbb{R}$ and some "classical" operators on function, addition (`+`), minus (`-`), multiplication (`*`), composition (`comp`) and comparisons (`<=,>=`) and application (`[_]`).

Because there is no real numbers in computers, rationals ($\mathbb{Q}$, the maude sort `Rat`) are used.

RAT-MIN and REAL-FCT-MIN Introduction of notations `/\` and `\/` for min and max.

File MinPlus.maude The core of the (min,+) diod

G-FCT Definition of the G set (`G` sort), the set of non-negative wide-sens increasing functions: `subsort G < RealFct .` Natural overloading of `+`, `/\`, `\/`, `comp`.

F-FCT Definition of the F set (`F` sort), the set of G functions nul on $]-\infty, 0[$. Natural overloading of `+`, `/\`, `\/`, `comp`. Definition of the pseudo-inverse operator $\cdot^{-1}$ (`^-1`), the `Zero` function and the $[\cdot]^+$ (`[_]+`) operator.

ERROR Help for error detection.

BASIC-MIN-PLUS-FCT Definition of some well known functions: δ_d (`Delta`), λ_r (`Lambda`), $\beta_{R,T}$ (`Beta`), $\gamma_{r,b}$ (`Gamma`), $v_{T,\tau}$ (`StairCase`). Some properties on the defined operators are also given (pseudo-inverse, composition, min, max, etc.).

MIN-PLUS-CONV-DECONV Introductions of the convolution ($\otimes$, `conv`) and deconvolution ($\oslash$, `deconv`) operators, and some known results on well known functions.

MIN-PLUS-GOOD-FCT Introduction of the "good functions" (`GoodFct`) and "sub-additive functions" (`SubAdditive`) sorts. Definition of the sub-additive closure operator ($\overline{f}$, `subAddClos`), and the sort of functions whose sub-additive closure is computable by our code (`SubAddClosComputable`). Some operators are overloaded in these new sort, and the membership of `Gamma` to `GoodFct` is added.

DEVIATIONS Introduction of the Vertical and Horizontal Deviations between the graphs of two curves f and g of sort `F`.

MIN-PLUS-FCT Enable some simplifications on "SubAddClosComputable functions" especially thanks to the normal form.

File MinPlus-Gamma.maude

MIN-GAMMA-FCT-SORTS Definition of the "gamma-min functions" (`sort GammaMin < SubAddClosComputable`) and the "gamma-min in normal form functions" (`sort GammaMin-NormalForm < GammaMin`).It stands for respectively, functions which are the minimum of gamma functions and these same fonctions put in normal form. We also define the operator `NormalForm` which put a "gamma-min function" in normal form.

This module contains very few equations: it defines sorts and operators, whose (efficient) implementation must be done in other modules. In fact, the implementations are done in `MinPlus-DoubleGamma.maude`, `MinPlus-GammaMin.maude` and `MinPlus-GammaCPL.maude`. You can load either one or the other. See section 10 for details.

File MinPlus-DoubleGamma.maude

MIN-GAMMA-FCT-IMPL-SIMPLE A basic implementation of MIN-GAMMA-FCT gathering the rules with the minimum of two gamma functions which compute the deconvolution and normal form.

MIN-GAMMA-FCT Wrapper used to have a common interface. See 10 for details.

File MinPlus-GammaMin-NF.maude
An *internal* module, used by other modules to compute normal form of minimum of Gamma functions.

File MinPlus-GammaMin.maude

MIN-GAMMA-FCT-IMPL-GENERAL A more general implementation of MIN-GAMMA-FCT using sequential and recursive code to compute deconvolution and normal form of minimum of gamma functions ($\bigwedge_i \gamma_{r_i,b_i}$). As maude is not made for this, it is quit boring and not easy to understand.

MIN-GAMMA-FCT Wrapper used to have a common interface. See 10 for details.

File MinPlus-GammaCPL.maude
TBD

MIN-GAMMA-FCT-IMPL-SIMPLE An efficient implementation to compute deconvolution and normal form of minimum of gamma functions ($\bigwedge_i \gamma_{r_i,b_i}$). It maintains the normal form of [1]. See 10 for details.

MIN-GAMMA-FCT Wrapper used to have a common interface. See 10 for details.

File NetworkUnits.maude

NET Definition of units used in Network such as time units, size units and rate units. There is also conversion operators.

File NC.maude

NC-SIMPLE Definition of the flow sort (`Flow`) with arrival curve sort (`subsorts F < ArrivalCurve`), NetworkElement sort (`NetworkElement`) with ServiceCurve sort (`subsorts F < ServiceCurve`). We also define the two classical operators in network. First delay (`delay`) which takes in parameter a flow and a network element and return the maximum delay for the flow to pass through the element. Then buffer (`buffer`) which takes the same parameters as delay and return the minimal size of the buffer not to loose PDU.

NC-SHARED It defines what is a shared network element (`subsort SharedNetworkElement < NetworkElement`). Instead of having only one fow in input, it has a list of flows. The operator sne-delay (`sne-delay`) is the intrasec delay of a shared network element. It takes in input a shared network element.

NC-PATH Definition of a path (`subsorts NeListNetworkElement < Path`) which is a non empty list of Network element. We have overloaded delay as well.

NC-SHAPER Definition of greedy shaping (`subsorts GreedyShaper < NetworkElement`) with Shaping Curve functions (`subsorts F < ShapingCurve`).

NC-GROUPS (under development) This module is under development and unstable. It is designed to test some efficient policies of grouping.

File startNC.maude Entry point of the project: load the necessary files for common usage.

Files makeReal.sh rat2float.sed These files transform the Maude code using rational numbers in a Maude code using Float.

10 Choosing a $\bigwedge_i \gamma_i$ implementation

10.1 Three implementations

There are three implantation to handle $\bigwedge_i \gamma_i$ functions. The first one, `MinPlus-DoubleGamma.maude` only accepts terms of size two, but is easy to understand. It can be read (as exercise) but not be used. The second one `MinPlus-GammaMin.maude` is produces results easy to read, handle any size, but is not very efficient. The third one, `MinPlus-GammaCPL.maude`, is efficient, but could be a little bit suprising at first use.

A little example First, we can to compute the sum $(\gamma_{2,1} \wedge \gamma_{1,3}) + (\gamma_{2,1} \wedge \gamma_{1,2})$.

Using `MinPlus-GammaMin.maude`, the input and output are very close to the mathematical notations. It should produce something like that [6]

```
% maude -no-banner MinPlus-GammaMin.maude
Maude> red  (Gamma 1 2 /\ Gamma 2 1) + (Gamma 1 3 /\ Gamma 2 1) .
...
result GammaMin: Gamma 2 5 /\ Gamma 3 3 /\ Gamma 4 2
```

The reader should notice that the order `Gamma 2 5 /\ Gamma 3 3 /\ Gamma 4 2` is an internal choice of `Maude` implementation. The operator `_ /\ _` is declared associative and commutative.

On the opposite, the implementation in `MinPlus-GammaCPL.maude` keep such terms as a sorted list of `Gamma` terms. With this implementation, you have to build the sorted list with the `mkCpl` operator. You should not directly enter a `cpl::` term, since the argument *must* have some properties[7], otherwise, the module should produce wrong results.

```
% maude -no-banner MinPlus-GammaCPL.maude
Maude> red (mkCpl(Gamma 1 2 /\ Gamma 2 1) )
        + ( mkCpl(Gamma 1 3 /\ Gamma 2 1) ) .
...
result CPL: cpl:: Gamma 4 2 : Gamma 3 3 : Gamma 2 5
```

10.2 What is the efficiency problem?

The efficiency problem of `MinPlus-DoubleGamma.maude` comes from the difference between mathematics and computer science. The implementation of $\bigwedge_i \gamma_i$ in `MinPlus-GammaMin.maude` is mathematically-written.

To efficiently handle such kind of curves, it must be ordered and without any useless term. But, the operator $\wedge$ is declared has commutative. So, it is hard to keep is sorted. But even the issue of useless term is hard. How to keep trace that there is no useless term in such a term?

[6] The dots are some warnings, performances results, etc., The `-no-banner` option is just there to avoid the welcome message of `Maude`.

[7] To be on the normal form of [1].

In the implementation of `MinPlus-GammaMin.maude`, terms are put under normal form only when is it really necessary (to compute deconvolution by example).

10.3 How to chose one or the other?

You have to load either the `MinPlus-DoubleGamma.maude` file and to include the `MIN-GAMMA-FCT-IMPL-SIMPLE` module, or the `MinPlus-GammaMin.maude` file and to include the `MIN-GAMMA-FCT-IMPL-SIMPLE` module.

Warning: to handle the full min-plus code, you have in you module to include MIN-PLUS-FCT *and* one implementation.

11 Membership, overloading, sort and kind level...

When loading `NC-maude`, the `Maude` interpreter will complains with messages like:

```
Warning: membership axioms are not guaranteed to work correctly for associative symbol _
    declarations that are not at the kind level.
```

What does it mean? This problem is presented in [2, § 14.2.8]. Let illustrate it with a little example: the sum operator and two sorts, the general functions ($\mathcal{F} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$), and the wide-sense increasing functions ($\mathcal{I} = \{f \in \mathcal{F} \,|\, x \leq y \implies f(x) \leq f(y)\}$). The sum is associative and commutative on $\mathcal{F}$, and it is stable on $\mathcal{I}$. It can have two declarations. The first one could be

```
fmod F-I is
  sorts F I .
  subsort I < F .
  op _ + _ : F F -> F [assoc comm] .
  op _ + _ : I I -> I [ditto] .
endfm
```

But this will produce a warning. The reason is that, as presented in [2, § 14.2.8], membership equations are not, for performance reasons, applied to all subterms. This implies that the computed sort is not the minimal sort (but still a valid sort, or kind).

The solution is to use kind level declaration, associated with membership equations.

```
fmod F-I is
  sorts F I .
  subsort I < F .
  op _ + _ : F F ~> F [assoc comm] .
  mb f:F + f':F : F .
  mb f:I + f':I : I .
endfm
```

Nevertheless, kind level declaration and membership equation can lead to big computation times (seems to be exponential). The problem is illustrated in a simple example in file `perf-AssocComMB.maude`. Then, since the effect is not false result, but only not as true as possible (the least sort is not computed, but a valid sort or kind is still produced), we chose to keep overloading (and warning).

12 On strategies, lazy evaluation, performances and readability

To be able to manage topologies, each flow encodes its source, and each shared network element encode its input. The source of a flow `fl | ne1 | ne2` is `fl | ne1`. But such expression will be reduced to its normal form, which could be very large, if for example, `ne1` is a shared element, with a large input set. Moreover, such large term can be costly to compute.

At first glance, for computing efficiency and for readability reason, a lazy strategy on topology encoding could be a good idea.

Moreover, such encoding will forbid to encode any cycle, since a loop in the topology will produce a computation loop. Even if loop handling is not a short term objective, it could be in one future...

The basic idea is to use the strategy possibilities of `Maude`, *i.e.* the `strat` attribute, to avoid to evaluate the topology information on flow and shared network elements.

Nevertheless, lazy evaluation means that the normal form is no more the same. And it means that the build-in equality operator will consider different some terms which are equals, up to eager evaluation.

The solution adopted in `NC-maude` consist in writing a dedicated equality operator, `=nf=`, which compare a normal form defined by the user.

For example, two flows could be considered equals if they have the same name, the same arrival curve, and the same source. But since the source could be a shared network element, the "same" notion must be implemented by a `=nf=` test...

13 Directories

Maude-Float Contains all the Maude files transformed from Rational numbers to Float.

TestsUnitaires Contains a groupe of unit tests on the code using rational numbers and the general implementation of MIN-GAMMA-FCT.

TestsUnitairesDoubleGamma Contains a groupe of unit tests on the code using rational numbers and the simple implementation of MIN-GAMMA-FCT.

doc Contains the documentation on the project.

References

[1] Marc Boyer and Christian Fraboul. Tightening end to end delay upper bound for AFDX network with rate latency FCFS servers using network calculus. In *Proc. of the 7th IEEE Int. Workshop on Factory Communication Systems Communication in Automation (WFCS 2008)*, pages 11–20. IEEE industrial Electrony Society, May 21-23 2008.

[2] Manuel Clavel, Francisco Durán, Steve Eke, Patrick Linkoln, Martí-Oliet Narciso, Jos Meseguer, and Corolyn Talcott. *Maude Manual*, october 2008.

[3] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*, volume 2050 of *LNCS*. Springer Verlag, 2001. http://lrcwww.epfl.ch/PS_files/NetCal.htm.

[4] The Maude system home page. http://maude.cs.uiuc.edu/.

[5] Benjamin C. Pierce. *Types ad Programming Languages*. MIT Press, 2002.

[6] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

A Map of specific class of functions

In this section, we are going to sum up some results on specific class of functions, presented in [3] This is particularly usefully for `NC-maude` since is make a intense use of sort and sub-sorts and related properties.

We are going to consider five specific sets of functions.

$\mathcal{G}$ is the set of wide-sense increasing functions from $\mathbb{R}$ to $\mathbb{R} \cup \{\infty\}$

$\mathcal{F}$ is the subset of $\mathcal{G}$ nul (strictly) before 0

$\mathcal{F}_0$ is the subset of $\mathcal{G}$ up to 0 (included)

$\mathcal{F}_{\mathbf{sa}}$ is the subset of $\mathcal{F}$ of sub-additive functions

$\mathcal{F}_{\mathbf{ssh}}$ is the subset of $\mathcal{F}$ of star-shaped functions, *i.e.* such that $\frac{f(t)}{t}$ is wide-sense decreasing

Good the set of good functions is the set of sub-additive functions nul at 0 [3, Definition 1.2.4, Corolary 3.1.1]

Conc is the subset of concave function in $\mathcal{F}$

$$\mathcal{F} \stackrel{\text{def}}{=} \{f \in \mathcal{G} \,|\, x < 0 \implies f(x) = 0\} \tag{1}$$

$$\mathcal{F}_0 \stackrel{\text{def}}{=} \{f \in \mathcal{F} \,|\, f(0) = 0\} \tag{2}$$

$$\mathcal{F}_{\text{sa}} \stackrel{\text{def}}{=} \{f \in \mathcal{F} \,|\, \forall x, y \in \mathbb{R}, f(x+y) \leq f(x) + f(y)\} \tag{3}$$

$$\mathcal{F}_{\text{ssh}} \stackrel{\text{def}}{=} \left\{ f \in \mathcal{F} \,\middle|\, \forall 0 < x < y, \frac{f(x)}{x} \geq \frac{f(y)}{y} \right\} \tag{4}$$

$$\mathit{Good} \stackrel{\text{def}}{=} \mathcal{F}_0 \cap \mathcal{F}_{\text{sa}} \tag{5}$$

$$\mathit{Conc} \stackrel{\text{def}}{=} \{f \in \mathcal{F} \,|\, \forall x, y \in \mathbb{R}, \forall u \in [0,1], f(ux + (1-u)y) \geq uf(x) + (1-u)f(y)\} \tag{6}$$

The main relations between these sets are presented in Figure 3.

Let us start with $\mathcal{F}_0$ and $\mathcal{F}_{\text{sa}}$. By definition, their intersection is the set of good functions *Good*.

Now, let us introduce the star-shaped functions ($\mathcal{F}_{\text{ssh}}$). From [3, Theorem 3.1.9], any star-shaped function nul at 0 is sub-additive. In fact, any star-shaped function is sub-additive, even f not nul at 0.

That is to say, $\mathcal{F}_{\text{ssh}} \subset \mathcal{F}_{\text{sa}}$. It obviously come $(\mathcal{F}_0 \cap \mathcal{F}_{\text{ssh}}) \subset \mathit{Good}$.

Lemma 1 (Star-shaped non negative functions are sub-additive)**.** *Any star-shaped function is sub-additive.*

$$\mathcal{F}_{ssh} \subset \mathcal{F}_{sa} \tag{7}$$

Proof. Let be t, s two positive reals. Assume without loss of generality that $t \geq s$. We of course have $t + s \geq t \geq s$. From $t + s \geq t$, and start-shaped property, it comes $\frac{f(t+s)}{s+t} \leq \frac{f(s)}{t}$, equivalent to $f(t + s) \leq f(t) + \frac{s}{t} f(t)$. And from $t \geq s$ and start-shaped property, we have $\frac{f(t)}{t} \leq \frac{s}{s} \iff \frac{t}{s} f(t) \leq f(s)$. It follows $f(t + s) \leq f(t) + f(s)$.

Now, if $t \geq 0$ and $s = 0$: $f(t + s) = f(t) \leq f(t) + f(0)$. □

The last set of interest is *Conc*, the set of concave functions. From [3, Theorem 3.1.4], it comes $Conc \subset \mathcal{F}_{\text{ssh}}$. But we also have $Conc \subset \mathcal{F}_{\text{sa}}$. This result also is not in [3], but obvious to proof.

Lemma 2 (Positive increasing concave functions are sub-additive)**.** *Any positive, wide-sense increasing concave function is sub-additive.*

$$Conc \subset \mathcal{F}_{sa} \tag{8}$$

Proof. The property holds for any positive, wide-sense increasing concave, *nul at 0* (a concave function is star-shaped, and any star-shaped function nul at 0 is sub-additive [3, Theorem 3.1.9]).

Consequently, let us consider $f \in \mathcal{F}$ (positive, wide-sense increasing), concave, with $f(0) \neq 0$. Let be $g(x) = f(x) - f(0)$. We have $g \in \mathcal{F}_0$, g concave. It implies g is sub-additive. Now, the sub-additivity of f can be proved:

$$\begin{aligned} f(x + y) &= g(x + y) + f(0) \\ &\leq g(x) + g(y) + f(0) && g \text{ is sub-additive} \\ &\leq g(x) + g(y) + 2f(0) && \text{because } f(0) \geq 0 = f(x) + f(y) \end{aligned}$$

□

This gives the overall picture of Figure 3.

This map can be refined, showing the non emptiness of areas (1) up to (7).

(1) a good function, not star-shaped: the ceil function, $\lceil x \rceil$.

Proof. The ceil function is the "next integer", *i.e.* the only integer function such that $\lceil x \rceil - 1 < x \leq \lceil x \rceil$.

It is null at 0 (obvious).

It is sub-additive: $x \leq \lceil x \rceil$ and $y \leq \lceil x \rceil$, then $x + y \leq \lceil x \rceil + \lceil y \rceil$. It is wide sense increasing: $\lceil x + y \rceil \leq \lceil \lceil x \rceil + \lceil y \rceil \rceil = \lceil x \rceil + \lceil y \rceil$.

It is not star shaped: just (gnu)plot it, or consider $\frac{\lceil 2/3 \rceil}{2/3} = \frac{3}{2} = 1.5$ and $\frac{\lceil 3/2 \rceil}{3/2} = \frac{4}{3} \approx 1.333$. □

(2) a sub-additive function, not nul at 0 neither star-shaped: the ceil function, $\lceil x \rceil + 1$.

(3) a star-shaped function (then sub-additive), not nul at 0 neither concave: $\beta_{R,T} + K$, with $0 < RT < K$.

Proof. This example comes from [3, § 3.1.7, Fig. 3.5]. It is clearly not concave (with $x = 0, y = 2T, f(x) = K, f(2T) = K + RT, u = \frac{1}{2}, f(ux + (1-u)y) = f(T) = K, uf(x) + (1-u)f(y) = uK + (1-u)(K + RT) = K + uRT$), not nul at 0. It is sub-additive (from [3, § 3.1.7]).

It is star-shaped $\frac{f(t)}{t}$ is decreasing on $[0, T]$ and constant on $[T, \infty[$. □

(4) a star-shaped function, sub-additive, nul at 0, but not concave: $\overline{\beta_{R,T} + K}$, with $0 < RT$.

Proof. Just take the sub-additive closure of the counter example of the two examples [3, § 3.1.8, Fig. 3.6]. Notice that a sub-additive closure is always nul at 0 ($\overline{f} = \delta_0 \wedge \ldots$). □

(5) a concave function, sub-additive, not nul at 0: $f(x) = ax + b$, $a > 0$, $b > 0$.

(6) concave functions, sub-additive and nul at 0: $\gamma_{r,b} = (ar + b) \wedge \delta_0$, $\bigwedge_i \gamma_{r_i,b_i} \ldots$

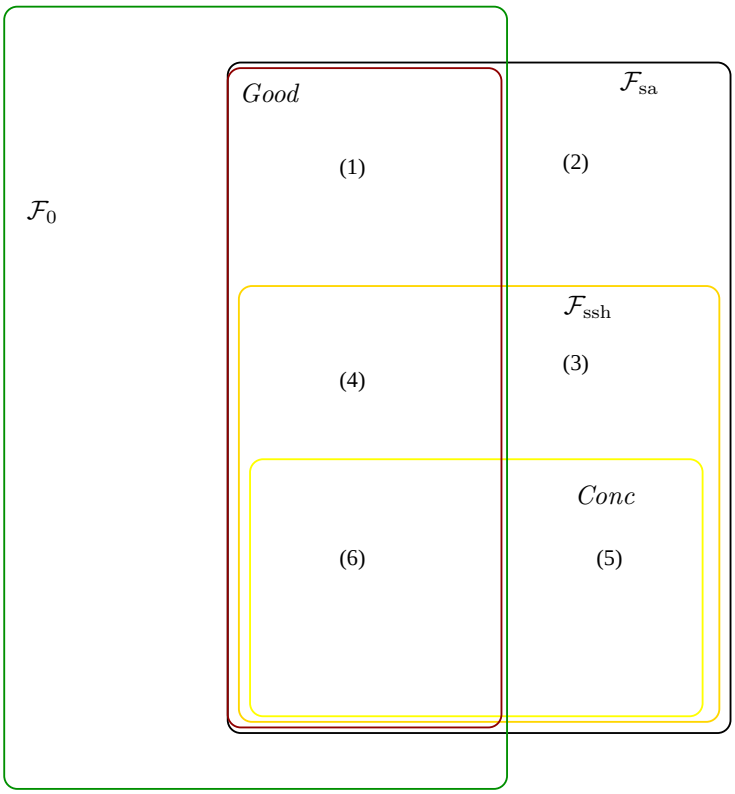

Figure 3: Map of some interersting subsets of $\mathcal{F}$