Nancy.Expressions
A library for automated optimizations

IACOPO CANETTA, ANTONIO A. SALVALAGGIO, ANDREA TRASACCO,
RAFFAELE ZIPPO, GIOVANNI STEA
UNIVERSITY OF PISA

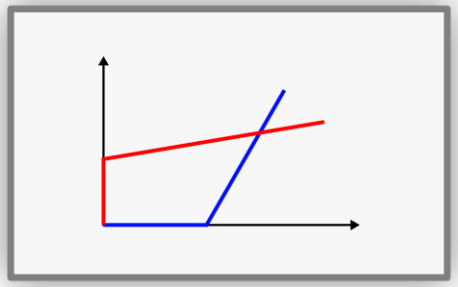
Nancy.Expressions

A library for computation of Deterministic Network Calculus *expressions*

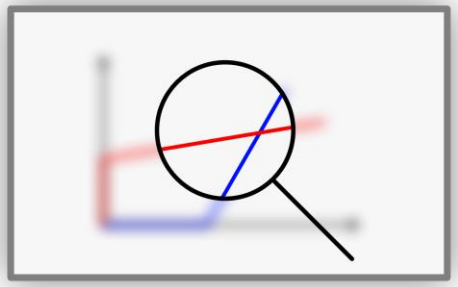
Design and implementation by **Andrea Trasacco** for his M.Sc. Thesis

Presented at VALUETOOLS 2024 (proceedings pending) as
Nancy.Expressions: Towards a CAS for DNC – A. Trasacco, R. Zippo, G. Stea

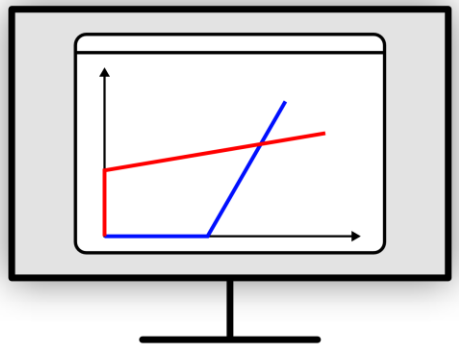
Packages, code, documentation already available on Nuget and GitHub



For Teaching



For Research



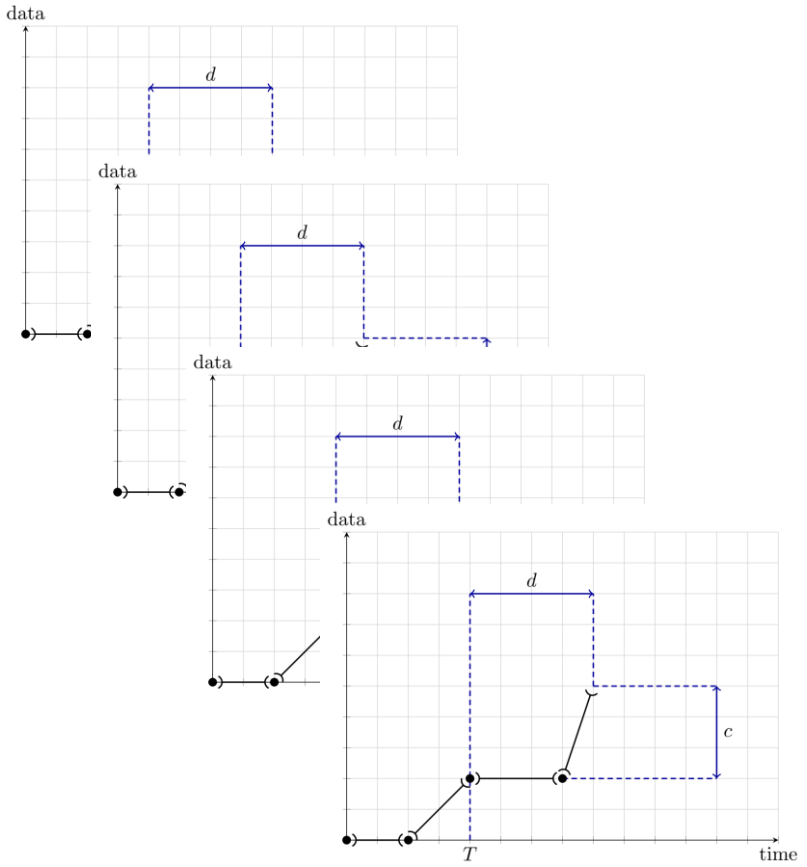
For Applications

A simple problem, using Nancy

$$(\beta_1 - \alpha_1)^+_\uparrow \otimes (\beta_4 - ((\alpha_3 + \alpha_4) \oslash \beta_3))^+_\uparrow$$

[BBL18]

```
var left = Curve.Subtraction(beta1, alpha1)
                .ToNonNegative()
                .ToUpperNonDecreasing();

var tmp = Curve.Addition(alpha3, alpha4)
               .Deconvolution(beta3);
var right = Curve.Subtraction(beta4, tmp)
                 .ToNonNegative()
                 .ToUpperNonDecreasing();

var conv = Curve.Convolution(left, right);
```



A simple problem, using Nancy

$$(\beta_1 - \alpha_1)^+_\uparrow \otimes (\beta_4 - ((\alpha_3 + \alpha_4) \oslash \beta_3))^+_\uparrow$$

[BBL18]

```
var left = Curve.Subtraction(beta1, alpha1)
                .ToNonNegative()
                .ToUpperNonDecreasing();

var tmp = Curve.Addition(alpha3, alpha4)
               .Deconvolution(beta3);
var right = Curve.Subtraction(beta4, tmp)
                 .ToNonNegative()
                 .ToUpperNonDecreasing();

var conv = Curve.Convolution(left, right);
```

Some undesirable things:

1. Debugging using only the resulting curve, not how are they computed
2. Code *is* computation strategy
   Trade-off between readability/simplicity and clever optimizations
3. Some models have *options*, like optimizable *parameters*
4. The above problems scale with program complexity – we usually *compose* studies on networks

The same problem, using Nancy.Expressions

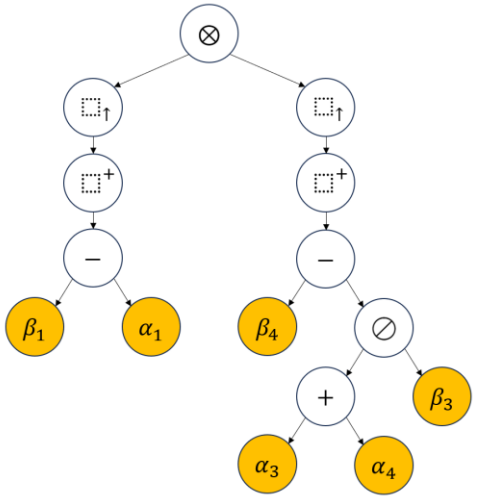
$$(\beta_1 - \alpha_1)^{+}_{\uparrow} \otimes (\beta_4 - ((\alpha_3 + \alpha_4) \oslash \beta_3))^{+}_{\uparrow}$$

[BBL18]

```
var left = Expressions.Subtraction(beta1, alpha1)
                      .ToNonNegative()
                      .ToUpperNonDecreasing();

var tmp = Expressions.Addition(alpha3, alpha4)
                     .Deconvolution(beta3);
var right = Expressions.Subtraction(beta4, tmp)
                       .ToNonNegative()
                       .ToUpperNonDecreasing();

var conv = Expressions.Convolution(left, right);
var result = conv.Compute();
```

$\mathcal{O}(1)$

Only constructs the *expression tree*



All computations happen here

Nice-to-haves

1. *Debugging using only the resulting curve, not how are they computed*

   Can print out the expression and its result, separately

```
var subLeft = Expressions.Subtraction(beta1, alpha1)
    .ToUpperNonDecreasing()
    .ToNonNegative();
var deconv = Expressions.Addition(alpha3, alpha4).Deconvolution(beta3);
var subRight = Expressions.Subtraction(beta4, deconv)
    .ToUpperNonDecreasing()
    .ToNonNegative();
var e2eServiceCurve2Exp = Expressions.Convolution(subLeft, subRight);
display(e2eServiceCurve2Exp);
```

[37] ✓ 0.1s

$$[(\beta_1 - \alpha_1)]^+_\uparrow \otimes [(\beta_4 - ((\alpha_3 + \alpha_4) \oslash \beta_3))]^+_\uparrow$$

Nice-to-haves

2. *Code is computation strategy. Trade-off between readability and optimizations*

In Nancy.Expressions, most of the program builds the expressions to be computed

We can implement many optimizations as transformations or alternative Compute() algorithms

Only the last part of the program needs to be altered to take advantage of these

Examples in the second part of the presentation

Nice-to-haves

3. *Some models have* options*, like optimizable* parameters

Expressions can be transformed by replacing sub-expressions

$$\beta_i^\theta = \left[\beta - \sum_{j \neq i} \alpha_j * \delta_\theta\right]^+ \wedge \delta_\theta$$

FIFO residual s.c. [BBL18]

$\forall \theta \geq 0, \beta_i^\theta$ is a s.c.

```
var delta_theta_0 = Expressions.FromCurve(new
 DelayServiceCurve(0));

var b_theta_0 = Expressions.Subtraction(
        beta,
        Expressions.Addition(contendingFlows)
            .Convolution(delta_theta_0))
        .ToNonNegative()
        .Minimum(delta_theta_0);

var wcd_0 = Expressions.HorizontalDeviation(foi,
b_theta_0).Compute();
```

```
var delta_theta_1 = Expressions.FromCurve(new
 DelayServiceCurve(1));

var b_theta_1 = b_theta_0.ReplaceByValue(
          delta_theta_0, delta_theta_1);

var wcd_1 = Expressions.HorizontalDeviation(foi,
b_theta_1).Compute();
```

Nice-to-haves

4. *The problems scale with program complexity – we* compose *network studies*

We converted a tool using Nancy (from NPBA repo) to Nancy.Expressions.

It was straightforward, requiring little understanding of the underlying logic

```
// Using Nancy
public static Curve MakePeriodicWorstCaseArrivalCurve(Flow flow)
//Theorem 1
{
    var linkSpeed = flow.Path[0].LinkSpeed;
    var m = flow.MaxIntervalFrame * flow.MaxFrameSize * 8;
    var linkSpeedCurve = new SigmaRhoArrivalCurve(0, linkSpeed);
    var stair = new StairCurve(m, flow.ClassMeasurementInterval);
    return Curve.Convolution(stair, linkSpeedCurve);
}
```

```
// Using Nancy.Expressions
public static CurveExpression MakePeriodicWorstCaseArrivalCurve(Flow flow)
//Theorem 1
{
    var linkSpeed = flow.Path[0].LinkSpeed;
    var m = flow.MaxIntervalFrame * flow.MaxFrameSize * 8;
    var linkSpeedCurve = new SigmaRhoArrivalCurve(0, linkSpeed);
    var stair = new StairCurve(m, flow.ClassMeasurementInterval);
    return Expressions.Convolution(stair, linkSpeedCurve,
        $"th_1stair{flow.Name}", $"linkspeed{flow.Path[0].Name}");
}
```

Nice-to-haves

4. *The problems scale with program complexity – we* compose *network studies*

We converted a tool using Nancy (from NPBA repo) to Nancy.Expressions.

It was straightforward, requiring little understanding of the underlying logic

```
hdev(((linkspeedFC-FS ∧ th_1brflow0) + (linkspeedFC-FS ∧ th_1brflow1) + (linkspeedFC-FS ∧ th_1brflow2) +
 (linkspeedFC-FS ∧ th_1brflow3) + (linkspeedFC-FS ∧ th_1brflow4) + (linkspeedFC-FS ∧ th_1brflow5) +
 (linkspeedFC-FS ∧ th_1brflow6) + (linkspeedFC-FS ∧ th_1brflow7) + (linkspeedFC-FS ∧ th_1brflow8) +
 (linkspeedFC-FS ∧ th_1brflow9)), minscAFC-FS) + hdev(((shaperAFC-FS ∧ ((((linkspeedFC-FS ∧ th_1brflow0) +
 (linkspeedFC-FS ∧ th_1brflow1) + (linkspeedFC-FS ∧ th_1brflow2) + (linkspeedFC-FS ∧ th_1brflow3) +
 (linkspeedFC-FS ∧ th_1brflow4) + (linkspeedFC-FS ∧ th_1brflow5) + (linkspeedFC-FS ∧ th_1brflow6) +
 (linkspeedFC-FS ∧ th_1brflow7) + (linkspeedFC-FS ∧ th_1brflow8) + (linkspeedFC-FS ∧ th_1brflow9)) ⊗ maxscAFC-FS)
⊘ minscAFC-FS))), minscAFS-BS) + hdev(((shaperAFS-BS ∧ ((((shaperAFC-FS ∧ ((((linkspeedFC-FS ∧ th_1brflow0) +
 (linkspeedFC-FS ∧ th_1brflow1) + (linkspeedFC-FS ∧ th_1brflow2) + (linkspeedFC-FS ∧ th_1brflow3) +
 (linkspeedFC-FS ∧ th_1brflow4) + (linkspeedFC-FS ∧ th_1brflow5) + (linkspeedFC-FS ∧ th_1brflow6) +
 (linkspeedFC-FS ∧ th_1brflow7) + (linkspeedFC-FS ∧ th_1brflow8) + (linkspeedFC-FS ∧ th_1brflow9)) ⊗ maxscAFC-FS)
⊘ minscAFC-FS))) ⊗ maxscAFS-BS) ⊘ minscAFS-BS)) + (shaperAHU-BS ∧ ((((linkspeedHU-BS ∧ th_1brflow10) +
 (linkspeedHU-BS ∧ th_1brflow11) + (linkspeedHU-BS ∧ th_1brflow12) + (linkspeedHU-BS ∧ th_1brflow13) +
 (linkspeedHU-BS ∧ th_1brflow14) + (linkspeedHU-BS ∧ th_1brflow15) + (linkspeedHU-BS ∧ th_1brflow16) +
 (linkspeedHU-BS ∧ th_1brflow17) + (linkspeedHU-BS ∧ th_1brflow18) + (linkspeedHU-BS ∧ th_1brflow19)) ⊗
 maxscAHU-BS) ⊘ minscAHU-BS))), minscABS-RU)
```

Optimizing with Nancy.Expressions

A few use cases and the road ahead

Nice-to-haves

2. *Code is computation strategy. Trade-off between readability and optimizations*

In Nancy.Expressions, most of the program builds the expressions to be computed
We can implement many optimizations as transformations or alternative Compute() algorithms
Only the last part of the program needs to be altered to take advantage of these

Equivalences

Transform expressions for speed of computation

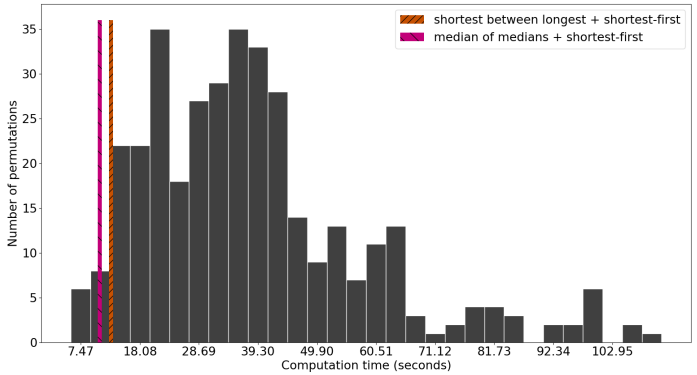
$$\beta^{eq'} = \bigotimes_{i=1}^{n-1} \left(\beta_i \bigotimes_{j=i}^{n-1} (\overline{\beta_j \otimes \beta_{j+1} + W_{j+1}}) \right) \otimes \beta_n$$

$$\overline{f} \otimes \overline{f} = \overline{f}$$

$$\beta^{eq'} = \left(\bigotimes_{i=1\ldots n} \beta_i \right) \otimes \bigotimes_{i=1\ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}}.$$

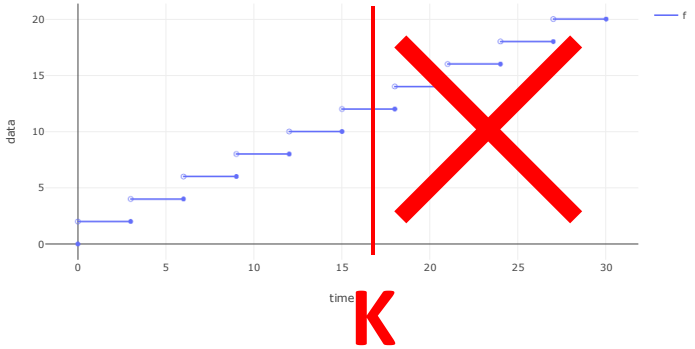
Computation Scheduling

Take the computation path that takes the least



Alternative Algorithms

Different *Compute()* algorithms to get the same result



Optimizing with Equivalences

Many results in literature simplify expressions into easier-to-compute ones

Instead of doing it with pen-and-paper, Nancy.Expressions allows to automate it

Example from literature [BJLL06, ZS23], involving $\mathrm{O}(n^2)$ convolutions

$$\beta^{eq'} = \bigotimes_{i=1}^{n-1} \left(\beta_i \bigotimes_{j=i}^{n-1} (\overline{\beta_j \otimes \beta_{j+1} + W_{j+1}}) \right) \otimes \beta_n$$

The convolution $\otimes$ is commutative and associative, and $\overline{f} \otimes \overline{f} = \overline{f}$

So in [BJLL06, ZS23], this was manually optimized into $O(n)$ convolutions

$$\beta^{eq'} = \left(\bigotimes_{i=1\ldots n} \beta_i \right) \otimes \bigotimes_{i=1\ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}}.$$

Optimizing with Equivalences

`Equivalence` objects represent these properties – including their hypotheses

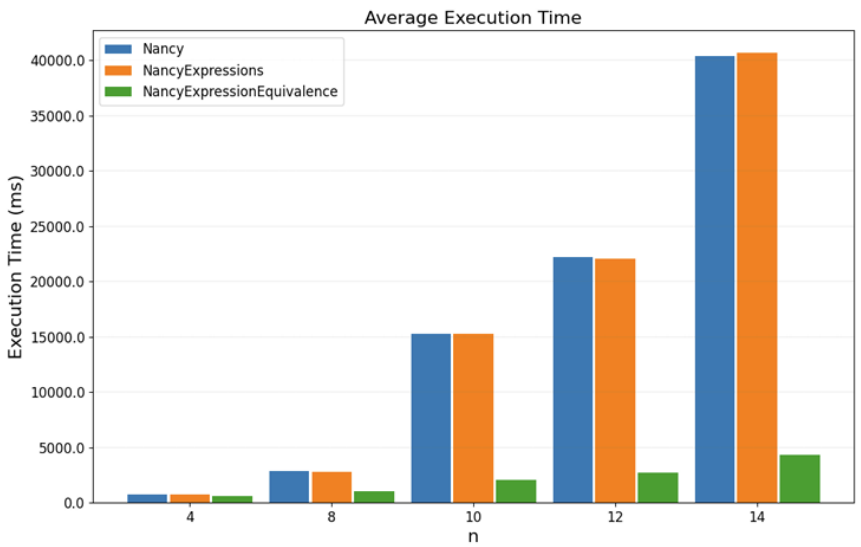
We can use them iteratively to simplify expressions

```
do
{
    previous = expression;
    expression = previous.ApplyEquivalence(equivalence);
} while (expression != previous);
```

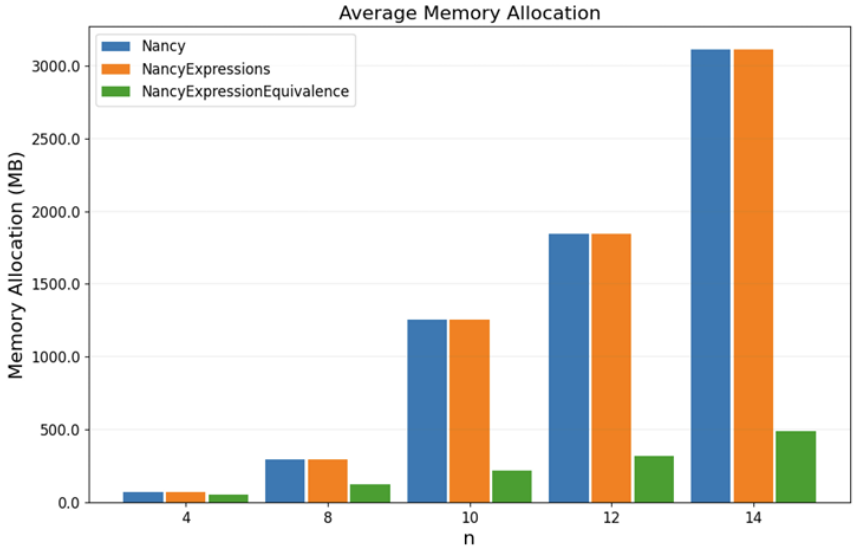
Scales well in instances where optimizations are hard to spot or apply

Optimizing with Equivalences

We benchmarked a simple program implementing the intuitive $O(n^2)$ expression



(a) Average execution time.



(b) Average memory allocation.

Fig. 4: Average execution time and memory allocation of the end-to-end service curve of a flow-controlled network with n nodes.

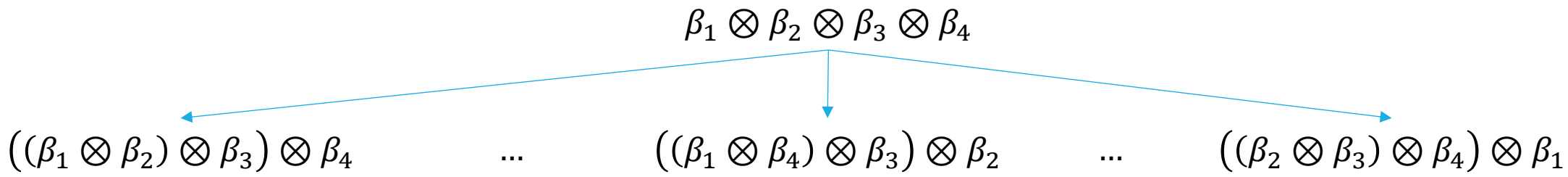
Optimizing with computation scheduling

This result was explored by **Antonio A. Salvalaggio** in his B.Sc. thesis

Presented as *Improving (min,+) Convolutions by Heuristic Operation Reordering* at JWRTC @RTNS24, won JRWRTC Best Paper Award

1. Associative and commutative operations yield the same result regardless of order of computation
2. But the *computation time* may not be the same

This work focused on (min,+) convolution and considered only its commutativity

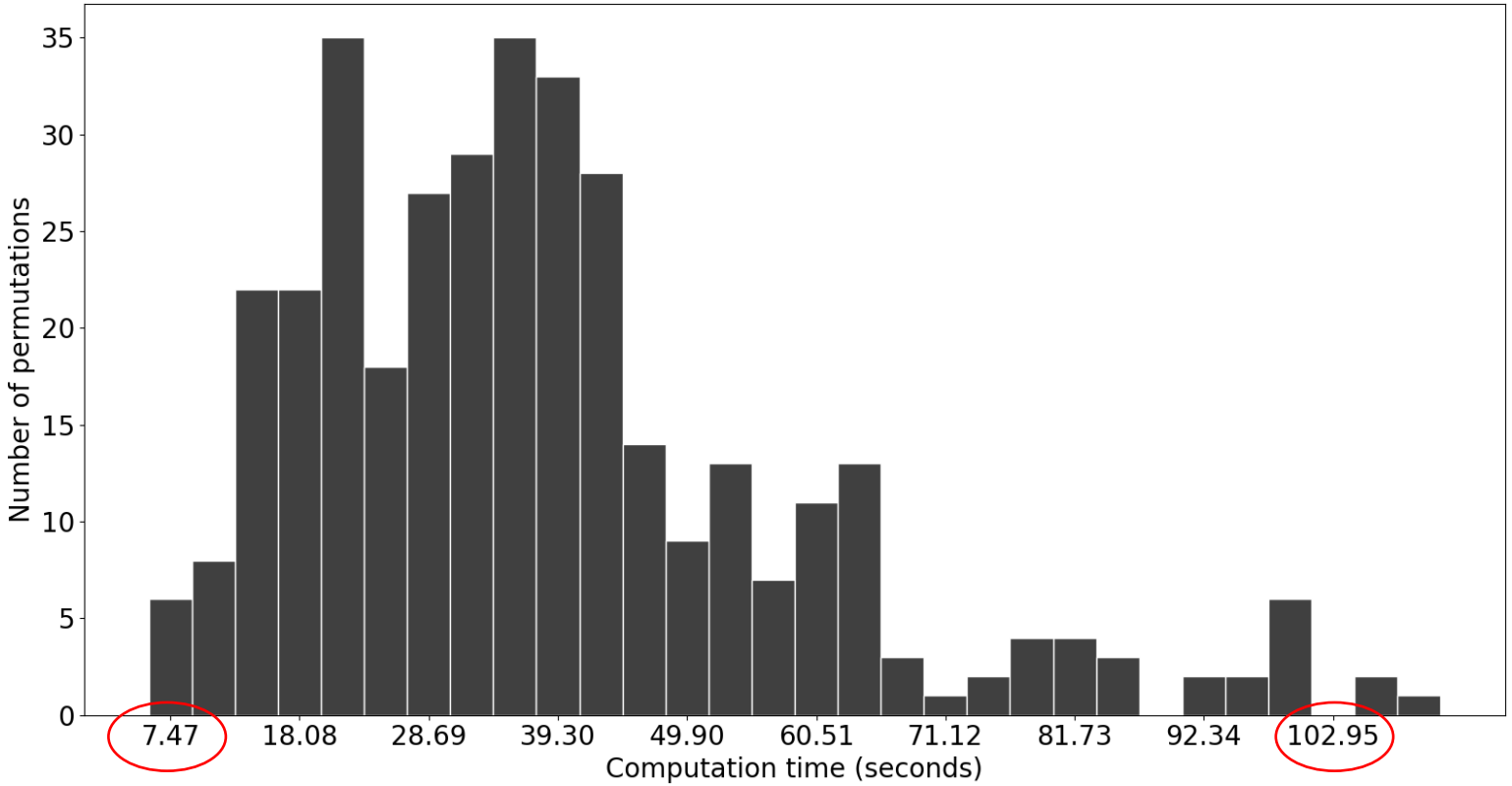


Optimizing with computation scheduling

We studied convolutions of 6 operands, comparing the computation time of all permutations

If left to chance, we may waste a lot of computation time

Goal: seek a “good” order

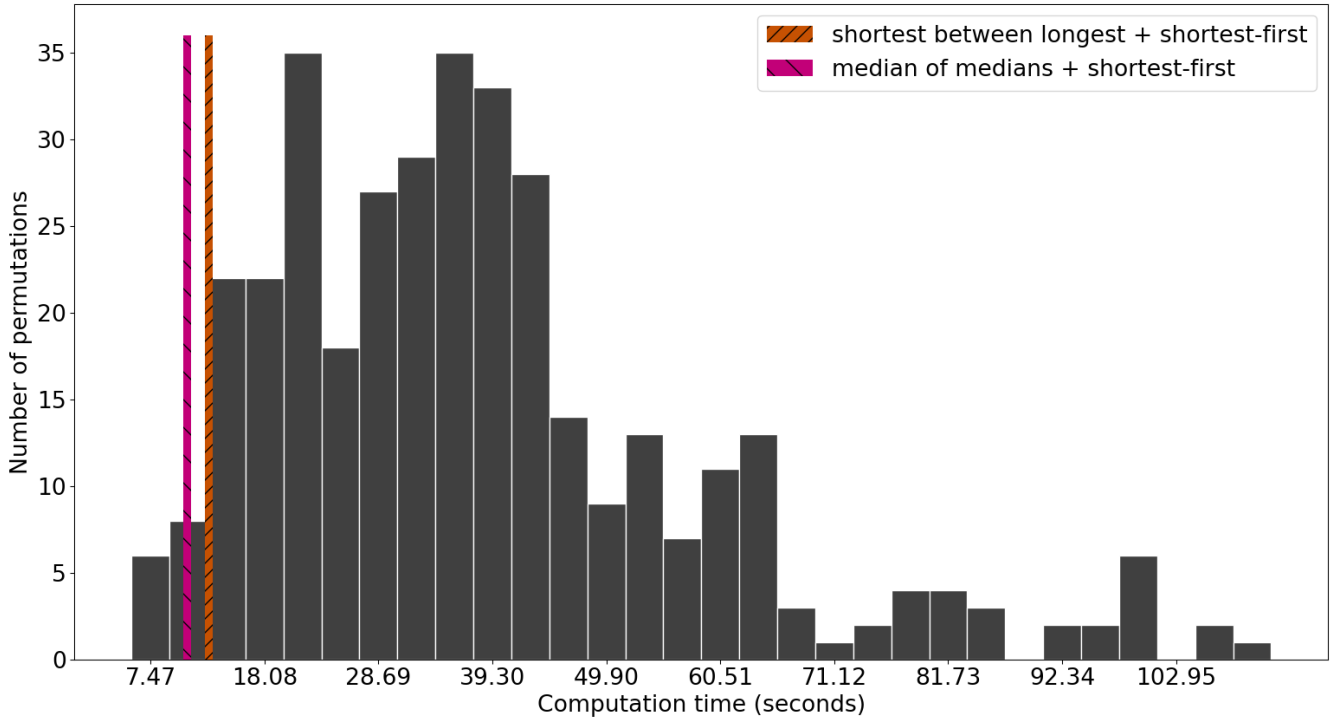


Optimizing with computation scheduling

We experimented with combining heuristics for *first-pair* and *next-curve* selection

The resulting algorithm can pick an order of computations that is “to the left” of the distribution

Better than random ordering, with little overhead



Current work is aiming to generalize this approach to consider associativity as well
The key are the *heuristics* used to predict cost and explore the options

Optimizing with *Compute()* algorithms

Before the call to `Compute()`, the program builds the expression tree object with all the necessary data

We can use this to use different strategies – like **Finitary RTC**

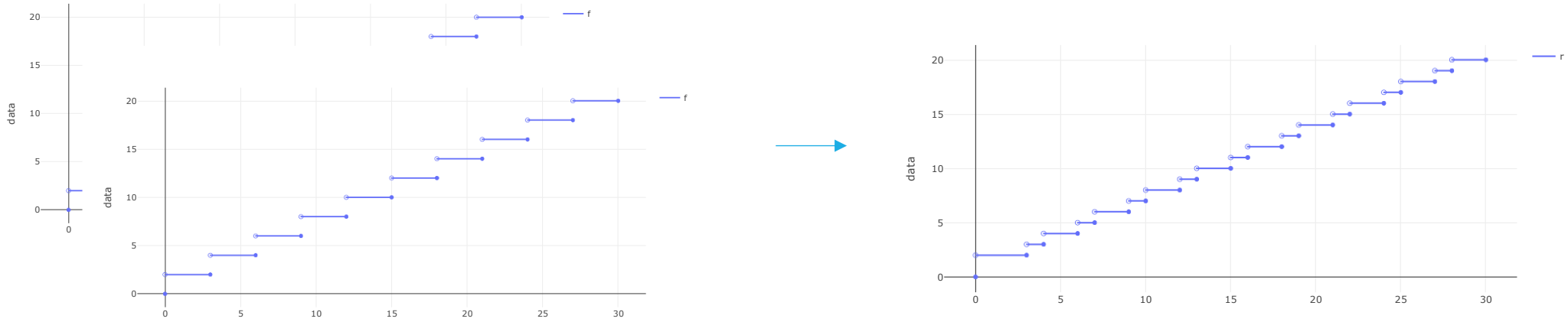
Finitary RTC in Nancy.Expressions

Finitary RTC was discussed in a series of papers [GY13, LBS16, LBSGY17]

Can be applied to $hdev()$ and $vdev()$ expressions, under some hypotheses

Generalized as an expression-based algorithm by **Iacopo Canetta** in his M.Sc. thesis

Main problem: period explosion in large studies



Period explosion

Finitary RTC in Nancy.Expressions

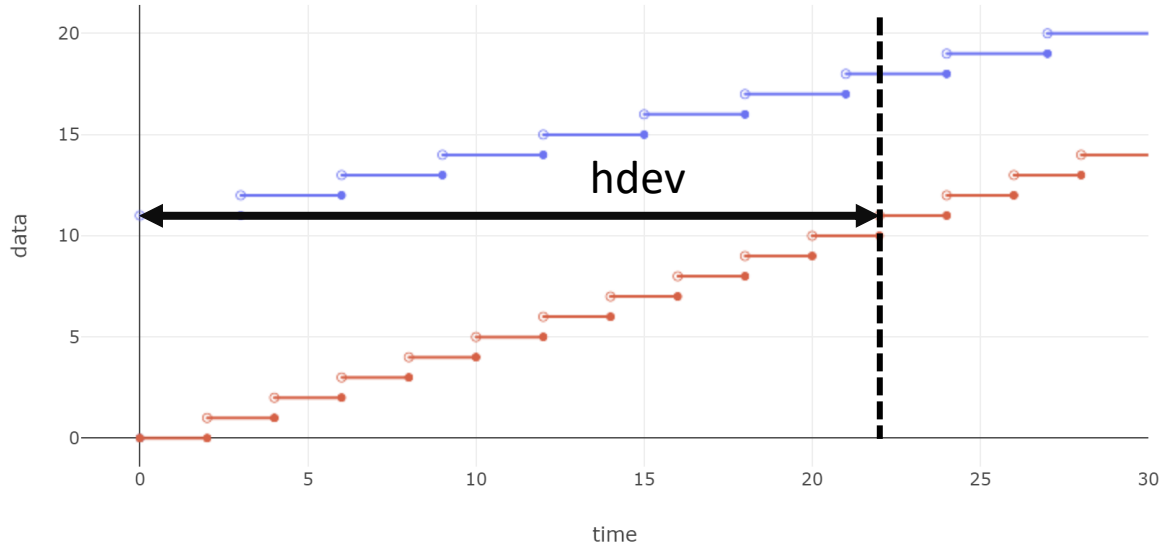
Finitary RTC was discussed in a series of papers [GY13, LBS16, LBSGY17]

Can be applied to $hdev()$ and $vdev()$ expressions, under some hypotheses

Generalized as an expression-based algorithm by **Iacopo Canetta** in his M.Sc. thesis

Main problem: period explosion in large studies

Core idea: bound the domain of interest, removing the unused periodic parts



Finitary RTC in Nancy.Expressions

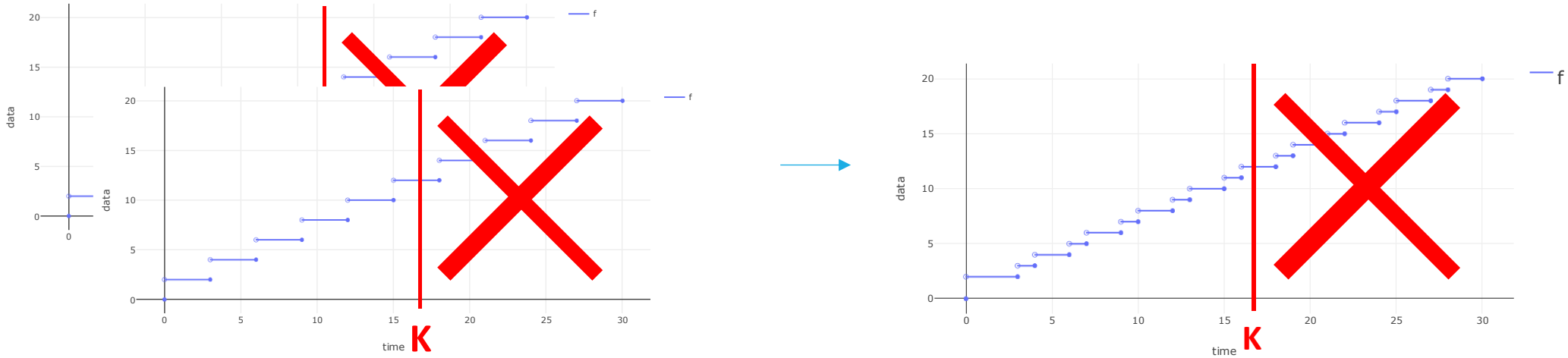
Finitary RTC was discussed in a series of papers [GY13, LBS16, LBSGY17]

Can be applied to $hdev()$ and $vdev()$ expressions, under some hypotheses

Generalized as an expression-based algorithm by **Iacopo Canetta** in his M.Sc. thesis

Main problem: period explosion in large studies

Core idea: bound the domain of interest, removing the unused periodic parts

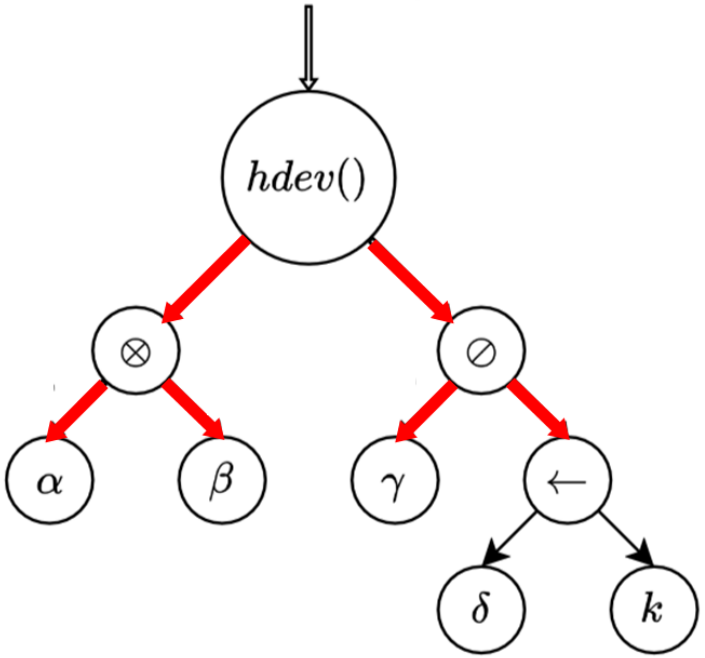


~~Period explosion~~

Finitary RTC in Nancy.Expressions

A 4-step algorithm:

1. Replace each operand with UA approximations

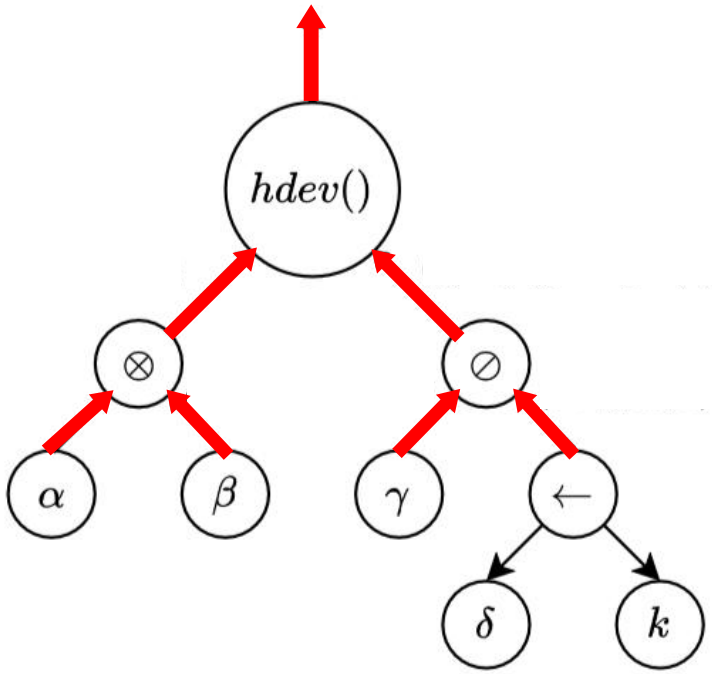


Top-down phase

Finitary RTC in Nancy.Expressions

A 4-step algorithm:

1. Replace each operand with UA approximations
2. Compute bound on hdev – say K_{hdev}

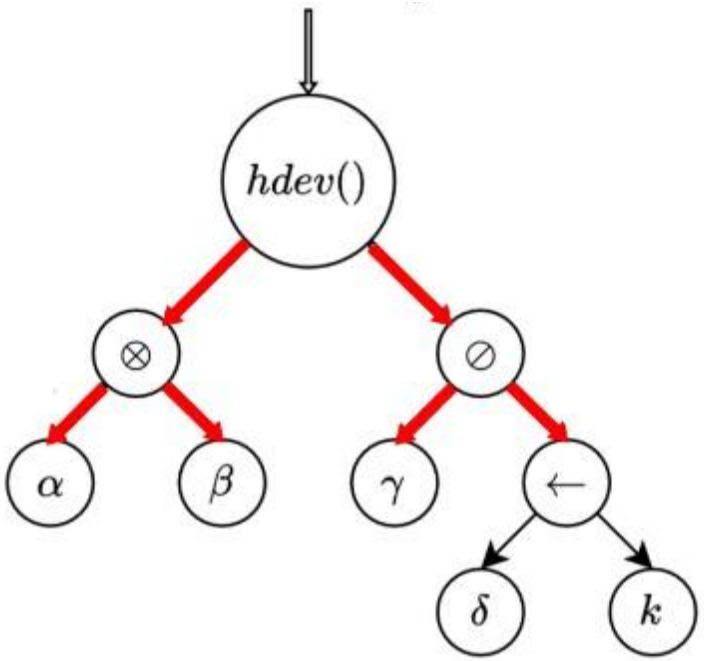


Bottom-up phase

Finitary RTC in Nancy.Expressions

A 4-step algorithm:

1. Replace each operand with UA approximations
2. Compute bound on hdev – say K_{hdev}
3. Propagate to find per-operand bounds – say $K_{\beta_1}, K_{\alpha_1}, \ldots$
   - Replace each operand with its finite cut, e.g. $\alpha_1' = \alpha_1([0, K_{\alpha_1}])$

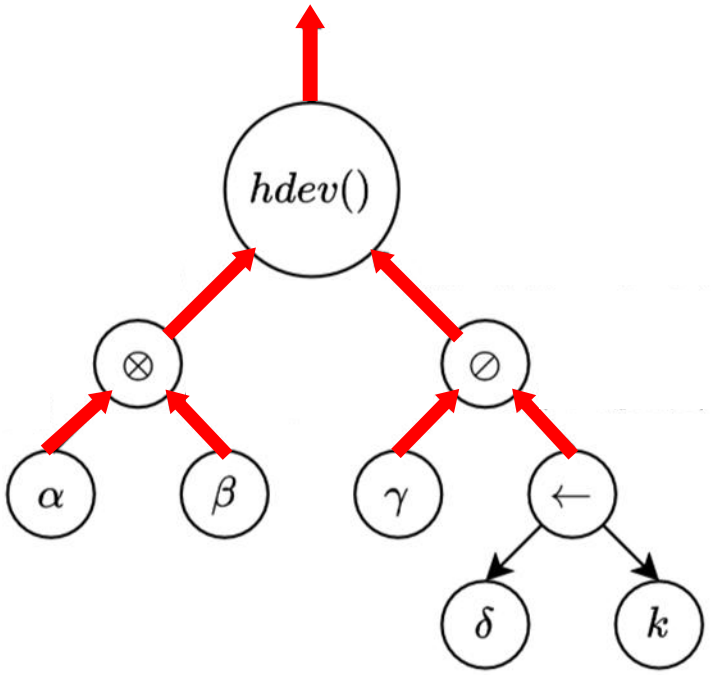


Top-down phase

Finitary RTC in Nancy.Expressions

A 4-step algorithm:

1. Replace each operand with UA approximations
2. Compute bound on hdev – say K_{hdev}
3. Propagate to find per-operand bounds – say $K_{\beta_1}, K_{\alpha_1}, \ldots$
   - Replace each operand with its finite cut, e.g. $\alpha_1' = \alpha_1([0, K_{\alpha_1}])$
4. Run the computation on finite cuts
   - Same result, no period explosion



Bottom-up phase

Finitary RTC in Nancy.Expressions

Exploiting this is problematic: you need to rewrite a great deal of code

```
var (c1AlphaU :Curve , c1AlphaL :Curve ,c1BetaU :Curve , c1BetaL :Curve ) =
    Gpc.ComputerGpc(alpha1CutU, alpha1CutL, beta1CutU, beta1CutL);
Console.WriteLine("Computed C1");

var c2AlphaU :Curve  = Gpc.ComputeAlphaU(c1AlphaU, beta2CutU, beta2CutL);
var c2BetaU :Curve  = Gpc.ComputeBetaU(c1AlphaL, beta2CutU);
var c2BetaL :Curve  = Gpc.ComputeBetaL(c1AlphaU, beta2CutL);
Console.WriteLine("Computed C2");

var c3BetaL :Curve  = Gpc.ComputeBetaL(c2AlphaU, beta3CutL);
Console.WriteLine("Computed C3");

var (c4AlphaU :Curve , c4AlphaL :Curve ,c4BetaU :Curve , c4BetaL :Curve ) =
    Gpc.ComputerGpc(alpha2CutU, alpha2CutL, c1BetaU, c1BetaL);
Console.WriteLine("Computed C4");

var c5AlphaU :Curve  = Gpc.ComputeAlphaU(c4AlphaU, c2BetaU, c2BetaL);
var c5BetaU :Curve  = Gpc.ComputeBetaU(c4AlphaL, c2BetaU);
var c5BetaL :Curve  = Gpc.ComputeBetaL(c4AlphaU, c2BetaL);
Console.WriteLine("Computed C5");

var c6BetaL :Curve  = Gpc.ComputeBetaL(c5AlphaU, c3BetaL);
Console.WriteLine("Computed C6");

var (c7AlphaU :Curve , c7AlphaL :Curve ,c7BetaU :Curve , c7BetaL :Curve ) =
    Gpc.ComputerGpc(alpha3CutU, alpha3CutL, c4BetaU, c4BetaL);
Console.WriteLine("Computed C7");

var c8AlphaU :Curve  = Gpc.ComputeAlphaU(c7AlphaU, c5BetaU, c5BetaL);
var c8BetaU :Curve  = Gpc.ComputeBetaU(c7AlphaL, c5BetaU);
var c8BetaL :Curve  = Gpc.ComputeBetaL(c7AlphaU, c5BetaL);
Console.WriteLine("Computed C8");

var c9BetaL :Curve  = Gpc.ComputeBetaL(c8AlphaU, c6BetaL);
Console.WriteLine("Computed C9");

var c10AlphaU :Curve  = Gpc.ComputeAlphaU(alpha4CutU, c7BetaU, c7BetaL);
Console.WriteLine("Computed C10");

var c11AlphaU :Curve  = Gpc.ComputeAlphaU(c10AlphaU, c8BetaU, c8BetaL);
Console.WriteLine("Computed C11");

var delay10 :Rational  = Curve.HorizontalDeviation(alpha4CutU, c7BetaL);
var delay11 :Rational  = Curve.HorizontalDeviation(c10AlphaU, c8BetaL);
var delay12 :Rational  = Curve.HorizontalDeviation(c11AlphaU, c9BetaL);
```

Nancy, without F-RTC, **35** lines of code

```
var (c1AlphaLinU, c1AlphaLinL, c10BetaLinU, c10BetaLinL) =
    Gpc.ComputerGpc(alpha1LinU, alpha1LinL, beta1LinU, beta1LinL);
Console.WriteLine("Computed L C1");

var (c2AlphaLinU, c2AlphaLinL, c2BetaLinU, c2BetaLinL) =
    Gpc.ComputerGpc(c1AlphaLinU, c1AlphaLinL, beta2LinU, beta2LinL);
Console.WriteLine("Computed L C2");

var c3BetaLinU = Gpc.ComputeBetaU(c2AlphaLinL, beta3LinU);
var c3BetaLinL = Gpc.ComputeBetaL(c2AlphaLinU, beta3LinL);
Console.WriteLine("Computed L C3");

var (c4AlphaLinU, c4AlphaLinL,c4BetaLinU, c4BetaLinL) =
    Gpc.ComputerGpc(alpha2LinU, alpha2LinL, c1BetaLinU, c1BetaLinL);
Console.WriteLine("Computed L C4");

var (c5AlphaLinU, c5AlphaLinL, c5BetaLinU, c5BetaLinL) =
    Gpc.ComputerGpc(c4AlphaLinU, c4AlphaLinL, c2BetaLinU, c2BetaLinL);
Console.WriteLine("Computed L C5");

var c6BetaLinU = Gpc.ComputeBetaU(c5AlphaLinL, c3BetaLinU);
var c6BetaLinL = Gpc.ComputeBetaL(c5AlphaLinU, c3BetaLinL);
Console.WriteLine("Computed L C6");

var (c7AlphaLinU, c7AlphaLinL,c7BetaLinU, c7BetaLinL) =
    Gpc.ComputerGpc(alpha3LinU, alpha3LinL, c4BetaLinU, c4BetaLinL);
Console.WriteLine("Computed L C7");

var (c8AlphaLinU, c8AlphaLinL,c8BetaLinU, c8BetaLinL) =
    Gpc.ComputerGpc(c7AlphaLinU, c7AlphaLinL, c5BetaLinU, c5BetaLinL);
Console.WriteLine("Computed L C8");

var c9BetaLinU = Gpc.ComputeBetaU(c8AlphaLinL, c6BetaLinU);
var c9BetaLinL = Gpc.ComputeBetaL(c8AlphaLinU, c6BetaLinL);
Console.WriteLine("Computed L C9");

var c10AlphaLinU = Gpc.ComputeAlphaU(alpha4LinU, c7BetaLinU, c7BetaLinL);
var c10AlphaLinL = Gpc.ComputeAlphaL(alpha4LinL, c7BetaLinU, c7BetaLinL);

Console.WriteLine("Computed L C10");

var c11AlphaLinU = Gpc.ComputeAlphaU(c10AlphaLinU, c8BetaLinU, c8BetaLinL);
var c11AlphaLinL = Gpc.ComputeAlphaL(c10AlphaLinL, c8BetaLinU, c8BetaLinL);
Console.WriteLine("Computed L C11");

//Compute all k
var (kaC10, kbC10) = FinitaryCalculus.DomainForBounds(alpha4LinU, c7BetaLinL);
var (kaC11, kbC11) = FinitaryCalculus.DomainForBounds(c10AlphaLinU, c8BetaLinL);
var (kaC12, kbC12) = FinitaryCalculus.DomainForBounds(c11AlphaLinU, c9BetaLinL);
var (kC11AlphaInU, kC11AlphaInL, kC11BetaInU, kC11BetaInL) = Gpc.FinitaryBoundaries
    (c10AlphaLinU, c10AlphaLinL, c8BetaLinU, c8BetaLinL,
        kaC12, kaC12,Rational.Zero, Rational.Zero);

var (kC10AlphaInU, kC10AlphaInL, kC10BetaInU, kC10BetaInL) = Gpc.FinitaryBoundaries
    (alpha4LinU, alpha4LinL, c7BetaLinU, c7BetaLinL,
        Rational.Max(kaC11, kC11AlphaInU), Rational.Max(kaC11, kC11AlphaInL),
        Rational.Zero, Rational.Zero);

var (kC9AlphaInU, kC9AlphaInL, kC9BetaInU, kC9BetaInL) = Gpc.FinitaryBoundaries
    (c8AlphaLinU, c8AlphaLinL, c6BetaLinU, c6BetaLinL,
    Rational.Zero, Rational.Zero, kbC12, kbC12);

var (kC8AlphaInU, kC8AlphaInL, kC8BetaInU, kC8BetaInL) = Gpc.FinitaryBoundaries
    (c7AlphaLinU, c7AlphaLinL, c5BetaLinU, c5BetaLinL,
    kC9AlphaInU, kC9AlphaInL, Rational.Max(kbC11, kC11BetaInU), Rational.Max(kbC11, kC11BetaInL));

var (kC7AlphaInU, kC7AlphaInL, kC7BetaInU, kC7BetaInL) = Gpc.FinitaryBoundaries
    (alpha3LinU, alpha3LinL, c4BetaLinU, c4BetaLinL,
    kC8AlphaInU, kC8AlphaInL,Rational.Max(kbC10, kC10BetaInU), Rational.Max(kbC10, kC10BetaInL));

var (kC6AlphaInU, kC6AlphaInL, kC6BetaInU, kC6BetaInL) = Gpc.FinitaryBoundaries
    (c5AlphaLinU, c5AlphaLinL, c3BetaLinU, c3BetaLinL,
    Rational.Zero, Rational.Zero, kC9BetaInU, kC9BetaInL);

var (kC5AlphaInU, kC5AlphaInL, kC5BetaInU, kC5BetaInL) = Gpc.FinitaryBoundaries
    (c4AlphaLinU, c4AlphaLinL, c2BetaLinU, c2BetaLinL,
    kC6AlphaInU, kC6AlphaInL, kC8BetaInU, kC8BetaInL);

var (kC4AlphaInU, kC4AlphaInL, kC4BetaInU, kC4BetaInL) = Gpc.FinitaryBoundaries
    (alpha2LinU, alpha2LinL, c1BetaLinU, c1BetaLinL,
    kC5AlphaInU, kC5AlphaInL, kC7BetaInU, kC7BetaInL);

var (kC3AlphaInU, kC3AlphaInL, kC3BetaInU, kC3BetaInL) = Gpc.FinitaryBoundaries
    (c2AlphaLinU, c2AlphaLinL, beta3LinU, beta3LinL,
    Rational.Zero, Rational.Zero, kC6BetaInU, kC6BetaInL);

var (kC2AlphaInU, kC2AlphaInL, kC2BetaInU, kC2BetaInL) = Gpc.FinitaryBoundaries
    (c1AlphaLinU, c1AlphaLinL, beta2LinU, beta2LinL,
    kC3AlphaInU, kC3AlphaInL, kC5BetaInU, kC5BetaInL);

var (kC1AlphaInU, kC1AlphaInL, kC1BetaInU, kC1BetaInL) = Gpc.FinitaryBoundaries
    (alpha1LinU, alpha1LinL, beta1LinU, beta1LinL,
    kC2AlphaInU, kC2AlphaInL, kC4BetaInU, kC4BetaInL);

var alpha1CutU = alpha1.UpperCurve.FinitaryCurve(kC1AlphaInU);
var alpha2CutU = alpha2.UpperCurve.FinitaryCurve(kC4AlphaInU);
var alpha3CutU = alpha3.UpperCurve.FinitaryCurve(kC7AlphaInU);
var alpha1CutL = alpha1.LowerCurve.FinitaryCurve(kC1AlphaInL);
var alpha2CutL = alpha2.LowerCurve.FinitaryCurve(kC4AlphaInL);
var alpha3CutL = alpha3.LowerCurve.FinitaryCurve(kC7AlphaInL);

var k4U = Rational.Max(kaC10, kC10AlphaInU);
var alpha4CutU = alpha4.UpperCurve.FinitaryCurve(k4U);

var beta1CutU = beta1.UpperCurve.FinitaryCurve(kC1BetaInU);
var beta2CutU = beta2.UpperCurve.FinitaryCurve(kC2BetaInU);
var beta1CutL = beta1.LowerCurve.FinitaryCurve(kC1BetaInL);
var beta2CutL = beta2.LowerCurve.FinitaryCurve(kC2BetaInL);
var beta3CutL = beta3.LowerCurve.FinitaryCurve(kC3BetaInL);

var (c1AlphaU, c1AlphaL,c1BetaU, c1BetaL) =
    Gpc.ComputerGpc(alpha1CutU, alpha1CutL, beta1CutU, beta1CutL);
Console.WriteLine("Computed C1");

var c2AlphaU = Gpc.ComputeAlphaU(c1AlphaU, beta2CutU, beta2CutL);
var c2BetaU = Gpc.ComputeBetaU(c1AlphaL, beta2CutU);
var c2BetaL = Gpc.ComputeBetaL(c1AlphaU, beta2CutL);
Console.WriteLine("Computed C2");

var c3BetaL = Gpc.ComputeBetaL(c2AlphaU, beta3CutL);
Console.WriteLine("Computed C3");

var (c4AlphaU, c4AlphaL,c4BetaU, c4BetaL) =
    Gpc.ComputerGpc(alpha2CutU, alpha2CutL, c1BetaU, c1BetaL);
Console.WriteLine("Computed C4");

var c5AlphaU = Gpc.ComputeAlphaU(c4AlphaU, c2BetaU, c2BetaL);
var c5BetaU = Gpc.ComputeBetaU(c4AlphaL, c2BetaU);
var c5BetaL = Gpc.ComputeBetaL(c4AlphaU, c2BetaL);
Console.WriteLine("Computed C5");

var c6BetaL = Gpc.ComputeBetaL(c5AlphaU, c3BetaL);
Console.WriteLine("Computed C6");

var (c7AlphaU, c7AlphaL,c7BetaU, c7BetaL) =
    Gpc.ComputerGpc(alpha3CutU, alpha3CutL, c4BetaU, c4BetaL);
Console.WriteLine("Computed C7");

var c8AlphaU = Gpc.ComputeAlphaU(c7AlphaU, c5BetaU, c5BetaL);
var c8BetaU = Gpc.ComputeBetaU(c7AlphaL, c5BetaU);
var c8BetaL = Gpc.ComputeBetaL(c7AlphaU, c5BetaL);
Console.WriteLine("Computed C8");

var c9BetaL = Gpc.ComputeBetaL(c8AlphaU, c6BetaL);
Console.WriteLine("Computed C9");

var c10AlphaU = Gpc.ComputeAlphaU(alpha4CutU, c7BetaU, c7BetaL);
Console.WriteLine("Computed C10");

var c11AlphaU = Gpc.ComputeAlphaU(c10AlphaU, c8BetaU, c8BetaL);
Console.WriteLine("Computed C11");

var delay10 = Curve.HorizontalDeviation(alpha4CutU, c7BetaL);
var delay11 = Curve.HorizontalDeviation(c10AlphaU, c8BetaL);
var delay12 = Curve.HorizontalDeviation(c11AlphaU, c9BetaL);
var delay = delay10 + delay11 + delay12;
Console.WriteLine($"The delay is: {delay}");
```

Nancy, with F-RTC, **139** lines of code

Finitary RTC in Nancy.Expressions

With the expression-based algorithm and Nancy.Expressions, this is now free

```
var result = delayExpression.Compute();
```

```
var result = delayExpression.ComputeFinitary();
```

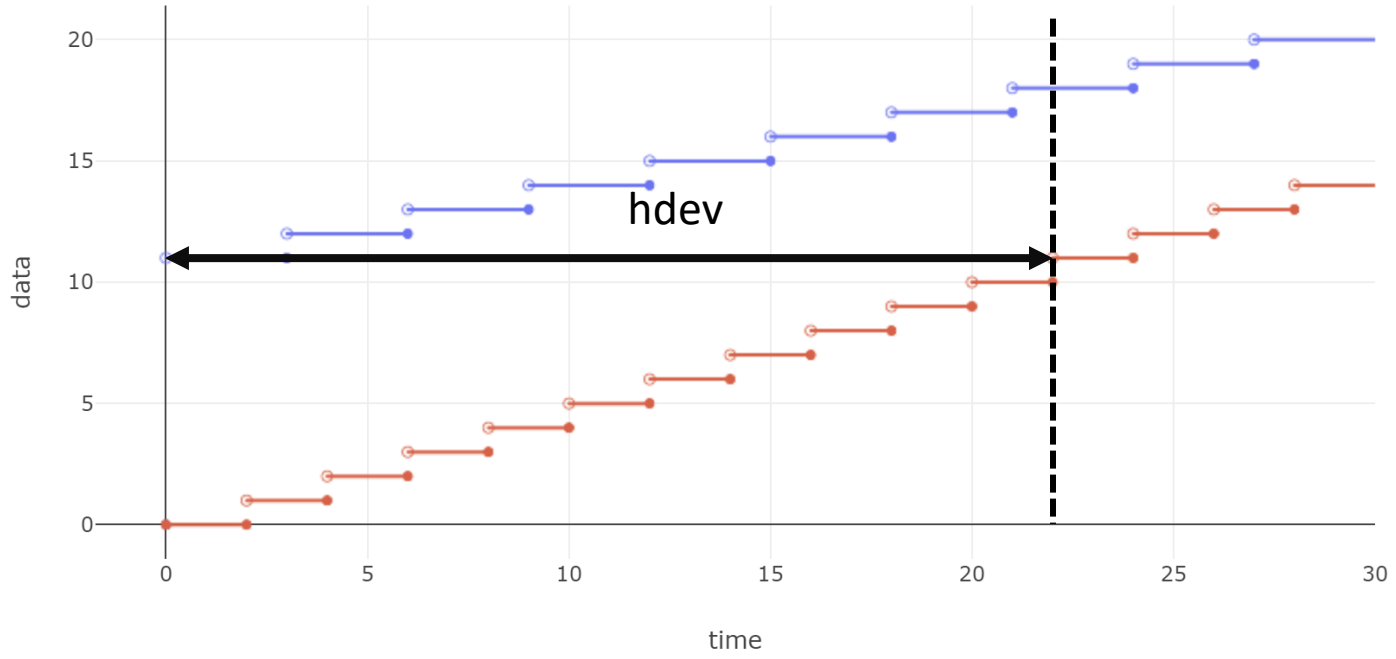
```
var (c1AlphaU :CurveExpression , c1AlphaL :CurveExpression , c1BetaU :CurveExpression , c1BetaL :CurveExpression ) =
    GpcExpressionComponent.GpcCurveExpression(alpha1U, alpha1L, beta1U, beta1L);
Console.WriteLine("Computed C1");

var c2AlphaU :CurveExpression  = GpcExpressionComponent.AlphaUExpression(c1AlphaU, beta2U, beta2L);
var c2BetaU :CurveExpression  = GpcExpressionComponent.BetaUExpression(c1AlphaL, beta2U);
var c2BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c1AlphaU, beta2L);
Console.WriteLine("Computed C2");

var c3BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c2AlphaU, beta3L);
Console.WriteLine("Computed C3");

var (c4AlphaU :CurveExpression , c4AlphaL :CurveExpression , c4BetaU :CurveExpression , c4BetaL :CurveExpression ) =
    GpcExpressionComponent.GpcCurveExpression(alpha2U, alpha2L, c1BetaU, c1BetaL);
Console.WriteLine("Computed C4");

var c5AlphaU :CurveExpression  = GpcExpressionComponent.AlphaUExpression(c4AlphaU, c2BetaU, c2BetaL);
var c5BetaU :CurveExpression  = GpcExpressionComponent.BetaUExpression(c4AlphaL, c2BetaU);
var c5BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c4AlphaU, c2BetaL);
Console.WriteLine("Computed C5");

var c6BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c5AlphaU, c3BetaL);
Console.WriteLine("Computed C6");

var (c7AlphaU :CurveExpression , c7AlphaL :CurveExpression , c7BetaU :CurveExpression , c7BetaL :CurveExpression ) =
    GpcExpressionComponent.GpcCurveExpression(alpha3U, alpha3L, c4BetaU, c4BetaL);
Console.WriteLine("Computed C7");

var c8AlphaU :CurveExpression  = GpcExpressionComponent.AlphaUExpression(c7AlphaU, c5BetaU, c5BetaL);
var c8BetaU :CurveExpression  = GpcExpressionComponent.BetaUExpression(c7AlphaL, c5BetaU);
var c8BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c7AlphaU, c5BetaL);
Console.WriteLine("Computed C8");

var c9BetaL :CurveExpression  = GpcExpressionComponent.BetaLExpression(c8AlphaU, c6BetaL);
Console.WriteLine("Computed C9");

var c10AlphaU :CurveExpression  = GpcExpressionComponent.AlphaUExpression(alpha4U, c7BetaU, c7BetaL);
Console.WriteLine("Computed C10");

var c11AlphaU :CurveExpression  = GpcExpressionComponent.AlphaUExpression(c10AlphaU, c8BetaU, c8BetaL);
Console.WriteLine("Computed C11");

var delay10 = new HorizontalDeviationExpression(alpha4U,  RightExpression: c7BetaL);
var delay11 =  new HorizontalDeviationExpression(c10AlphaU,  RightExpression: c8BetaL);
var delay12 =  new HorizontalDeviationExpression(c11AlphaU,  RightExpression: c9BetaL);
var delay :Rational  = delay10.ComputeFinitary() + delay11.ComputeFinitary() + delay12.ComputeFinitary();
```

Nancy.Expressions, **35** lines of code

Finitary RTC in Nancy.Expressions

Now that it is easy to use – is it always worth it?

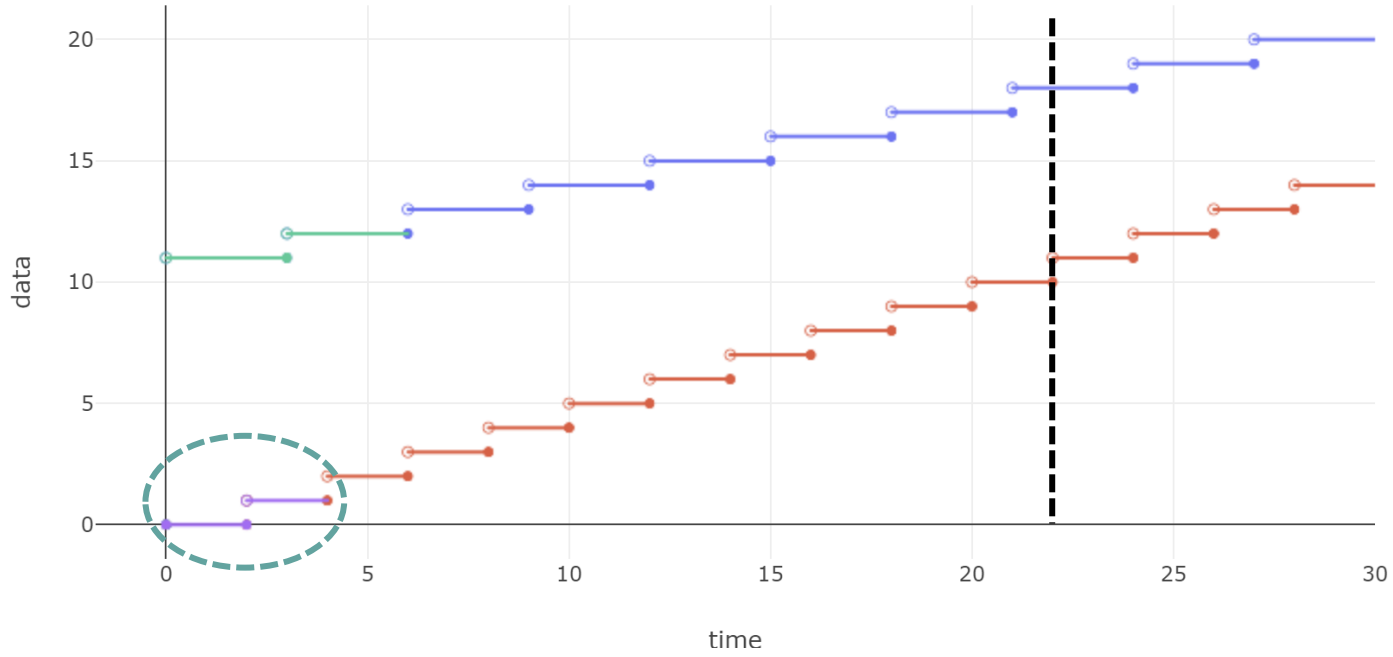
No.



Finitary RTC in Nancy.Expressions

Now that it is easy to use – is it always worth it?

No. The finite prefix from F-RTC may be larger than the *minimal representation*



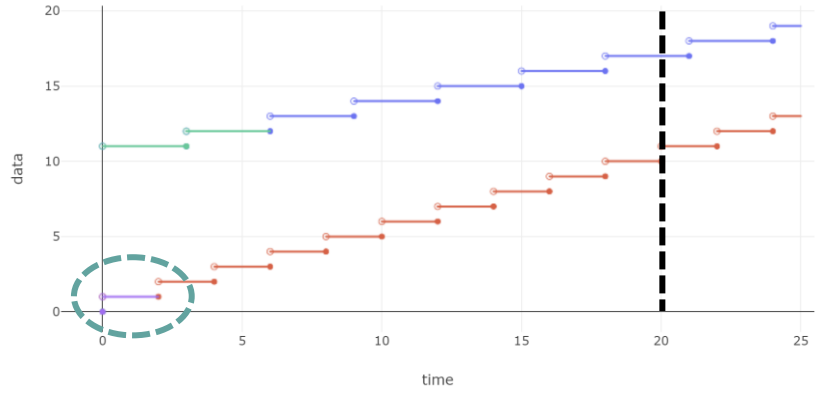
Finitary RTC in Nancy.Expressions

Now that it is easy to use – is it always worth it?

No. The finite prefix from F-RTC may be larger than the *minimal representation*

Increasing the burst or the rate utilizations are easy ways to find examples where plain UPP curves are much faster – using Nancy optimizations like super-isospeed

Next goal: find **heuristic** to transparently apply the best method



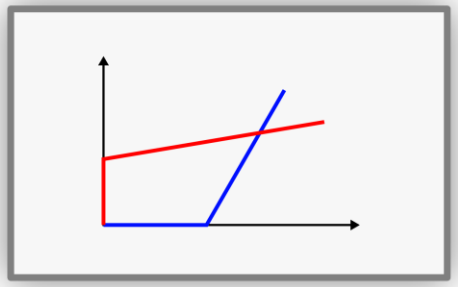
Nancy.Expressions

A library for computation of Deterministic Network Calculus *expressions*

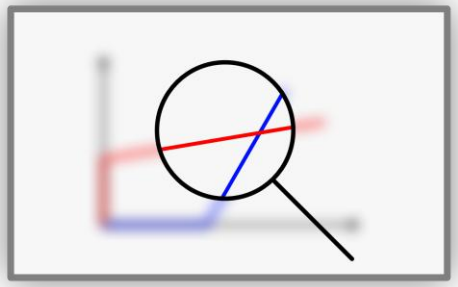
Open-source, written in C#, extends the existing *Nancy* library

While *Nancy* implements operators, *Nancy.Expressions* generalizes it to expressions

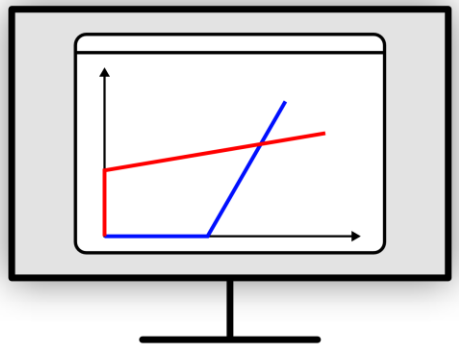
- Build, visualize and manipulate expressions before computing them
- Apply equivalences to simplify expressions
- Use different computation algorithms without rewriting the entire program



For Teaching



For Research



For Applications

Nancy.Expressions

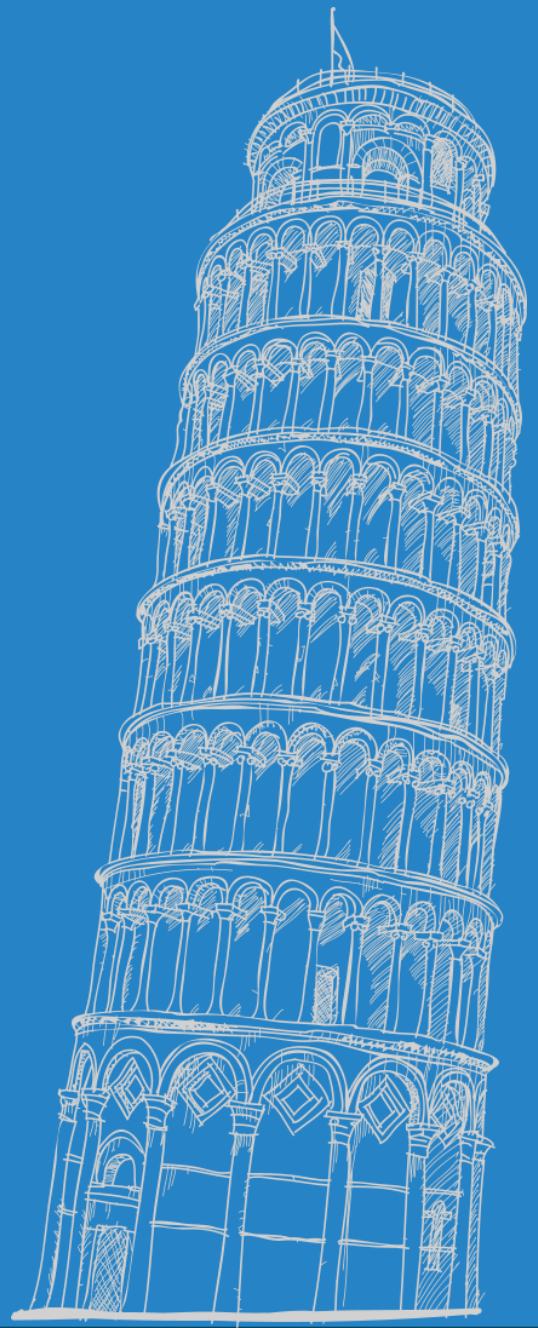
Developer experience
Writing and debugging analysis tool is improved thanks to the focus on expressions, that remove a lot of noise from the program structure

State of the art accessibility
Results from literature like theorems, properties and compute algorithms can be coded in and re-used much more easily

New paths for optimizations
Our experiments with the above showed many trade-off instances

There is much room to research heuristics to improve runtime and make harder or more accurate studies feasible

Thanks!



References

[BJLL06] A. Bose, X. Jiang, B. Liu, and G. Li, *Analysis of Manufacturing Blocking Systems with Network Calculus*, Performance Evaluation, 2006

[GY13] N. Guan and W. Yi, *Finitary Real-Time Calculus: Efficient Performance Analysis of Distributed Embedded Systems*, IEEE RTSS 2013

[LBS16] K. Lampka, S. Bondorf and J. B. Schmitt, *Achieving Efficiency without Sacrificing Model Accuracy: Network Calculus on Compact Domains*, IEEE MASCOTS 2016

[LBSGY17] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan and W. Yi, *Generalized Finitary Real-Time Calculus*, IEEE INFOCOM 2017

[ZS22] R. Zippo and G. Stea, *Nancy: an efficient parallel Network Calculus library*, SoftwareX, 2022

[ZS23] R. Zippo and G. Stea, *Computationally Efficient Worst-Case Analysis of Flow-Controlled Networks With Network Calculus*, IEEE Transactions on Information Theory, 2023

[SZS24] A. A. Salvalaggio, R. Zippo, and G. Stea, *Improving (min,+) Convolutions by Heuristic Operation Reordering*, JWRTC @RTNS24

[TZS24] A. Trasacco, R. Zippo, and G. Stea*, Nancy.Expressions: Towards a CAS for DNC*, VALUETOOLS 2024

Equivalences and *cut-through* properties

An important part of applying an equivalence is <u>testing its hypotheses</u>

Example: knowing that f is subadditive, we can simplify $f \otimes f = f$

To test subadditivity, we need to *verify* that $f \otimes f = f$

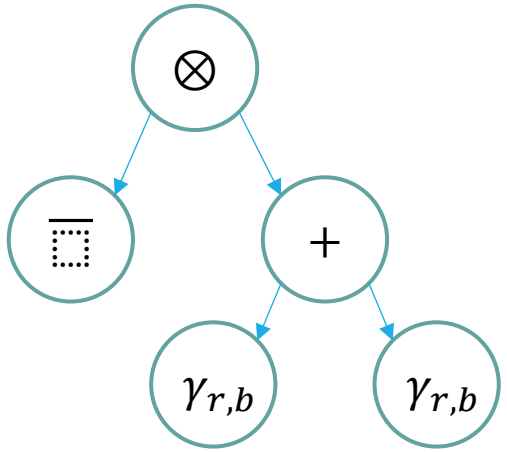
Recall that f may be an *expression*

$$\beta^{eq'} = \bigotimes_{i=1}^{n-1} \left(\beta_i \bigotimes_{j=i}^{n-1} (\overline{\beta_j \otimes \beta_{j+1} + W_{j+1}}) \right) \otimes \beta_n$$

$$\beta^{eq'} = \left(\bigotimes_{i=1 \ldots n} \beta_i \right) \otimes \bigotimes_{i=1 \ldots n-1} \overline{\beta_i \otimes \beta_{i+1} + W_{i+1}}.$$

Equivalences and *cut-through* properties

Core idea: while testing properties, many computations can be skipped if the results in literature are cleverly used



$$\left(\gamma_{r_1,b_1} + \gamma_{r_2,b_2}\right) \otimes \overline{\square}$$

The convolution is (surely) subadditive if its operands are

The subadditive closure is subadditive

The sum is (surely) subadditive if its operands are

Token-bucket arrival curves are subadditive

So we can conclude this expression is subadditive without computing it and its self-convolution

Nancy.Expressions is designed to take advantage of these inferences avoiding unnecessary computations

Optimizing with Equivalences - Syntax

In code

```
var equivalence = new Equivalence(
    leftSideExpression: Expressions.Convolution(
        Expressions.Placeholder("f"),
        Expressions.Placeholder("g")),
    rightSideExpression: Expressions.Placeholder("f")
);
equivalence.AddHypothesis("f", f => f.IsSubAdditive);
equivalence.AddHypothesis("g", g => g.IsZeroAtZero);
equivalence.AddHypothesis(
    "f", "g", (CurveExpression f, CurveExpression g) => f <= g);
equivalence.AddHypothesis(
    "f", "g", (f, g) => f.Convolution(g).IsWellDefined);
```

Using an ad-hoc syntax

```
{
    f in U subadditive;
    g in U zero-at-zero;
    f <= g;
    f * g well-defined;
} == > f * g = f
```

Optimizing with Equivalences - Limitations

In its current state, application is automated but **manually triggered**.

The user is expected to know which equivalences may help

A valuable goal of this is to *transparently* apply known equivalences to optimize

This is a non-trivial, with different challenges

- What is the cost of looking for matches and verifying the hypotheses?
- What is the time saved by using the equivalence?
- What about substitutions that enable *other* equivalences?
- How should we limit the exploration overhead vs. the computation time?

Optimizing with computation scheduling - insights

The difference comes from wasted computations

Nancy can minimize the result *after* a computation, but also uses optimizations that skip unnecessary ones *before*

A “good path” has good intermediate operations that avoid useless computations that are later discarded

Addressing associativity is more complicated, but has several advantages

- A larger search space should include lower minima
- Can exploit parallelization on multicore systems
- Can be mixed with other results, e.g. convolutions of convex/concave/subadditive functions

An abstraction hierarchy

Abstraction Level	Possible optimizations	Software
DNC and system model	How the system model is expressed into curves, e.g. a strict service curve can be replaced by its superadditive closure.	Your analysis tool, ??
(min,+) and (max,+) expressions	Use algebraic properties to compute the *expression* faster	Nancy.Expressions, NC Maude?
(min,+) and (max,+) operations	Use algebraic properties to compute the *operation* faster	Nancy, RTC toolbox