McMaster University

Type-safe modeling for optimization

Nhan Thai
thain1@mcmaster.ca

Supervisors:
Dr. Christopher Anand
Dr. Wolfram Kahl

A Dissertation Presented in Partial Fulfillment of the Requirements
for the Degree Master of Science in Computer Science

McMaster University
June 17, 2021

Outline

McMaster University

- Context, problem and introduction
- Proposal
- Implementation
- Conclusion and future work

Outline

McMaster University

- **Context, problem and introduction**
- Proposal
- Implementation
- Conclusion and future work

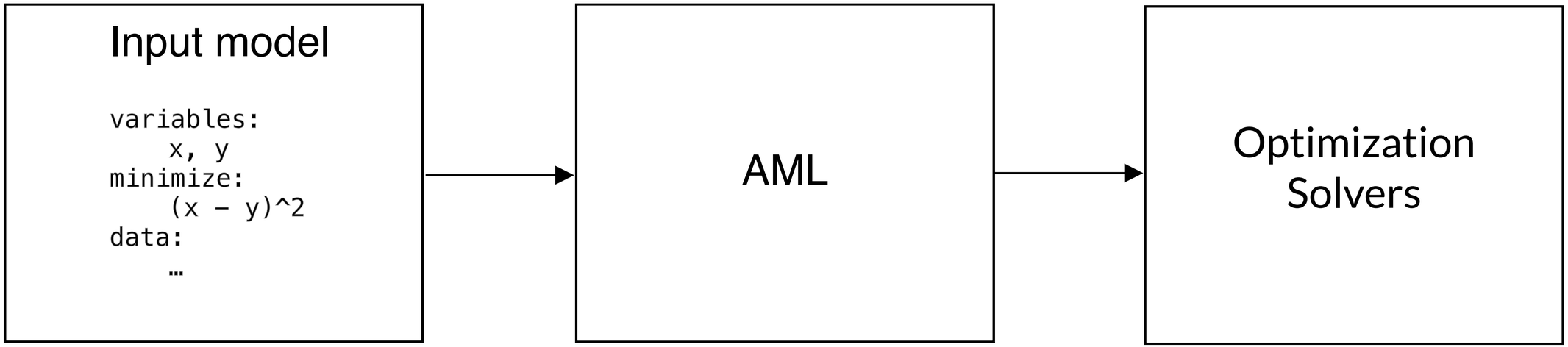
Mathematical Optimization

McMaster University

$$
\begin{aligned}
&\text{minimize:} && f(x) \\
&\text{subject to:} && g_i(x) \leq 0, \quad i = 1, 2, ..., m_g \\
& && h_i(x) = 0, \quad i = 1, 2, ..., m_h \\
& && (f, g_i, h_i : \mathbb{R}^n \longrightarrow \mathbb{R})
\end{aligned}
$$

AML

- Algebraic Modeling Languages (AMLs)



AML

- Algebraic Modeling Languages (AMLs)
  - AMPL, GAMS
  - Pyomo, PuLP, CVXPY
  - YALMIP, CVX
  - JuMP
  - Tensorflow, Pytorch
- Solvers
  - Gradient descent, L-BFGS, L-BFGS-B, etc.

Problem

McMaster University

- Embeddability
  - GAMS, AMPL

Problem

- Type-safety
  - Python, MATLAB
  - Julia

```python
# internal library representation, hidden to users
class Plus:
  def __init__(self, arg1, arg2):
    self.arg1 = arg1
    self.arg2 = arg2
  def eval(self):
    return self.arg1 + self.arg2

# modeling
x         = 1
y         = [2, 3]
objective = Plus(x, y)

# the library calls eval() later, also hidden to users
objective.eval()
```

```
Traceback (most recent call last):
  File "…", line 15, in <module>
    objective.eval()
  File "…", line 7, in eval
    return self.arg1 + self.arg2
TypeError: unsupported operand
type(s) for +: 'int' and 'list'
```

Problem

McMaster University

- Model & data intermingling
- Mutable
- Hard to analyze & optimize models:
  - Identify common computations
  - Using algebraic properties to optimize computations

```
# Create the concrete model
model = ConcreteModel()

# Rate constants
model.k1 = Var(initialize = 5.0/6.0, within=PositiveReals) # min^-1
model.k2 = Var(initialize = 5.0/3.0, within=PositiveReals) # min^-1
model.k3 = Var(initialize = 1.0/6000.0, within=PositiveReals) # m^3/(gmol mi
model.k1.fixed = True
model.k2.fixed = True
model.k3.fixed = True

# Inlet concentration of A, gmol/m^3
model.caf = Var(initialize = float(data['caf']), within=PositiveReals)
model.caf.fixed = True

# Space velocity (flowrate/volume)
model.sv = Var(initialize = float(data['sv']), within=PositiveReals)
model.sv.fixed = True

# Outlet concentration of each component
model.ca = Var(initialize = 5000.0, within=PositiveReals)
model.cb = Var(initialize = 2000.0, within=PositiveReals)
model.cc = Var(initialize = 2000.0, within=PositiveReals)
model.cd = Var(initialize = 1000.0, within=PositiveReals)

# Objective
model.obj = Objective(expr = model.cb, sense=maximize)
```

Problem

McMaster University

- Performance
  - High-level programming languages
  - Slow-code vs fast-code

Problem

McMaster University

- Other
  - Support for complex numbers
  - Support for higher dimensional variables

Outline

McMaster University

- Context, problem and introduction
- **Proposal**
- Implementation
- Conclusion and future work

Proposal

McMaster University

- HashedExpression, an algebraic modeling language:
  - Open-source, embedded in Haskell
  - Type-safety
    - Invalid models result in compile errors

Proposal

McMaster University

- HashedExpression, an algebraic modeling language:
  - Decouple model from data
    - Symbolic representation
  - High performance
    - Low-level C code generation
  - Complex numbers, multidimensional variables

Contributions

McMaster University

- Continuation of Jessica Pavlin's thesis work and others
- My contributions:
  - The new type systems
    - Type-level programming
    - Units
  - A rewrite of the core system
  - Reverse-mode differentiation
  - Solver-agnostic code generator
    - Combined with any optimizer

Example

- Magnetic Resonance Imaging (MRI) Reconstruction

$$\hat{x} = \arg\min_{x} \ \|\pi_{\text{mask}}(\text{FT}(x) - m)\|^2 + \lambda \sum_{i,j} \left((x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2 \right)$$

$$\text{subject to:}$$

$$x_{lb} \preceq x \preceq x_{ub}$$

Example

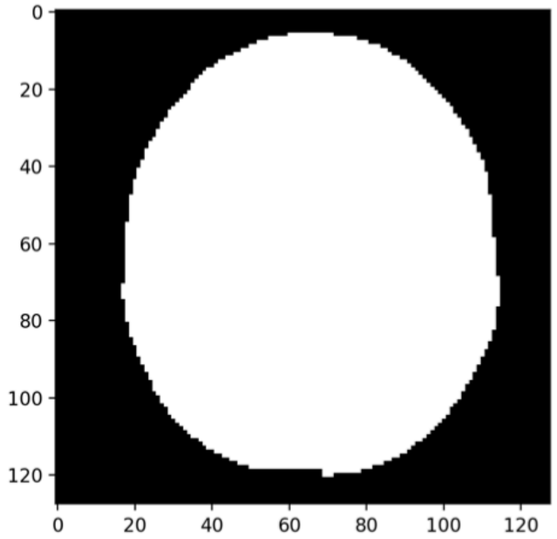
```
brainReconstructFromMRI :: OptimizationProblem
brainReconstructFromMRI =
  let -- variables
      x = variable2D @128 @128 "x"
      --- bound
      xLowerBound = bound2D @128 @128 "x_lb"
      xUpperBound = bound2D @128 @128 "x_ub"
      -- parameters
      a = param2D @128 @128 "a"
      b = param2D @128 @128 "b"
      mask = param2D @128 @128 "mask"
      -- regularization
      regularization = norm2square (rotate (0, 1) x - x) + norm2square (rotate (1, 0) x - x)
      lambda = 3000
   in OptimizationProblem
        { objective =
            norm2square ((mask +: 0) * (ft (x +: 0) - (a +: b)))
              + lambda * regularization,
          constraints =
            [ x .<= xUpperBound,
              x .>= xLowerBound
            ],
          values =
            [ a :-> VFile (HDF5 "kspace.h5" "a"),
              b :-> VFile (HDF5 "kspace.h5" "b"),
              mask :-> VFile (HDF5 "mask.h5" "mask"),
              xLowerBound :-> VFile (HDF5 "bound.h5" "lb"),
              xUpperBound :-> VFile (HDF5 "bound.h5" "ub")
            ]
        }
```

$$\hat{x} = \arg\min_x \ \|\pi_{\text{mask}}(\text{FT}(x) - m)\|^2 + \lambda \sum_{i,j} \left((x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2\right)$$

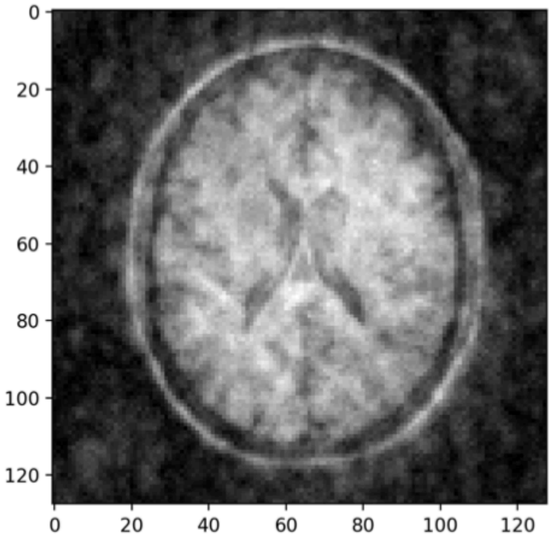
subject to:

$$x_{lb} \preceq x \preceq x_{ub}$$

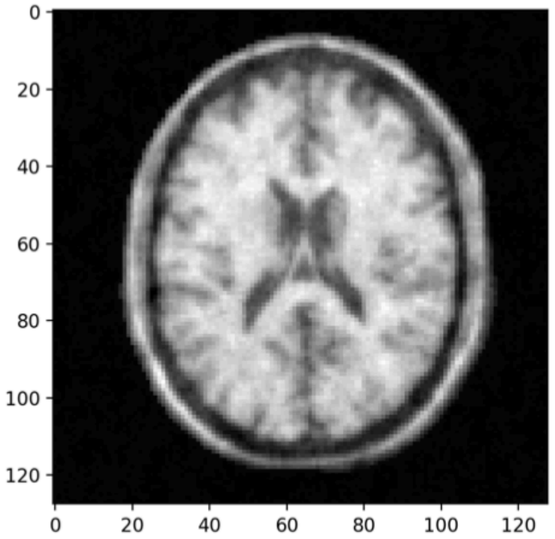
Example



(a) Boundary of the head.



(b) Naively reconstructed by taking the inverse Fourier transform.



(c) Using optimization problem with regularization.

Outline

McMaster University

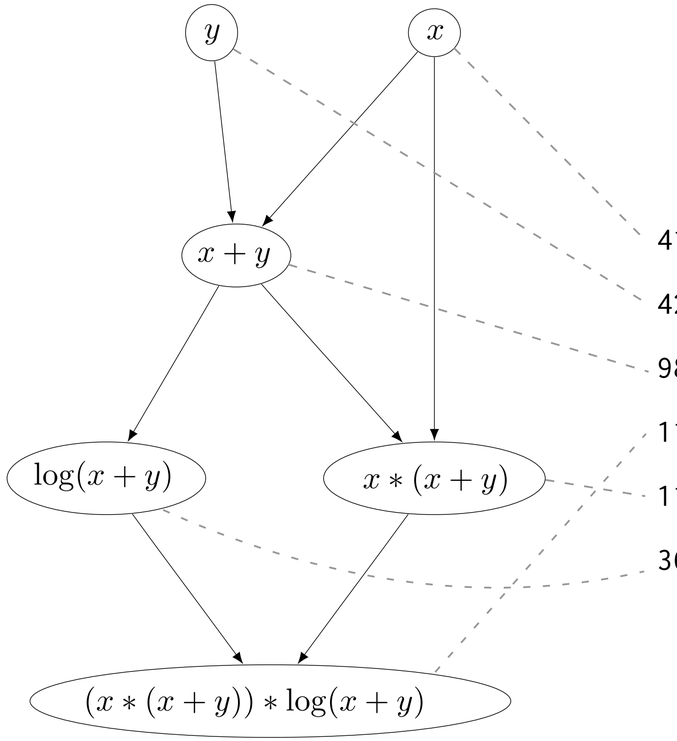
- Context, problem and introduction
- Proposal
- **Implementation**
- Conclusion and future work

Expression

McMaster University

- Directed Acyclic Graph
- Map
  - Each entry is a node in the graph
- Node indexed by its hash
  - Common subexpression elimination
  - Efficient matching

Expression




```
41509096:          ([], R, Var “x”)
42080883:          ([], R, Var “y”)
989450128496424:   ([], R, Sum [41509096, 42080883])
1117216945168872:  ([], R, Mul [1138089690287577, 3633358430263915])
1138089690287577:  ([], R, Mul [41509096, 989450128496424])
3633358430263915:  ([], R, Log 98945012849642)
```

Expression

McMaster University

```haskell
-- []     => scalar
-- [n]    => 1 dimensional n
-- [m, n] => 2 dimensional mxn
type Shape = [Int]

data ElementType = R | C

data Op
  = Var String              -- variable with an identifier
  | Param String            -- parameter with an identifier
  | Const Double            -- constant
  | Sum [NodeID]            -- sum
  | Prod [NodeID]           -- product
  | Scale NodeID NodeID     -- scale
  | RealImag NodeID NodeID -- complex from real and imaginary
  | ..

type Node = (Shape, ElementType, Op)

-- NodeID is a zero-overhead wrapper of Int
newtype NodeID = NodeID Int
-- IntMap Node is used instead of Map NodeID Node
-- for performance purpose
type ExpressionMap = IntMap Node
```

Type-safe modeling

McMaster University

- Lift metadata to type-level
  - Phantom type parameters

```haskell
newtype Builder a
  = Builder (State ExpressionMap a)
  deriving (Functor, Applicative, Monad)

type ExprBuilder = Builder NodeID
```

```haskell
newtype TypedExpr (sh :: [Nat]) (et :: ElementType)
  = TypedExpr ExprBuilder
```

  - `TypedExpr '[15, 15, 15] C`
    - 3D complex expression of shape 15 x 15 x 15
    - C is a type-level value with kind ElementType

Type-safe modeling

McMaster University

- Lift metadata to type-level

```
variable2D :: forall m n. (KnownNat m, KnownNat n) => String ->
↪  TypedExpr '[m, n] R
variable2D name = TypedExpr $ introduceNode (toShape @'[m, n], R,
↪  Var name)

-- define a variable
x = variable2D @20 @15 "x" -- TypedExpr '[20, 15] R
```

Type-safe modeling

McMaster University

- Operator specification

```
-- element-wise addition
(+) :: TypedExpr sh et -> TypedExpr sh et -> TypedExpr sh et

-- element-wise complex expression from real and imaginary parts
(+:) :: TypedExpr sh R -> TypedExpr sh R -> TypedExpr sh C

-- element-wise exp
exp :: TypedExpr sh R -> TypedExpr sh R

-- Fourier transform
ft :: TypedExpr sh C -> TypedExpr sh C
```

Type-safe modeling

McMaster University

- Operator specification

```
> x = variable2D @10 @10 "x" -- TypedExpr '[10, 10] R
> y = variable2D @20 @10 "y" -- TypedExpr '[20, 10] R
> z = x + y

• Couldn't match type ‘20’ with ‘10’
Expected type: TypedExpr '[10, 10] R
  Actual type: TypedExpr '[20, 10] R
```

Type-safe modeling

McMaster University

- Operator specification

```
(*.) ::
  (Scalable etScalar etVector) =>
  TypedExpr Scalar etScalar ->
  TypedExpr sh etVector ->
  TypedExpr sh etVector


type family Scalable (etScalar :: ElementType) (etVector ::
↪  ElementType) :: Constraint where
  Scalable R e = Satisfied  -- Real scalar can scale both real and
  ↪  complex vectors
  Scalable C C = Satisfied  -- Complex scalar can only scale
  ↪  complex vectors
  Scalable C R = TypeError (Text "Scaling a real vector by a
  ↪  complex scalar is not allowed")
```

Type-safe modeling

McMaster University

- Operator specification

```
> x = variable2D @10 @10 "x" -- TypedExpr '[10, 10] R
> y = variable2D @10 @10 "y" -- TypedExpr '[10, 10] R
> s1 = variable "s1" -- TypedExpr Scalar R
> s2 = variable "s1" -- TypedExpr Scalar R
> s = s1 +: s2  -- TypedExpr Scalar C
> z = s1 *. (x +: y) -- OK, TypedExpr '[10, 10] C
> z2 = s *. x -- Type Error

• Scaling a real vector by a complex scalar is not allowed
• In the expression: s *. x
  In an equation for ‘z2’: z2 = s *. x
```

Type-safe modeling

McMaster University

- Operator specification

```
project ::
  IsSelectors selectors =>
  TypedExpr inputShape et ->
  TypedExpr (ProjectionShape inputShape selectors) et
```

Type-safe modeling

McMaster University

- Operator specification

```
x = variable2D @20 @30
-- y = x[0:9, 0:5], has type TypedExpr '[10, 6] R
y = project @'[Range 0 9 1, Range 0 5 1] x
-- z = x[1, 5], has type TypedExpr Scalar R
z = project @'[At 1, At 5] x
-- t = x[1:2, :], has type TypedExpr '[2, 30] R
t = project @'[Range 1 2 1, All] x
```

Type-safe modeling

McMaster University

- Operator specification

```
x = variable2D @20 @30
-- Type Error: end index 30 is out of range in the 1st dimension
y = project @'[Range 10 30 1, At 5] x

• Invalid end index 30, must be in range (0, 20)
```

Type-safe modeling

McMaster University

- Catching errors early
- Modeling correctness
- Interactivity

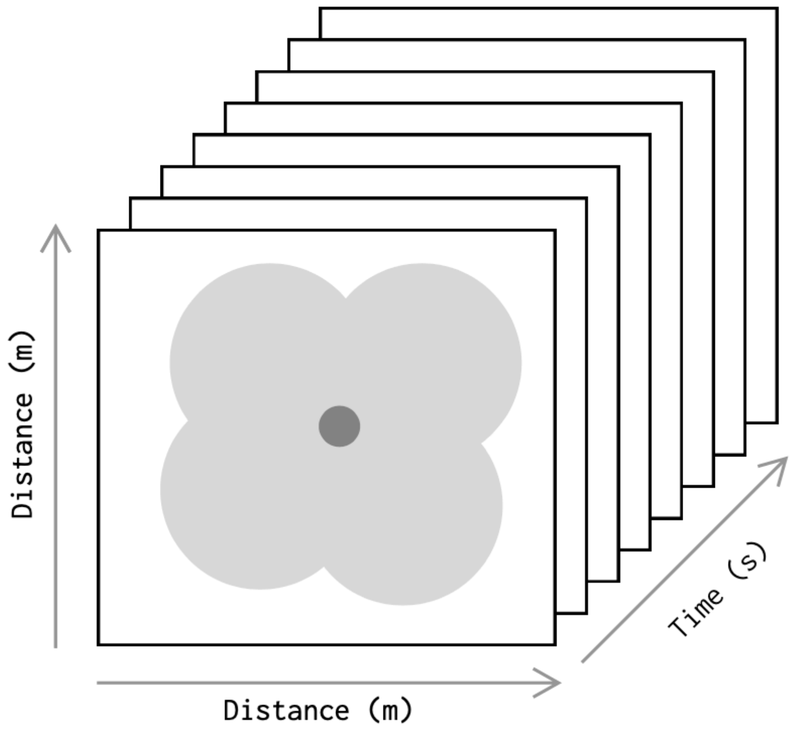
```
    x :: TypedExpr '[20, 20] R
78  x = variable2D @20 @20 "x"
    y :: TypedExpr '[20, 20] R
79  y = variable2D @20 @20 "y"
    z :: TypedExpr '[20, 20] R
80  z = x + _
```

HashedExpression.hs 1 of 1 problem

```
• Found hole: _ :: TypedExpr '[20, 20] R
• In the second argument of ‘(+)’, namely ‘_’
  In the expression: x + _
  In an equation for ‘z’: z = x + _
• Relevant bindings include
    z :: TypedExpr '[20, 20] R
  Valid hole fits include
    x
    y
```

Type-safe modeling

- More metadata
  - Physical units (domain, range)
  - Sampling step



Type-safe modeling

McMaster University

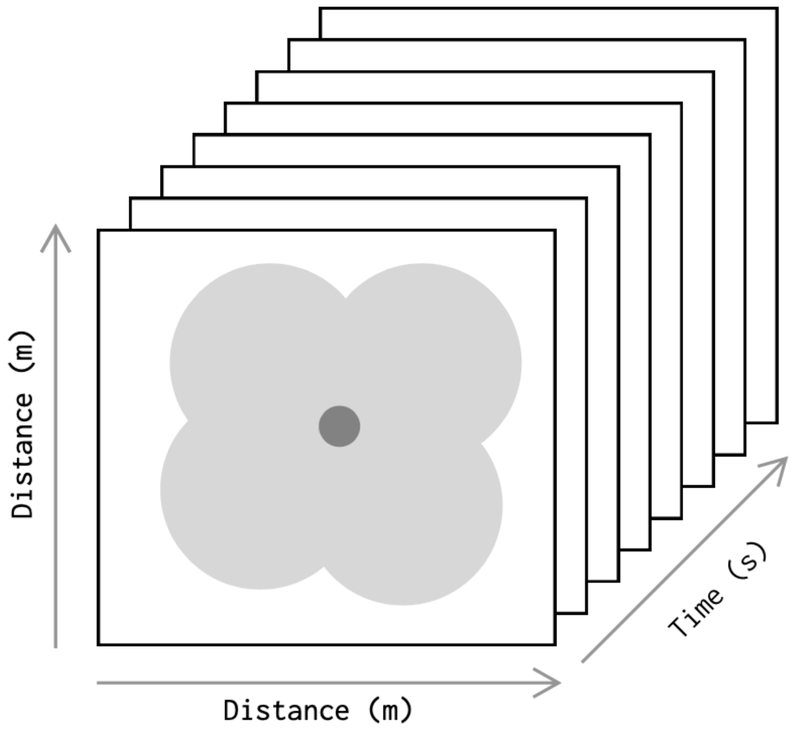
- More metadata
  - Physical units (domain, range)
  - Sampling step

```haskell
data Sampling
  = D
      Nat          -- number of samples
      SampleStep -- sampling steps
      Unit         -- domain unit

newtype
  UnitExpr
    (ds :: [Sampling])
    (unit :: Unit)       -- range unit
    (et :: ElementType) -- real or copmlex
  = UnitExpr ExprBuilder
```

Type-safe modeling

McMaster University

```
v =
  variable
    @'[ D 256 (1 :/ 2000) Meter,    -- 1st dimension: distance
        D 256 (1 :/ 2000) Meter,    -- 2nd dimension: distance
        D 100 (1 :/ 60) Second      -- 3rd dimension: time
      ]
    @Candela                        -- range unit: luminance
    "x"
```



Type-safe modeling

McMaster University

- Operators act on units & sampling steps

```
-- x has type UnitExpr '[D 10 (1 ':/: 2000) Meter] Kilogram R
x = variable @'[D 10 (1 :/ 2000) Meter] @Kilogram "x"

-- x has type UnitExpr '[D 10 (1 ':/: 2000) Meter] Candela R
y = variable @'[D 10 (1 :/ 2000) Meter] @Candela "y"

-- z has type UnitExpr '[ 'D 10 (1 ':/: 2000) (Meter :*: Candela)]
↪  Kilogram 'R
z = x * y
```

Type-safe modeling

McMaster University

- Operators act on units & sampling steps

```
-- x has type UnitExpr '[D 10 (1 :/: 2000) Meter] Ampere R
x = variable @'[D 10 (1 :/ 2000) Meter] @Ampere "x"

-- y has type UnitExpr '[D 10 (1 :/: 2000) Meter] Ampere R
y = variable @'[D 10 (1 :/ 2000) Meter] @Ampere "y"

-- z has type UnitExpr '[D 10 (200 :/: 1) (Meter :^: (Negative 1))]
↪  Ampere R
z = dft $ (x +: y)
```

Core

McMaster University

- Verify the model
  - Objective, contraints
- Perform rewriting and simplification

Rewriting and simplification

McMaster University

- Obtain equivalent but simpler expressions
  - Reduce evaluation time
  - Share computation
    - e.g., *x + (y + z)* and *(x + z) + y* both rewritten to *sum(x, y, z)*

Rewriting and simplification

McMaster University

- A rewriting system
  - Matching and replacing

```
complexNumRules :: [Substitution]
complexNumRules =
  [ xRe (x +: y) |.~~> x,
    xIm (x +: y) |.~~> y,
    (x +: y) * (zero +: zero) |.~~> zero +: zero
  ]
```

Rewriting and simplification

McMaster University

- Identify special patterns
  - NNs

```
nnRule :: Substitution
nnRule = log (1 / (1 + exp (- x))) |.~~> logSigmoid x
```

  - Special instructions
  - Hardware accelerator

Core

McMaster University

- Verify the model
  - Objective, contraints
- Perform rewriting and simplification
- Compute derivatives

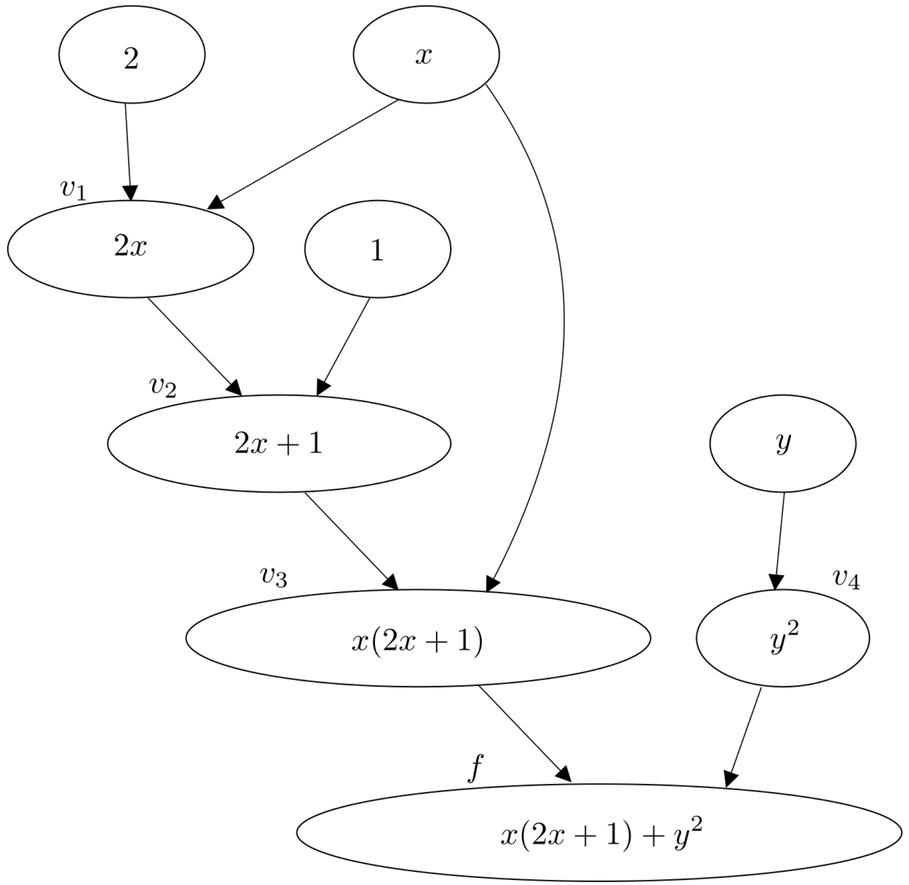
Computing derivatives

McMaster University

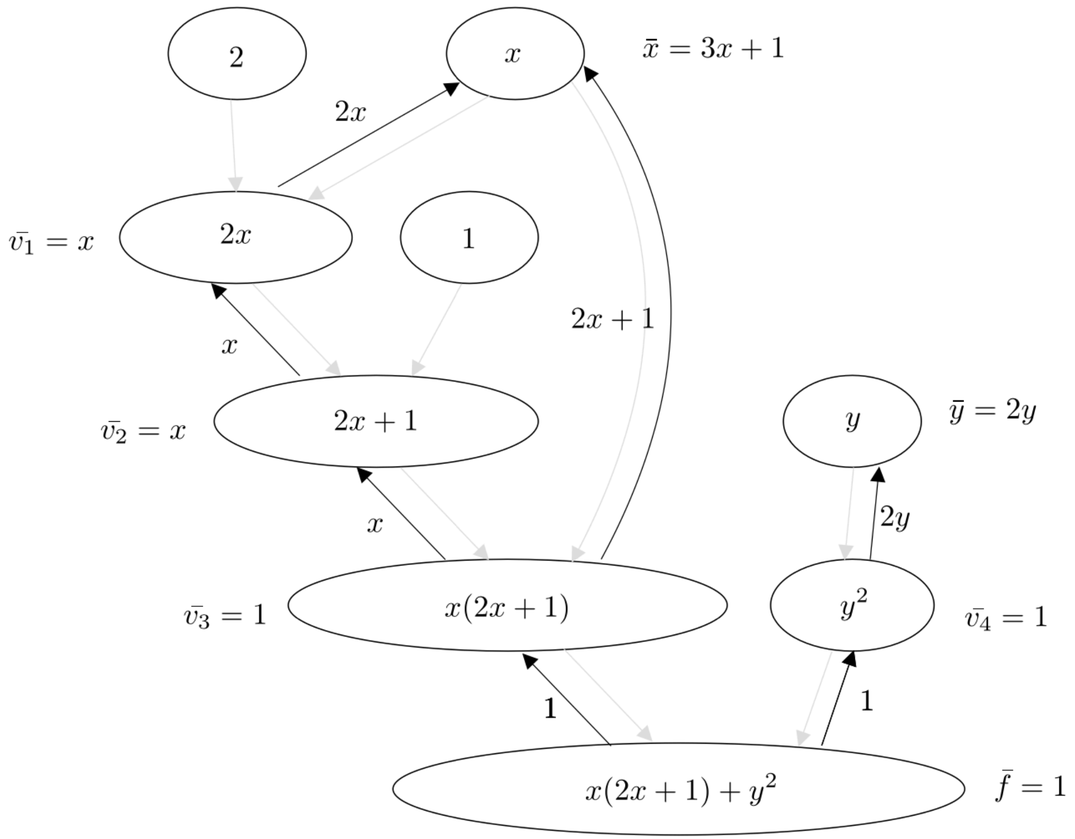
- Reverse accumulation method
- Producing symbolic derivative expressions
  - Indexed in the same expression graph (as the original objective function)

Computing derivatives

$$f = x(2x + 1) + y^2$$



Computing derivatives



$\rightarrow \overline{f} = 1$

$\frac{\partial f}{\partial v_4} = 1 \Rightarrow add(\overline{v_4}, 1 * 1)$

$\frac{\partial f}{\partial v_3} = 1 \Rightarrow add(\overline{v_4}, 1 * 1)$

$\rightarrow \overline{v_4} = 1 * 1 = 1$

$\frac{\partial v_4}{\partial y} = 2y \Rightarrow add(\overline{y}, 1 * 2y)$

$\rightarrow \overline{v_3} = 1 * 1 = 1$

$\frac{\partial v_3}{\partial v_2} = x \Rightarrow add(\overline{v_2}, 1 * x)$

$\frac{\partial v_3}{\partial x} = 2x + 1 \Rightarrow add(\overline{x}, 1 * (2x + 1))$

$\rightarrow \overline{v_2} = 1 * x = x$

$\frac{\partial v_2}{\partial v_1} = 1 \Rightarrow add(\overline{v_1}, x * 1)$

$\rightarrow \overline{v_1} = x * 1 = x$

$\frac{\partial v_1}{\partial x} = 2 \Rightarrow add(\overline{x}, x * 2)$

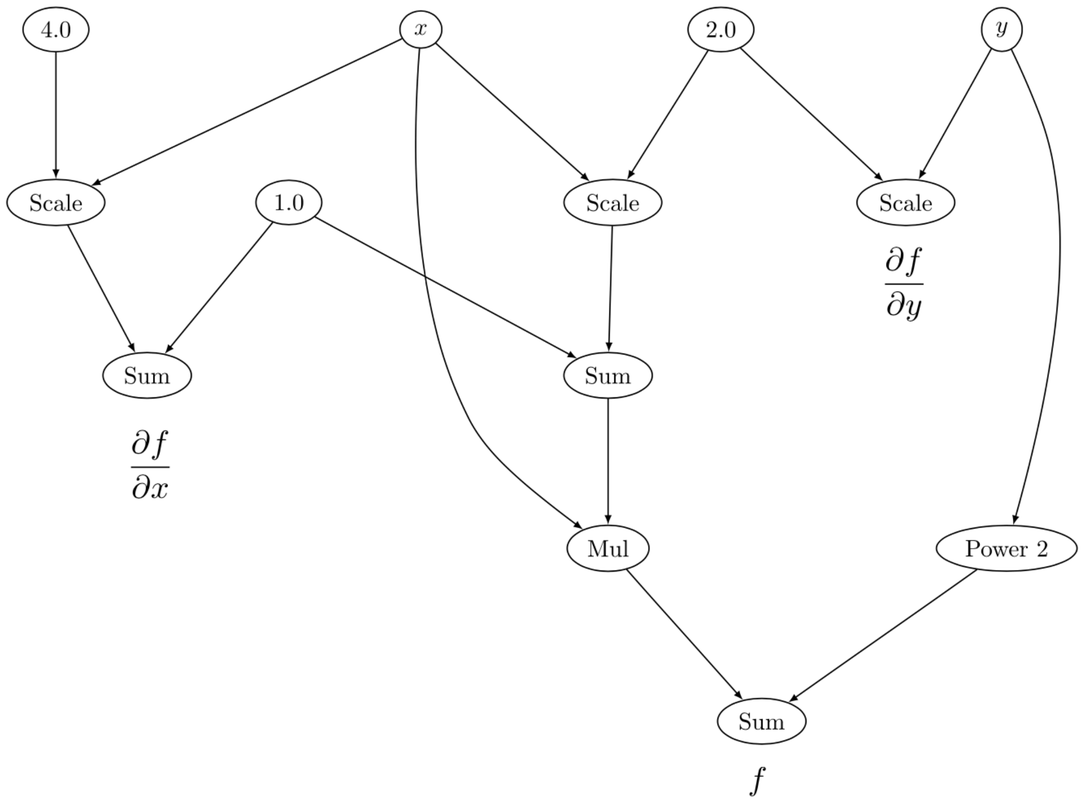
$\rightarrow \overline{y} = 1 * 2y = 2y$

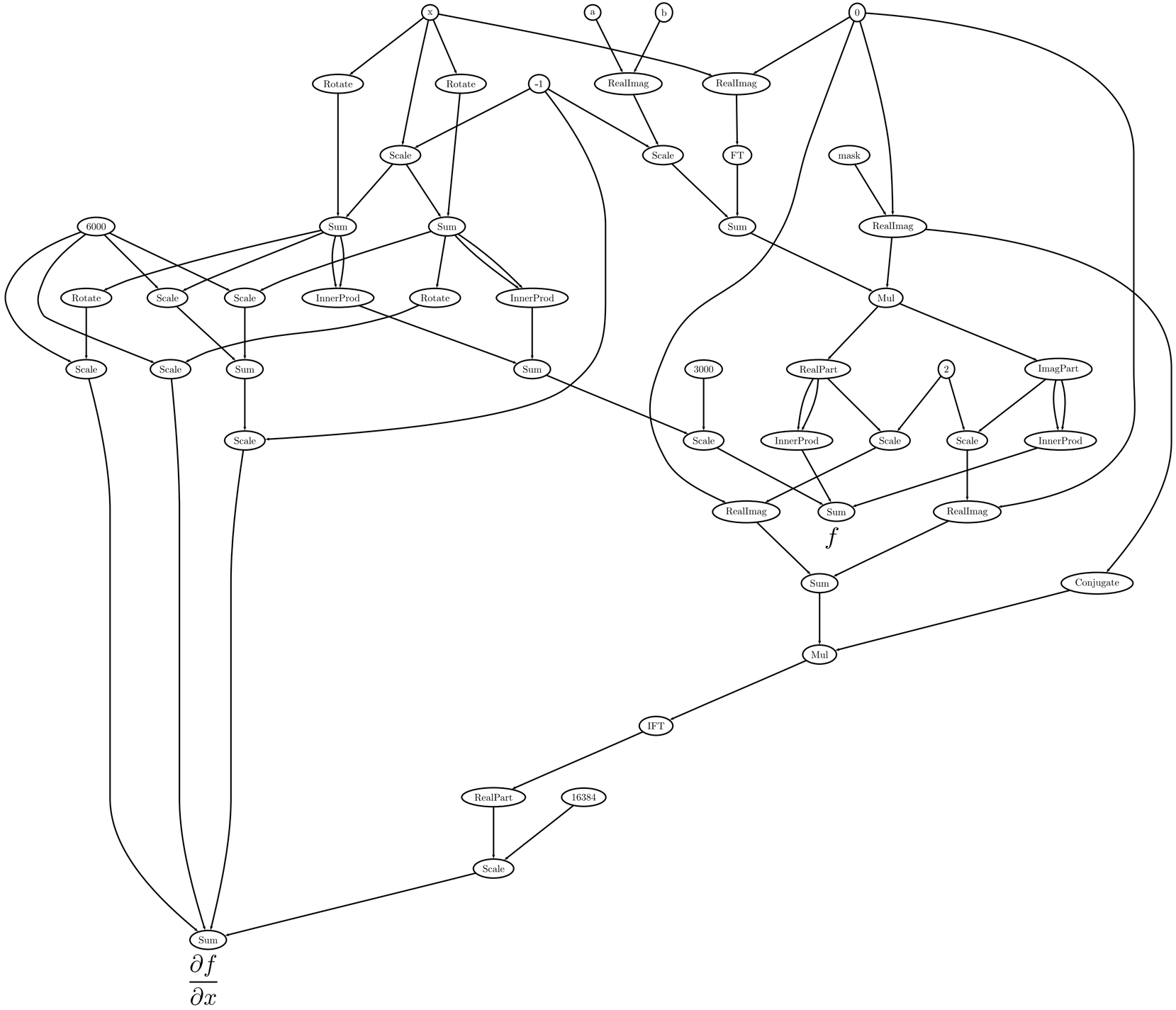
$\rightarrow \overline{x} = 1 * (2x + 1) + x * 2 = 3x + 1$

Computing derivatives

- Trivial computations (e.g., multiply with 1) get simplified away
- Shared computations between objective function and its derivatives

$$f = x(2x + 1) + y^2$$




x
a
b
0
Rotate
Rotate
-1
RealImag
RealImag
Scale
Scale
FT
mask
6000
Sum
Sum
Sum
RealImag
Rotate
Scale
Scale
InnerProd
Rotate
InnerProd
Mul
Scale
Scale
Sum
Sum
3000
RealPart
2
ImagPart
Scale
Scale
InnerProd
Scale
Scale
InnerProd
RealImag
Sum
RealImag
f
Sum
Conjugate
Mul
IFT
RealPart
16384
Scale
Sum
$\frac{\partial f}{\partial x}$

Computing derivatives

McMaster University

- No “expression swelling”
  - Working on a single expression lookup table
  - Same expressions always indexed to the same node

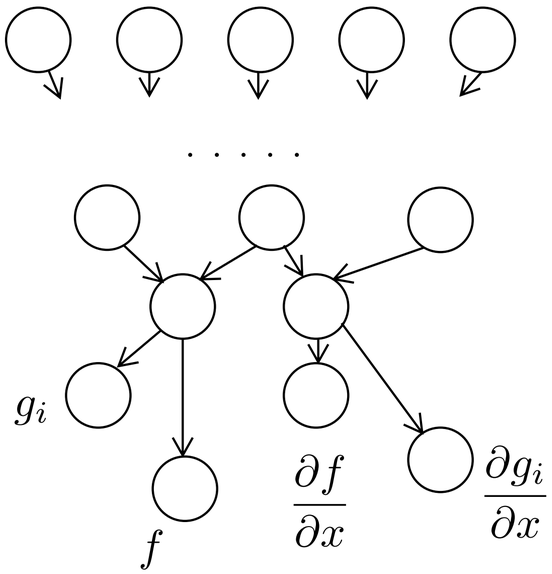
Core

McMaster University

- Verify the model
  - Objective, contraints
- Perform rewriting and simplification
- Compute derivatives
- Construct evaluation graph

  —> Code generation

Code generation



GENERATE →

```
evaluate_objective()
evaluate_objective_and_gradient()
evaluate_general_constraints()
evaluate_general_constraints_jacobian()
evaluate_everything()
. . . . .
```

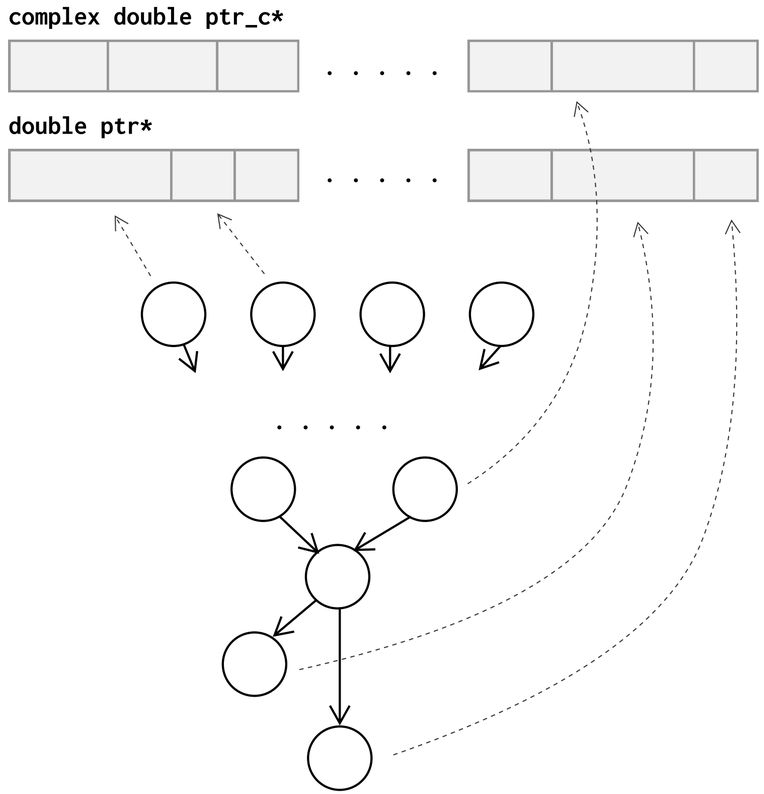
USED BY →

OPTIMIZATION SOLVERS

(LBFGS, LBFGS-B, IPOPT, …)

Code generation

- Low level C/C++ evaluation code
  - Memory allocation
  - Evaluate in topological order
  - C99 code generator
    - Simple for-loops
    - FFTW3



Code generation

McMaster University

- Using generated codes
  - problem.c

```
...
const char* var_name[NUM_HIGH_DIMENSIONAL_VARIABLES] = { "theta1",
↪  "theta0" };
const int var_size[NUM_HIGH_DIMENSIONAL_VARIABLES] = { 1, 1 };

const int var_offset[NUM_HIGH_DIMENSIONAL_VARIABLES] = { 0, 1 };
const int partial_derivative_offset[NUM_HIGH_DIMENSIONAL_VARIABLES]
↪  = { 1075, 1074 };
const int objective_offset = 1076;

double ptr[MEMORY_NUM_DOUBLES];
complex double ptr_c[MEMORY_NUM_COMPLEX_DOUBLES];
....

void evaluate_partial_derivatives_and_objective() { ... };
void evaluate_objective() { ... };
void evaluate_partial_derivatives() { ... };
void evaluate_scalar_constraints() { ... };
void evaluate_scalar_constraints_jacobian() { ... };
```

Code generation

McMaster University

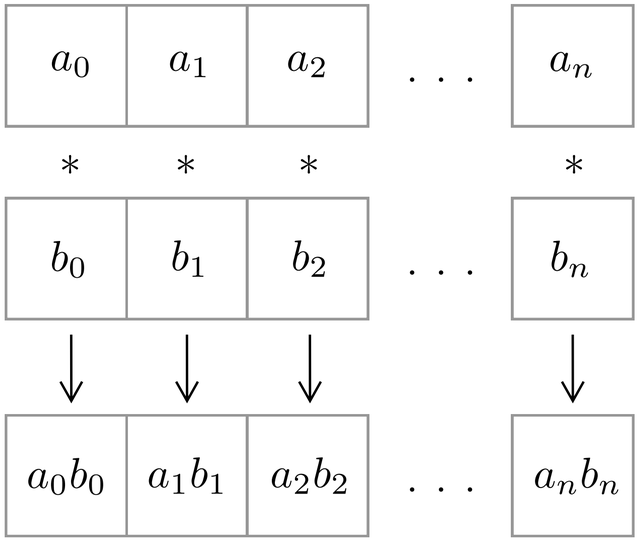
- Using generated codes
  - Adapter
  - Available
    - Gradient descent
    - L-BFGS
    - L-BFGS-B
    - IPOPT

```
#include <lbfgs.h>
#include "problem.c"
...

static lbfgsfloatval_t evaluate(void *instance,
    const lbfgsfloatval_t *x,
    lbfgsfloatval_t *g,
    const int n,
    const lbfgsfloatval_t _step) {
  evaluate_partial_derivatives_and_objective();
  int i;
  int cnt = 0;
  for (i = 0; i < NUM_HIGH_DIMENSIONAL_VARIABLES; i++) {
    memcpy(g + cnt, ptr + partial_derivative_offset[i], var_size[i] * sizeof(double));
    cnt += var_size[i];
  }

  return ptr[objective_offset];
}

int main() {
  ..
  lbfgsfloatval_t *x = ptr + VARS_START_OFFSET;.
  ret = lbfgs(N, x, &fx, evaluate, progress, NULL, &param);
  ...
}
```

Code generation

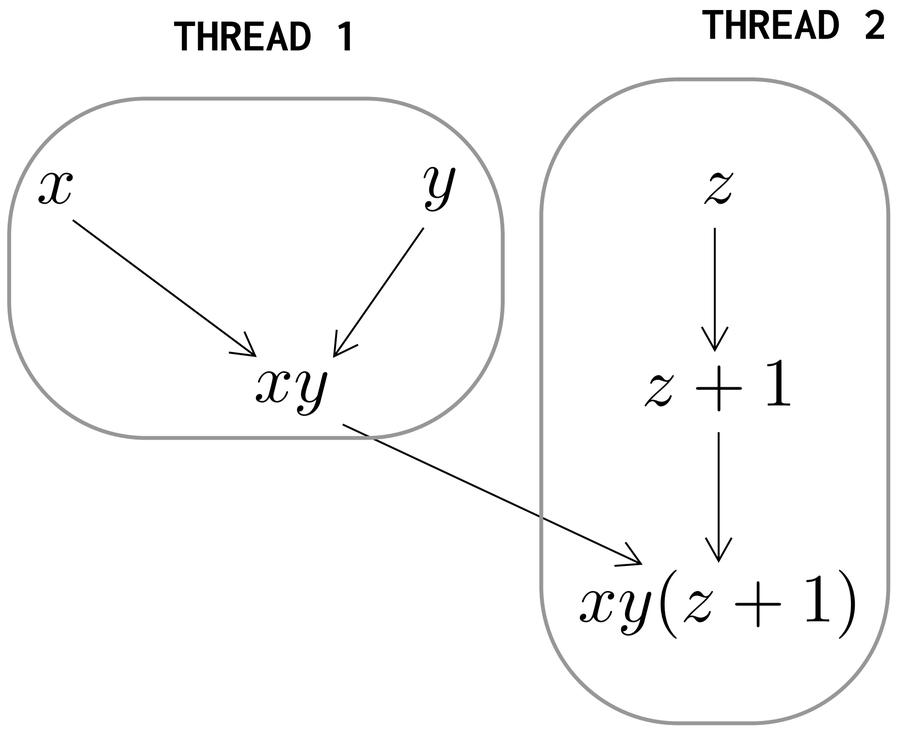
McMaster University

- Parallelization opportunity
  - Shape: grid of numbers
  - Point-wise operation (addition, multiplication)
  - Embarrassing parallel
    - SIMD
    - GPU Acceleration



Code generation

- Parallelization opportunity
  - Code graph analysis
  - Multi-threading



Outline

McMaster University

- Context, problem and introduction
- Proposal
- Implementation
- **Conclusion and future work**

Conclusion

McMaster University

- Type-safe modeling with Haskell type system
- Optimize computations with symbolic representation, hashing and simplification
- Compute derivatives without the expression swelling issue
- High-performance via low level C code generation

Future work

McMaster University

- Parallelized and more memory-efficient code generators
- Improve the type system
- Allow rules to add simplification rules
  - Domain specific
- Extensibility