Benchmarking Binding

Stephanie Weirich

Lambda Calculus implementation

```
data Exp = Var Var          -- x
         | Lam (Bind Exp)   -- λx.a
         | App Exp Exp      -- (a b)
```

```
aeq   :: Exp -> Exp -> Bool
subst :: Var -> Exp -> Exp -> Exp
nf    :: Exp -> Exp
…
```

Multiple Binding Representations

- De Bruijn

```
type Var = Int
newtype Bind a = Bind a
```

- Locally Nameless

```
type Var = Bound Int | Free Name
newtype Bind a = Bind a
```

- Named

```
type Var = String
data Bind a = Bind Var a
```

- HOAS

```
type Var = Exp
type Bind Exp = Exp -> Exp
```

Other Variants: Nested DeBruijn, Well-scoped, Well-typed, Weak HOAS, PHOAS

Binding Library (unbound-generics)

```
import LocallyNameless.UnboundGenerics
data Exp = Var (Name Exp)               -- x
         | Lam (Bind (Name Exp) Exp)    -- λx.a
         | App Exp Exp                  -- (a b)
             deriving (Show, Generic)
instance Alpha Exp
  -- aeq :: Exp -> Exp -> Bool
  -- bind :: Name Exp -> Exp -> Bind (Name Exp)
  -- unbind :: Bind (Name Exp) -> FreshM (Name Exp, Exp)
instance Subst Exp Exp where
  isvar (Var x) = Just (SubstName x)
  isvar _ = Nothing
  -- subst :: Var -> Exp -> Exp -> Exp
```

Reduction w/ locally nameless terms

```
nfd :: Exp -> FreshM Exp
nfd (Var x) = return (Var x)
nfd (Lam e) = do
  (x, e') <- unbind e
  e1 <- nfd e'
  return $ Lam (bind x e1)
nfd (App f a) = do
  f' <- whnf f
case f' of
  Lam b -> do
    (x, b') <- unbind b
    nfd (subst x a b')       == free variable substitution
_ -> App <$> nfd f' <*> nfd a
```

```
nf :: Exp -> Exp
nf = runFreshM . nfd
```

What is the best way to do this?

Multiple libraries available on hackage

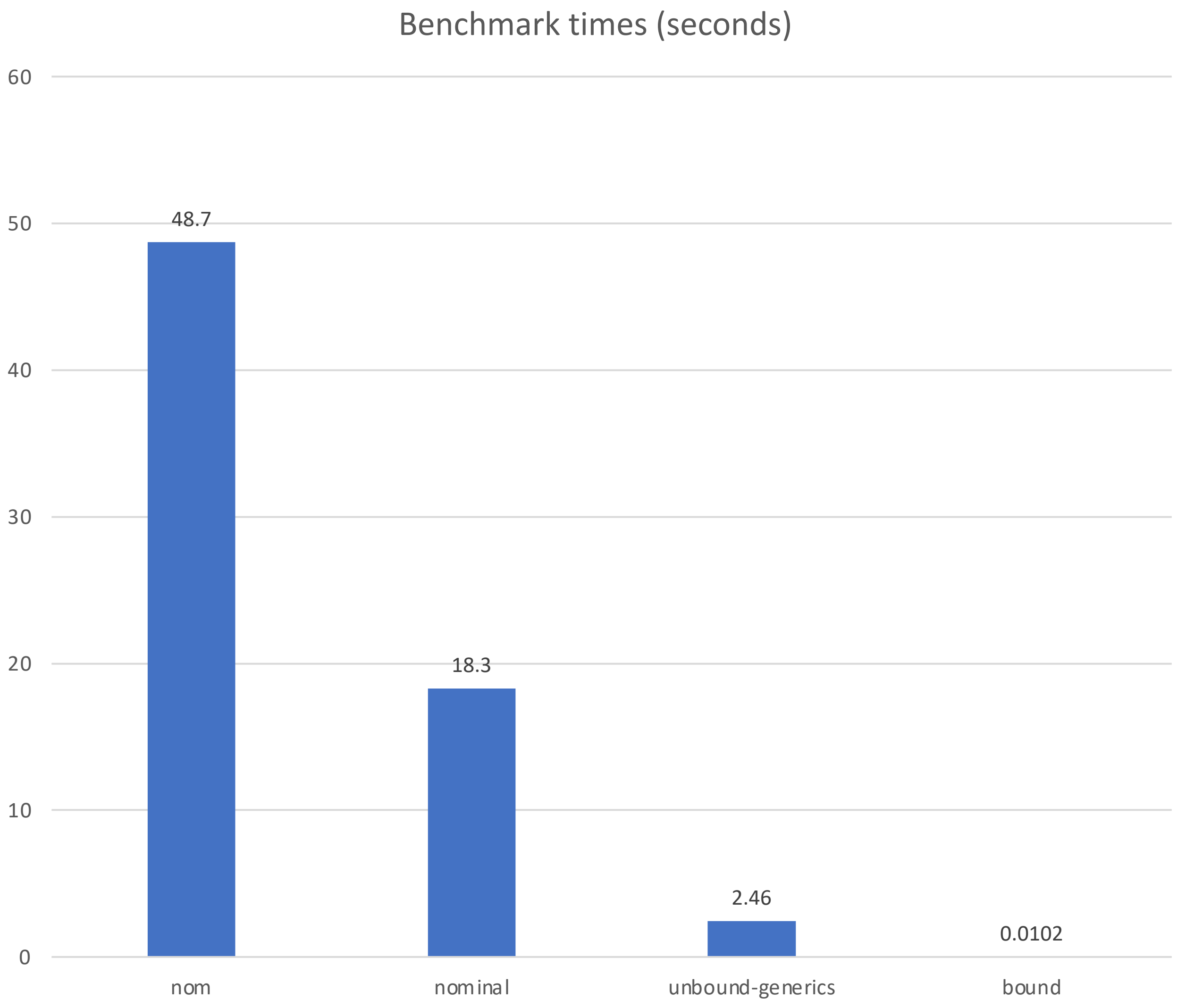
- unbound-generics

  Locally nameless
- bound

  Nested de Bruijn
- nominal

  Named
- nom

  Named



GHC 8.8.3, MacBook pro, 2.4 GHz 8-Core Intel Core i9, 64 GB, measured using criterion

Reduction w/ locally nameless terms

```
nfd :: Exp -> FreshM Exp
nfd (Var x) = return (Var x)
nfd (Lam e) = do
  (x, e') <- unbind e
  e1 <- nfd e'
  return $ Lam (bind x e1)
nfd (App f a) = do
  f' <- whnf f
case f' of
  Lam b -> do
    nfd (instantiate a b)      -- bound variable substitution
                               -- (can be defined generically)
_ -> App <$> nfd f' <*> nfd a
```

```
nf :: Exp -> Exp
nf = runFreshM . nfd
```

Implementation interface

```
data LambdaImpl =
  forall a. NFData a => LambdaImpl
  { impl_name :: String,
    impl_fromLC :: LC IdInt -> a,
    impl_toLC :: a -> LC IdInt,
    impl_nf :: a -> a,
    impl_aeq :: a -> a -> Bool
  }
```

Benchmark Implementations

- Lennart.* – 4 original (DeBruijn, Simple, Unique, HOAS)
- DeBruijn.* – 28 versions
  - single vs. multiple substitution (plus various optimizations)
  - strict vs. lazy datatypes
  - plain vs. nested vs. scoped vs. well-typed
  - typeclass based interface? generic programming?
- LocallyNameless.*, Unbound.* – 20 versions
- Named.* – 10 versions
- NBE.*, DeBruijn.Krivine – 9 versions
  - Direct impl of normalization, w/o substitution (env, NBE, abstract machine)

Benchmark Platform

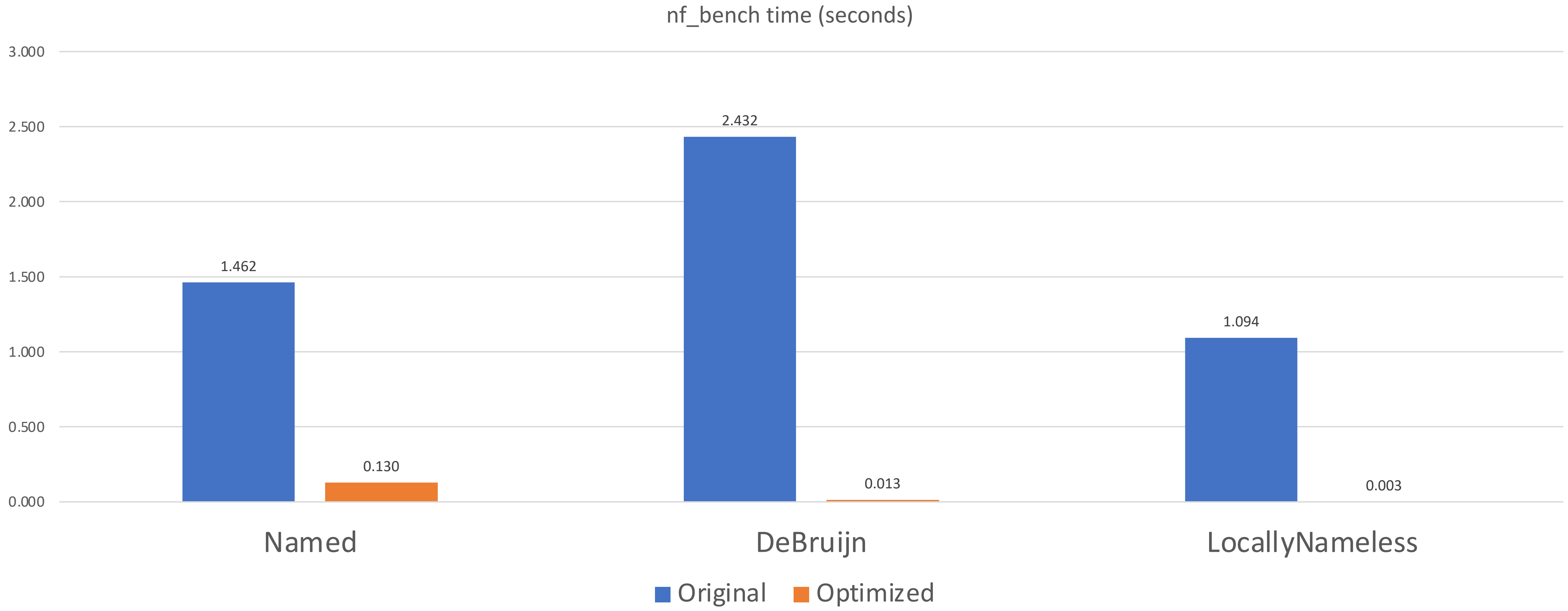
- https://github.com/sweirich/lambda-n-ways/
- Forked from Lennart Augustsson's Lambda-Calculus Four Ways
  - DeBruijn indices
  - Named (rename to avoid capture)
  - Named (globally unique)
  - HOAS
- Common interface
  - Representation of untyped lambda-calculus
  - Conversion to/from string representation
  - Alpha-equivalence
  - Full normalization (based on substitution)

Benchmarks

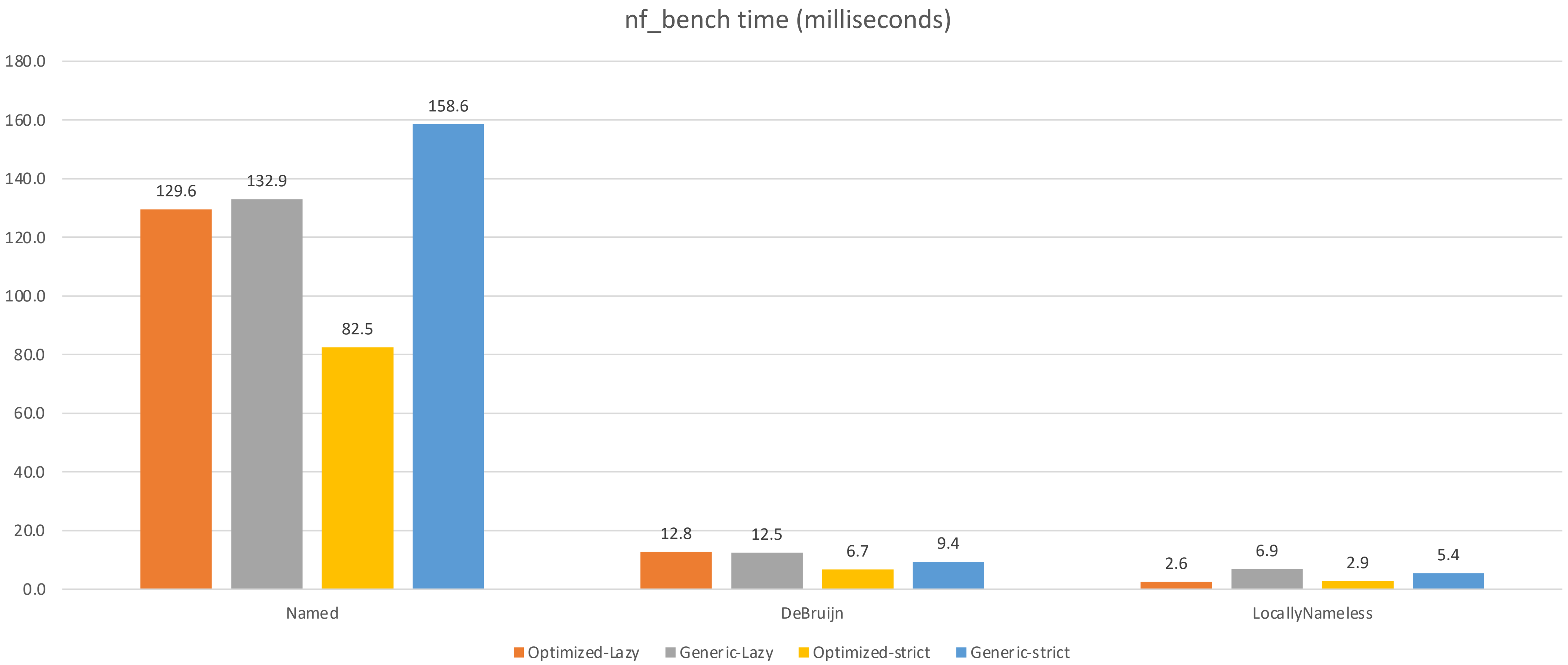
- Full results available: http://www.cis.upenn.edu/~sweirich/wits22/
- full normalization of large term (nf_bench.html)
  - Church encoding of "6! == sum [1 .. 37] + 17"
  - needs **119,697 beta-reductions,** Binding depth 25
- full normalization of random terms
  - random15_bench.html
  - random20_bench.html
- alpha-equality
  - freshen large term, then compare (aeq_bench.html)
  - compare large term with itself (aeqs_bench.html)
- conversion to/from named representation (conv_bench.html)

Benchmark Observations

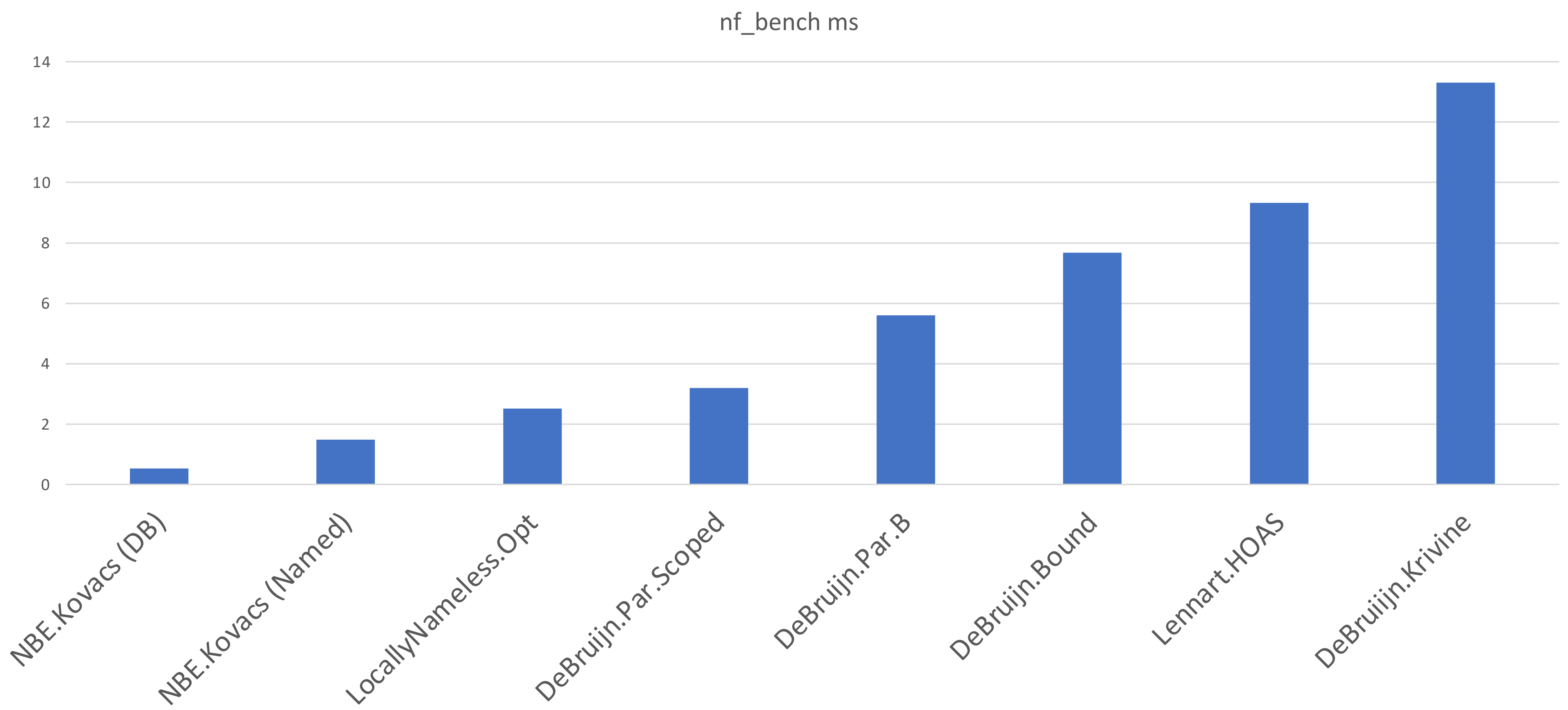
Comparison: Original vs. optimized



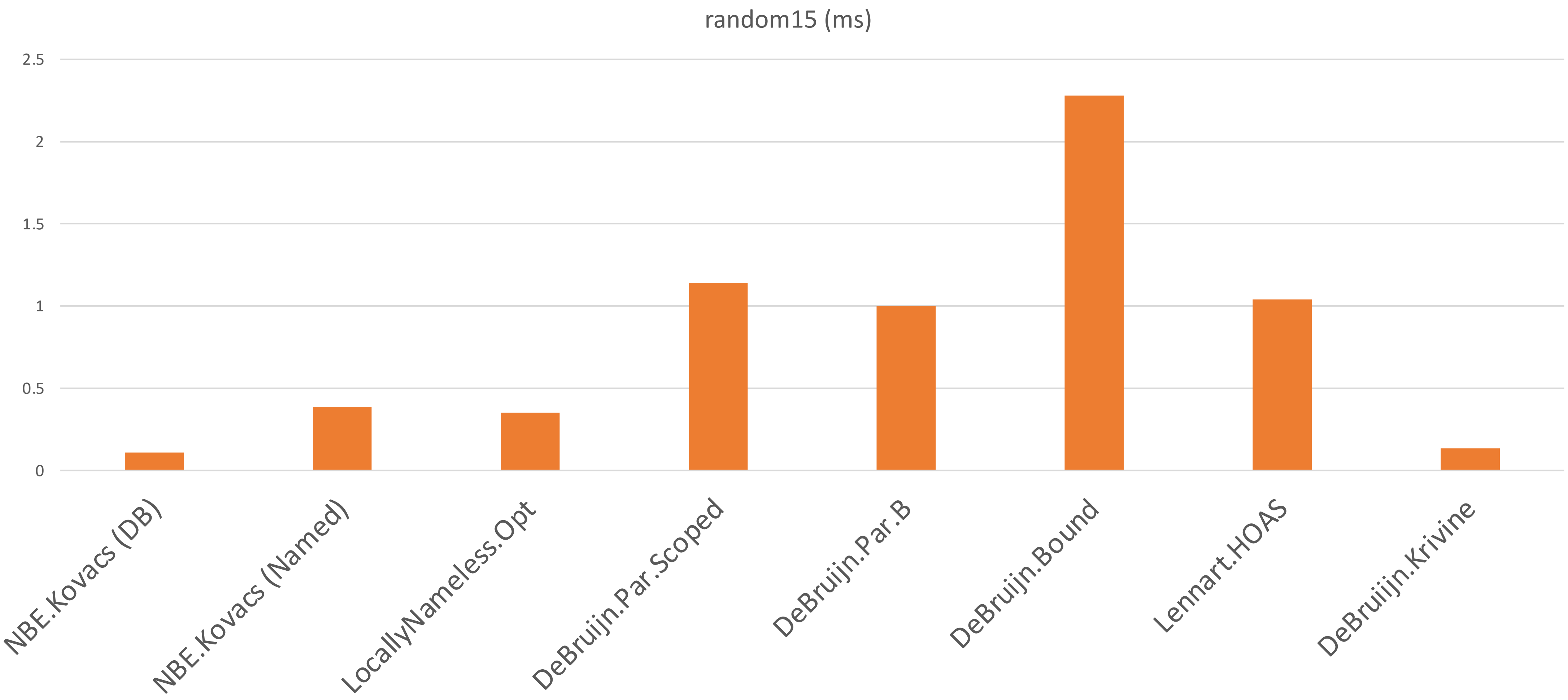
Strictness annotations and optimization



Normalization for large term



Normalization of random terms



What is used in practice?

How do you represent binding? What do you do with it?

Are these the right operations to benchmark?

substitution, aeq, normalization, etc?

What optimizations help?

Delaying substitutions, laziness, etc

How important/expensive is library support?

Can we make our implementations look like our papers? Should we?