# Building blocks for exact and approximate inference

Jacques Carette[2], Praveen Narayanan[1], Wren Romano[1],
Chung-chieh Shan[1], Robert Zinkov[1]

[1] Indiana University
[2] McMaster University

## Motivation

We address the need for *modular inference methods* that can be reused over models and composed to build more methods. We present a collection of black-box methods implemented as probabilistic-program transformations. Modular building-blocks such as algebraic simplification can be reused to mechanically derive an *efficient program* from the original one. This collection creates a search space of inference strategies that combines exact and approximate techniques.

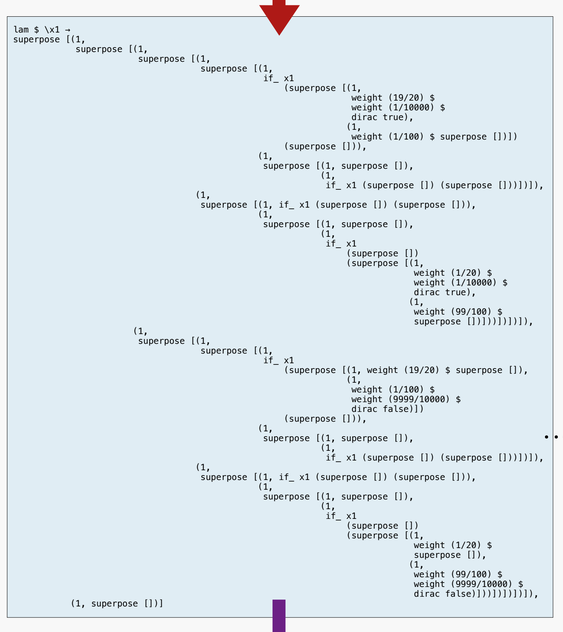
## Workflow examples using Hakaru

**Model**

Burglary-alarm model (Pearl 1988)

```
burg :: (Mochastic repr) ⇒ repr (HMeasure (HPair HBool HBool))
burg = bern 0.0001 `bind` \burglary →
       bern (if_ burglary 0.95 0.01) `bind` \alarm →
       dirac (pair alarm burglary)
```

disintegrate



simplify

```
lam $ \x1 →
if_ x1
    (superpose [(19/200000, dirac true), (9999/1000000, dirac false)])
    (superpose [(1/200000, dirac true), (989901/1000000, dirac false)])
```

Linear sequential model for device calibration

```
thermometer :: (Mochastic repr)
            ⇒ repr (HMeasure (HPair (HPair HReal HReal)
                                    (HPair HProb HProb)))
thermometer =
    liftM unsafeProb (uniform 3 8) `bind` \noiseT →
    liftM unsafeProb (uniform 1 4) `bind` \noiseM →
    normal 21 noiseT `bind` \t1 →
    normal t1 noiseM `bind` \m1 →
    normal t1 noiseT `bind` \t2 →
    normal t2 noiseM `bind` \m2 →
    dirac (pair (pair m1 m2) (pair noiseT noiseM))
```

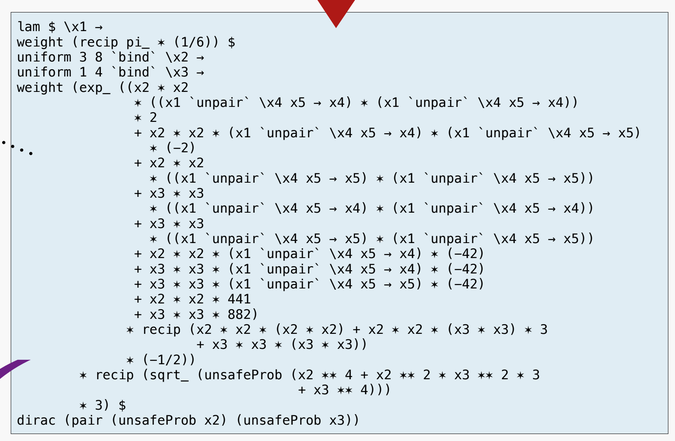
disintegrate

```
lam $ \x1 →
x1 `unpair` \x2 x3 →
uniform 3 8 `bind` \x4 →
normal 21 (unsafeProb x4) `bind` \x5 →
uniform 1 4 `bind` \x6 →
normal x5 (unsafeProb x4) `bind` \x7 →
weight (exp_ (−(x3 − x7) ∗ (x3 − x7)
                / fromProb (2 ∗ pow_ (unsafeProb x6) 2))
        / unsafeProb x6
        / sqrt_ (2 ∗ pi_)) $
weight (exp_ (−(x2 − x5) ∗ (x2 − x5)
                / fromProb (2 ∗ pow_ (unsafeProb x6) 2))
        / unsafeProb x6
        / sqrt_ (2 ∗ pi_)) $
dirac (pair (unsafeProb x4) (unsafeProb x6))
```

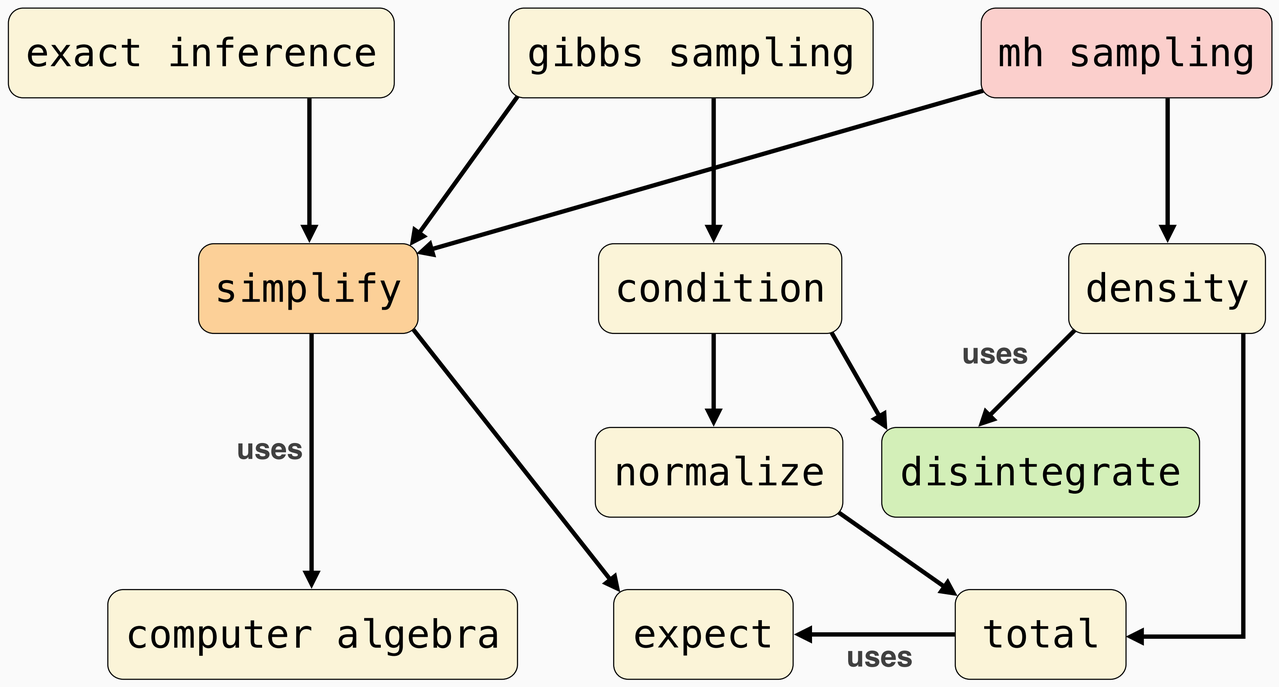
**Posterior**

**Simplified posterior**

simplify

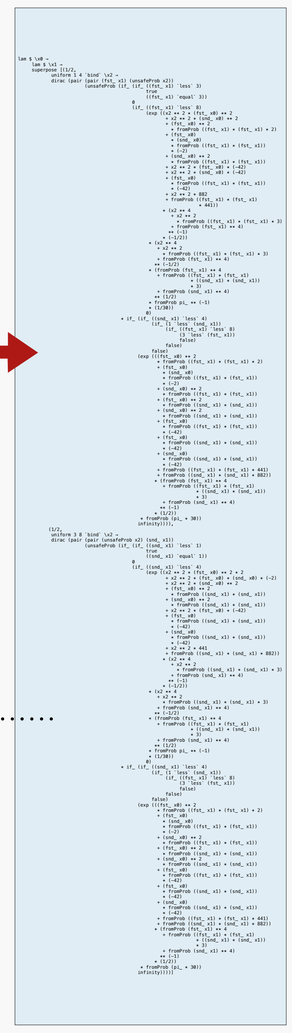


**Dependencies among building blocks**




mh sampling

simplify



**MH transition kernel**

**Simplified transition kernel**

| Source of code | Average run time (ms) |
|---|---|
| Generated by disintegrator | 2015 ± 4 |
| Generated, then automatically simplified | 569 ± 4 |
| Written by hand | 529 ± 10 |
| Expressed in WebPPL | 948 ± 8 |

## Related work

We share the modularity concerns of Venture (Mansinghka et al. 2014) and WebPPL (Goodman and Stuhlmüller 2014). Ścibior et al. (2015) use a monadic probabilistic language to describe and compose sophisticated models and inference methods. We describe in detail the techniques of disintegration (Shan and Ramsey 2015) and simplification (Carette and Shan 2015).

This research was supported by DARPA grant FA8750-14-2-0007, NSF grant CNS-0723054, Lilly Endowment, Inc. (through its support for the Indiana University Pervasive Technology Institute), and the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU is also supported in part by Lilly Endowment, Inc.