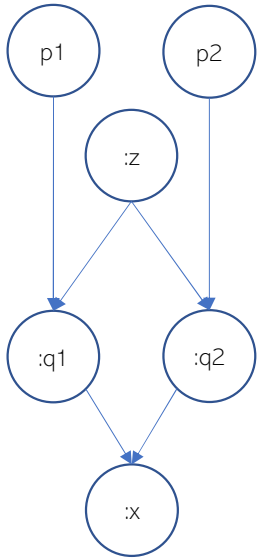
Trace-based probabilistic programming systems allow for the flexible expression of probability measures using addressed randomness and nonstandard execution.

Listing 1. Model program

```
function model(p1::Float64, p2::Float64)
    z = rand(:z, Normal(0.0, 1.0))
    q1 = rand(:q1, Normal(p1, exp(z)))
    q2 = rand(:q2, Normal(p2, exp(z)))
    x = rand(:x, Normal(q1 + q2, 1.0))
    x
end
```

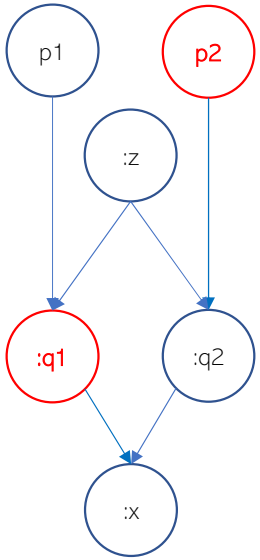


Listing 2. SSA form IR

```
1: (%1, %2, %3)
  %4 = Normal(0.0, 1.0)
  %5 = rand(:z, %4)
  %6 = exp(%5)
  %7 = Normal(%2, %6)
  %8 = rand(:q1, %7)
  %9 = exp(%5)
  %10 = Normal(%3, %9)
  %11 = rand(:q2, %10)
  %12 = %8 + %11
  %13 = Normal(%12, 1.0)
  %14 = rand(:x, %13)
  return %14
```

```
ret, cl = simulate(model, 3.0, 5.0)
display(cl.trace)

# produces
# ──────────────────────────────────────
#
#              Address Map
#
# (:x,) = 7.613727286549811
# (:q2,) = 6.430497321311673
# (:z,) = -0.26237761570473084
# (:q1,) = 2.1390457952084216
# ──────────────────────────────────────
```



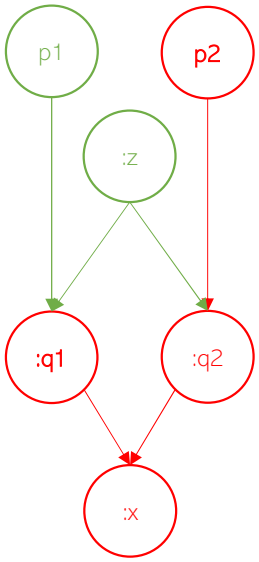
Listing 3. Trace update operation

```
obs = static([(:q1, ) => 5.0])
ret, cl, _ = update(obs, cl, Δ(3.0, NoChange()), Δ(6.0, ScalarDiff(1.0)))

# produces
# ──────────────────────────────────────
#
#              Address Map
#
# (:x,) = 7.605846180514769
# (:q2,) = 4.595901540194288
# (:z,) = -0.6770701252616422
# (:q1,) = 5.0
# ──────────────────────────────────────
```

Inference in these systems is expressed using nonstandard re-execution of the program (with constraints on address and possibly changed arguments). This re-execution can be expensive if the inference primitive has to re-visit redundant computation (choices or computation not affected by changes).

This allows for specialization of inference operations to programs – optimizing performance for iterative algorithms in inference programming.



Listing 8. Original IR

```
1: (%1, %2, %3)
  %4 = Normal(0.0, 1.0)
  %5 = rand(:z, %4)
  %6 = exp(%5)
  %7 = Normal(%2, %6)
  %8 = rand(:q1, %7)
  %9 = exp(%5)
  %10 = Normal(%3, %9)
  %11 = rand(:q2, %10)
  %12 = %8 + %11
  %13 = Normal(%12, 1.0)
  %14 = rand(:x, %13)
  return %14
```

Listing 9. Inferred IR

```
1: (%1 :: var"#1#2", %2 :: NoChange, %3 :: Change)
  %4 = (Normal)(0.0, 1.0) :: NoChange
  %5 = (rand)(:z, %4) :: NoChange
  %6 = (exp)(%5) :: NoChange
  %7 = (Normal)(%2, %6) :: NoChange
  %8 = (rand)(:q1, %7) :: NoChange
  %9 = (exp)(%5) :: NoChange
  %10 = (Normal)(%3, %9) :: Change
  %11 = (rand)(:q2, %10) :: Change
  %12 = (+)(%8, %11) :: Change
  %13 = (Normal)(%12, 1.0) :: Change
  %14 = (rand)(:x, %13) :: Change
  return %14
```

Listing 10. Optimized IR

```
1: (%1, %2, %3, %4)
  %5 = record_cached!(Self(), :z)
  %6 = (exp)(%5)
  %7 = (Normal)(%3, %6)
  %8 = (Self())(rand, :q1, %7)
  %9 = (exp)(%5)
  %10 = (Normal)(%4, %9)
  %11 = (Self())(rand, :q2, %10)
  %12 = (+)(%8, %11)
  %13 = (Normal)(%12, 1.0)
  %14 = (Self())(rand, :x, %13)
  return %14
```

We implement a multi-stage transformation which uses a dataflow-based type system to identify where argument changes flow (before the nonstandard interpretation transformation)

Listing 4. Original IR

```
1: (%1, %2, %3)
  %4 = Normal(0.0, 1.0)
  %5 = rand(:z, %4)
  %6 = exp(%5)
  %7 = Normal(%2, %6)
  %8 = rand(:q1, %7)
  %9 = exp(%5)
  %10 = Normal(%3, %9)
  %11 = rand(:q2, %10)
  %12 = %8 + %11
  %13 = Normal(%12, 1.0)
  %14 = rand(:x, %13)
  return %14
```

Listing 5. Inferred IR

```
1: (%1 :: var"#1#2", %2 :: NoChange, %3 :: Change)
  %4 = (Normal)(0.0, 1.0) :: NoChange
  %5 = (rand)(:z, %4) :: NoChange
  %6 = (exp)(%5) :: NoChange
  %7 = (Normal)(%2, %6) :: NoChange
  %8 = (rand)(:q1, %7) :: NoChange
  %9 = (exp)(%5) :: NoChange
  %10 = (Normal)(%3, %9) :: Change
  %11 = (rand)(:q2, %10) :: Change
  %12 = (+)(%8, %11) :: Change
  %13 = (Normal)(%12, 1.0) :: Change
  %14 = (rand)(:x, %13) :: Change
  return %14
```

This is followed by a randomness reachability analysis which determines where addresses with changes propagate in the dataflow. If an IR address depends on a changed randomness address – it can't be pruned or retrieved from the cached trace and must be re-visited.