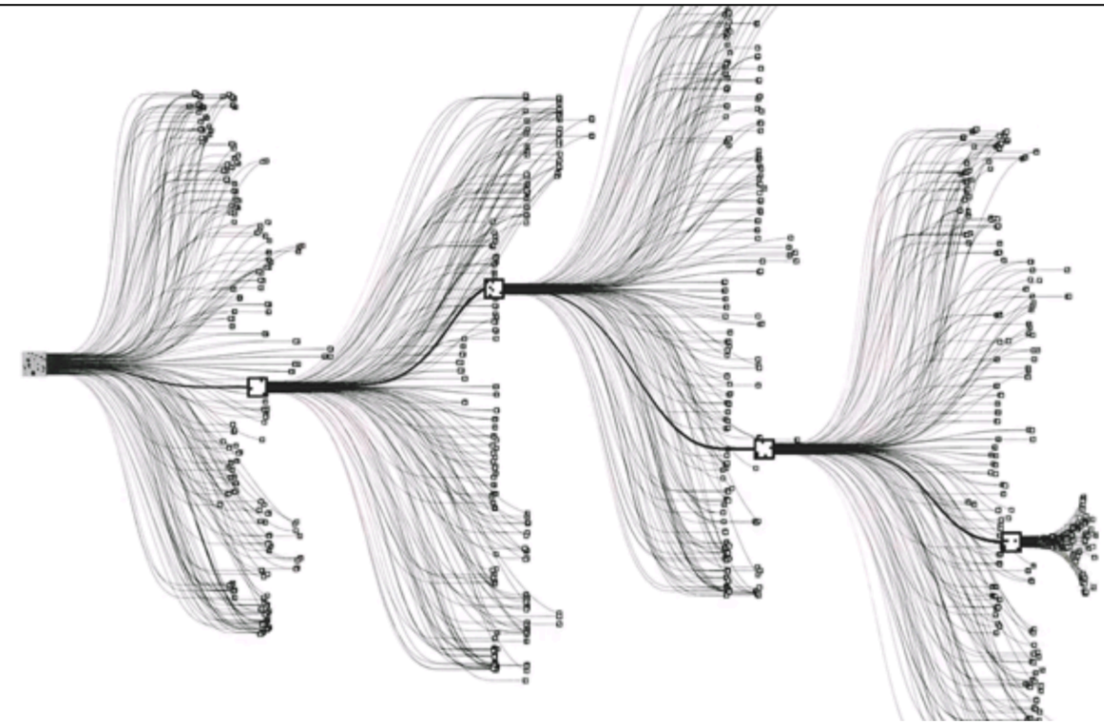
PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs

Di Wang[1], Jan Hoffmann[1], Thomas Reps[2]

[1] Carnegie Mellon University
[2] University of Wisconsin; GrammaTech, Inc.

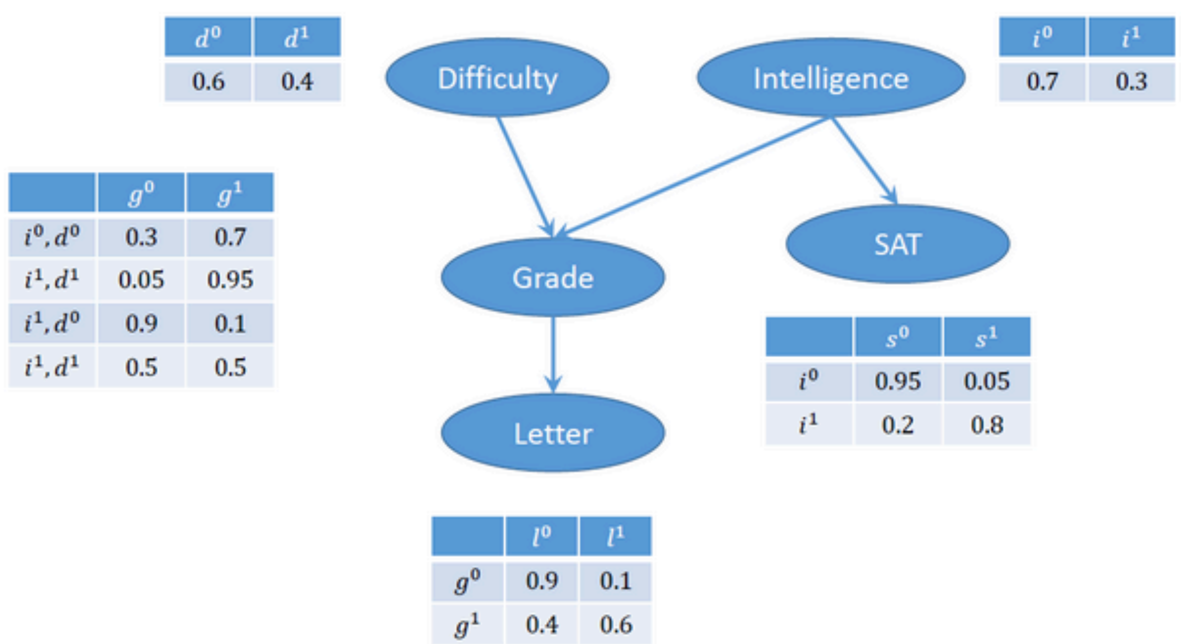
PLDI 2018

What is **probabilistic programming**?

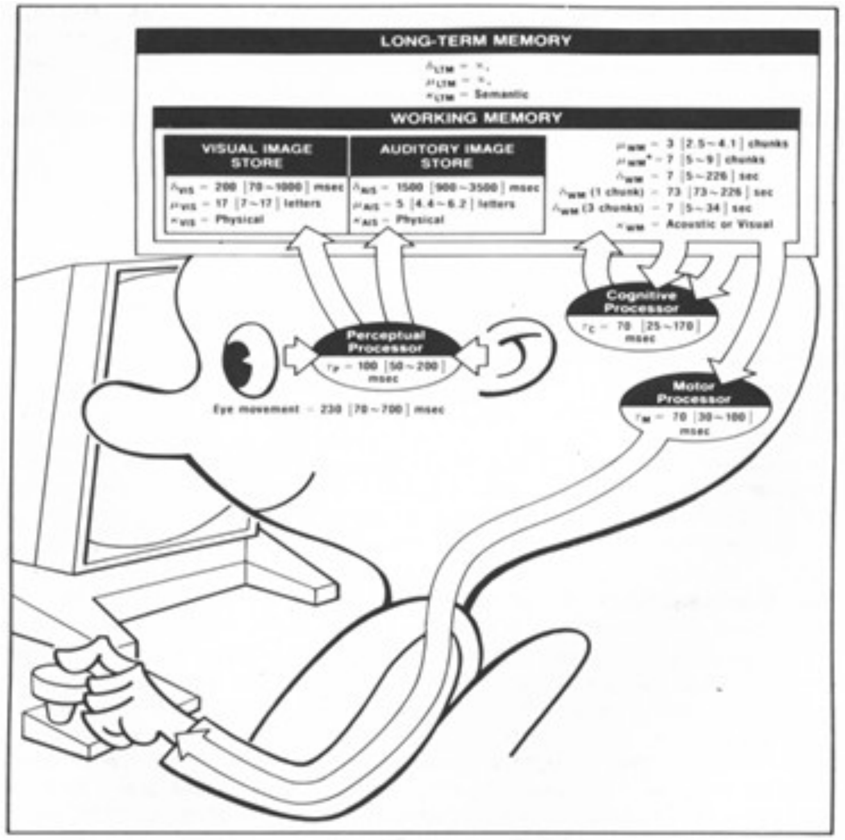


Randomized Algorithms

Cryptography Protocols



Bayesian Modeling



Cognitive Models

Probabilistic Programming

- Deterministic programs with:

Probabilistic Programming

- Deterministic programs with:
  - the ability to draw random **data** from distributions

Probabilistic Programming

- Deterministic programs with:
  - the ability to draw random **data** from distributions

Probabilistic Programming

- Deterministic programs with:
  - the ability to draw random **data** from distributions
  - the ability to condition **control-flow** at random

Probabilistic Programming

- Deterministic programs with:
  - the ability to draw random **data** from distributions
  - the ability to condition **control-flow** at random

Probabilistic Programming

- An example: an asymmetric 1d random walk

```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

Probabilistic Programming

- An example: an asymmetric 1d random walk

```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

Data Randomness

Probabilistic Programming

- An example: an asymmetric 1d random walk

```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

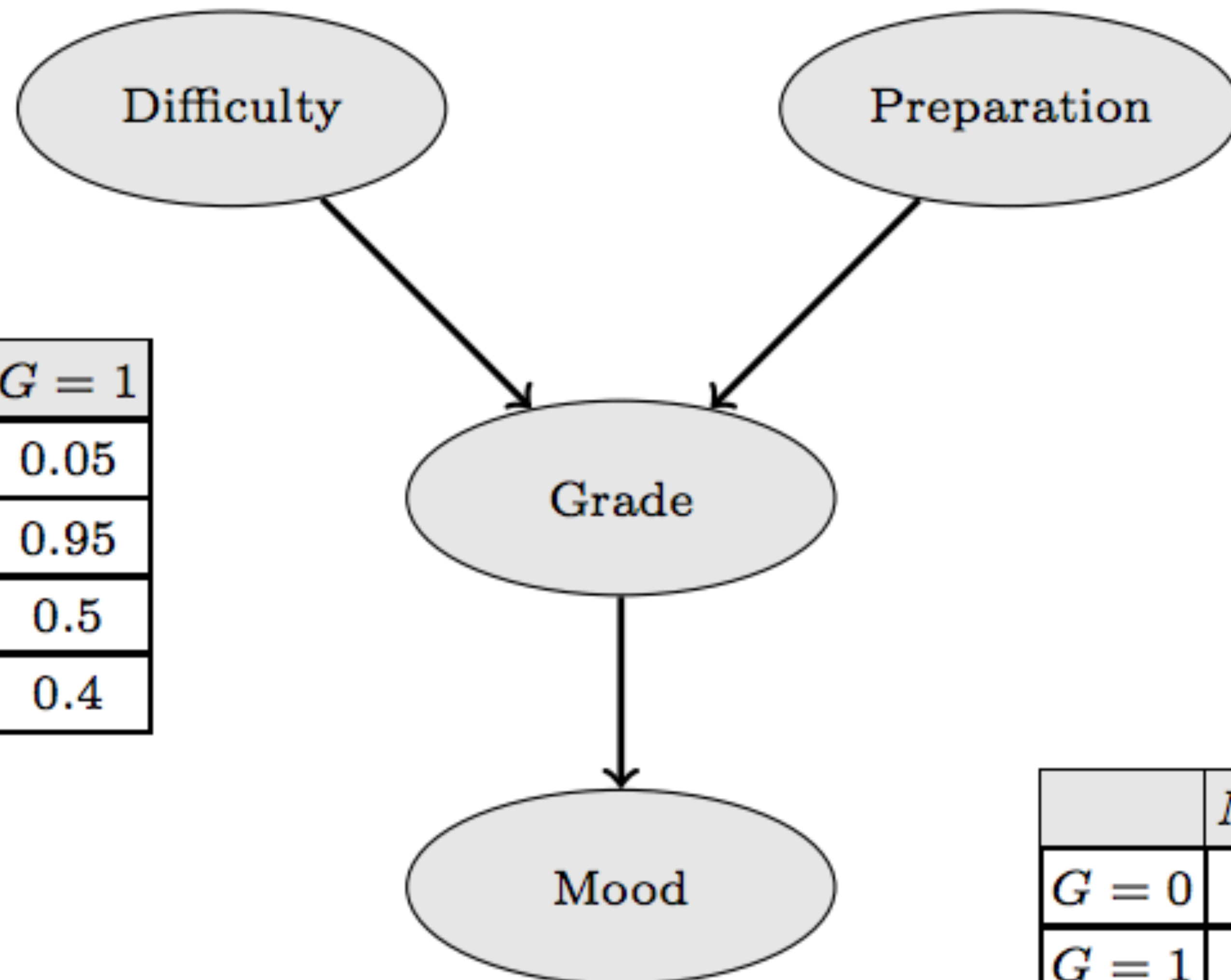
Data Randomness

Control-flow Randomness

Why is **static analysis** useful?

Bayesian Inference

$D=0$	$D=1$
0.6	0.4



$P=0$	$P=1$
0.7	0.3

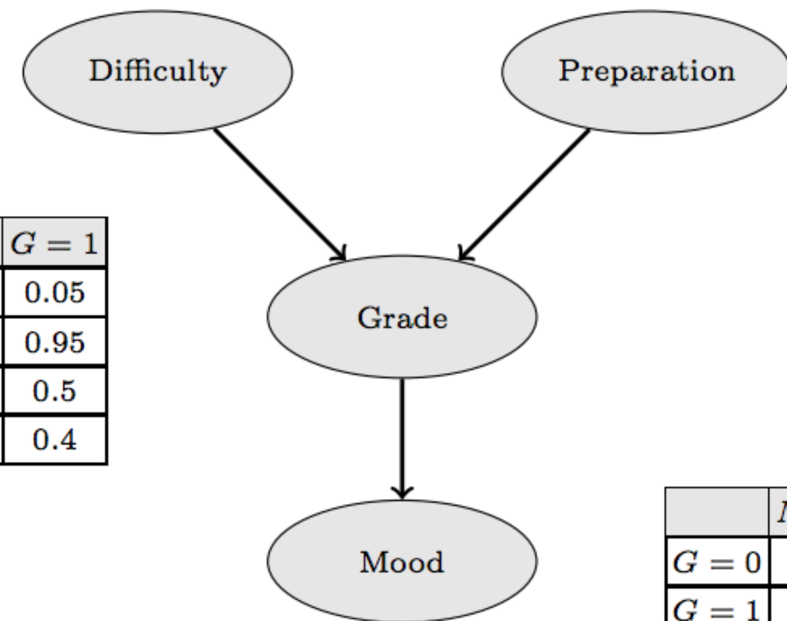
	$G=0$	$G=1$
$D=0,\ P=0$	0.95	0.05
$D=1,\ P=1$	0.05	0.95
$D=0,\ P=1$	0.5	0.5
$D=1,\ P=0$	0.6	0.4

	$M=0$	$M=1$
$G=0$	0.9	0.1
$G=1$	0.3	0.7

What is the probability that I am poorly prepared but end up with a good mood?

Bayesian Inference

$D=0$	$D=1$
0.6	0.4



$P=0$	$P=1$
0.7	0.3

	$G=0$	$G=1$
$D=0,\ P=0$	0.95	0.05
$D=1,\ P=1$	0.05	0.95
$D=0,\ P=1$	0.5	0.5
$D=1,\ P=0$	0.6	0.4

	$M=0$	$M=1$
$G=0$	0.9	0.1
$G=1$	0.3	0.7

```
repeat do
  D := 0 [0.6] D := 1;
  P := 0 [0.7] P := 1;
  if D == 0 && P == 0 then
    G := 0 [0.95] G := 1
  else if D == 1 && P == 1 then
    G := 0 [0.05] G := 1
  else if D == 0 && P == 1 then
    G := 0 [0.5] G := 1
  else
    G := 0 [0.6] G := 1
  fi;
  if G == 0 then
    M := 0 [0.9] M := 1
  else
    M := 0 [0.3] M := 1
  fi
until P == 0 && M == 1
```

Sampling-base Techniques

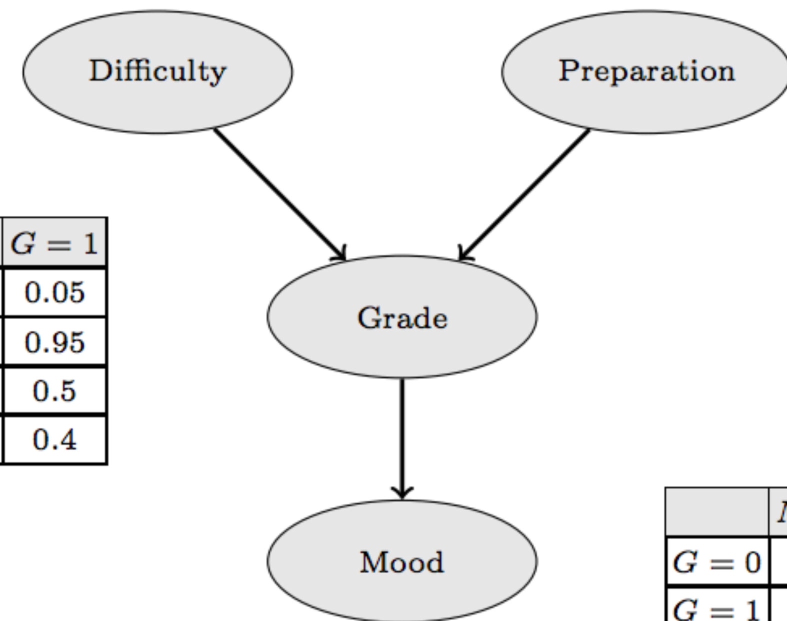
- Rejection Sampling, Markov Chain Monte Carlo, etc.
  - sample multiple times to **approximate** the distribution

Sampling-base Techniques

- Rejection Sampling, Markov Chain Monte Carlo, etc.
  - sample multiple times to **approximate** the distribution
- Two concerns:
  - not a **sound** guarantee — only **suggests** some property
  - may sample incredibly many times to get a good precision

Bayesian Inference

$D=0$	$D=1$
0.6	0.4



$P=0$	$P=1$
0.7	0.3

	$G=0$	$G=1$
$D=0,\ P=0$	0.95	0.05
$D=1,\ P=1$	0.05	0.95
$D=0,\ P=1$	0.5	0.5
$D=1,\ P=0$	0.6	0.4

	$M=0$	$M=1$
$G=0$	0.9	0.1
$G=1$	0.3	0.7

- The probability that I am poorly prepared but end up with a good mood is about 0.15
- Rejection sampling needs 1/0.15=6.7 rounds to obtain an accepting sample
- For some networks, the expectation is incredibly large ($>10^{18}$)

Static Analysis

- Formally **prove** a program satisfies some properties
- Eg:
  - Bayesian inference on general probabilistic programs
  - expected running time analysis
  - lower bound analysis for probability of post-conditions

Static Analysis

Safe & Sound

- Formally **prove** a program satisfies some properties
- Eg:
  - Bayesian inference on general probabilistic programs
  - expected running time analysis
  - lower bound analysis for probability of post-conditions

Contributions

- Developed an algebraic **framework** for **dataflow analysis** of first-order **probabilistic programs**
- Reformulated Bayesian inference & Markov decision problem in the framework
- Developed a novel expectation-invariant analysis by instantiating the framework
- Implemented an effective prototype

Example: Expectation Invariants

```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

```
x := 1; t := 0;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi;
  t := t + 1
od
```

- Want to know its expected termination time
- Analyze expectation invariants of the loop body
  - E[r’]=1, E[t’]=t+1, E[x’]=x-0.5
  - E[2x’+t’]=2x+t
- Martingales

Data & Control-flow Randomness

- Data randomness
  - `r ~ Uniform(0,2)`
- Control-flow randomness
  - **`if`** `prob(0.75)` **`then`** … **`else`** … **`fi`**

Data & Control-flow Randomness

- Data randomness
  - `r ~ Uniform(0,2)`
- Control-flow randomness
  - **`if`** `prob(0.75)` **`then`** … **`else`** … **`fi`**

- Design choice: **explicit separation**

Data & Control-flow Randomness

- One can actually simulate control-flow randomness using data randomness
  - `p ~ Uniform(0,1);` **`if`** `p < 0.75` **`then`** `…` **`else`** `…` **`fi`**

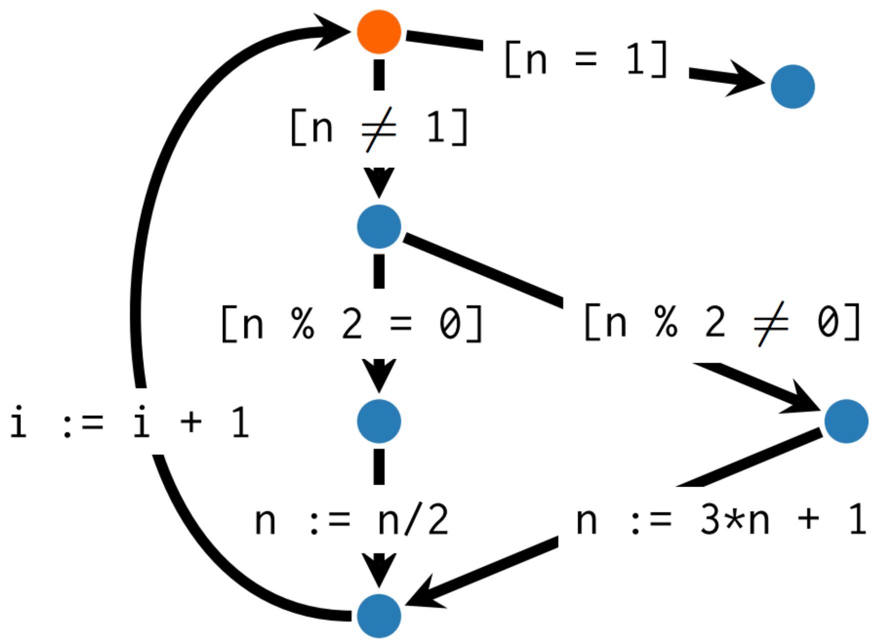
Data & Control-flow Randomness

- One can actually simulate control-flow randomness using data randomness
  - `p ~ Uniform(0,1);` **`if`** `p < 0.75` **`then`** … **`else`** … **`fi`**
- Design choice: **flexibility** for analysis designer
  - only keeping track of expectation still produces meaningful results

Control-flow Graphs

- A traditional approach to **separate** data and control-flow

```
while(n ≠ 1){
  if(n % 2 == 0)
    n := n/2;
  else
    n := 3*n+1;
  i := i+1;
}
```

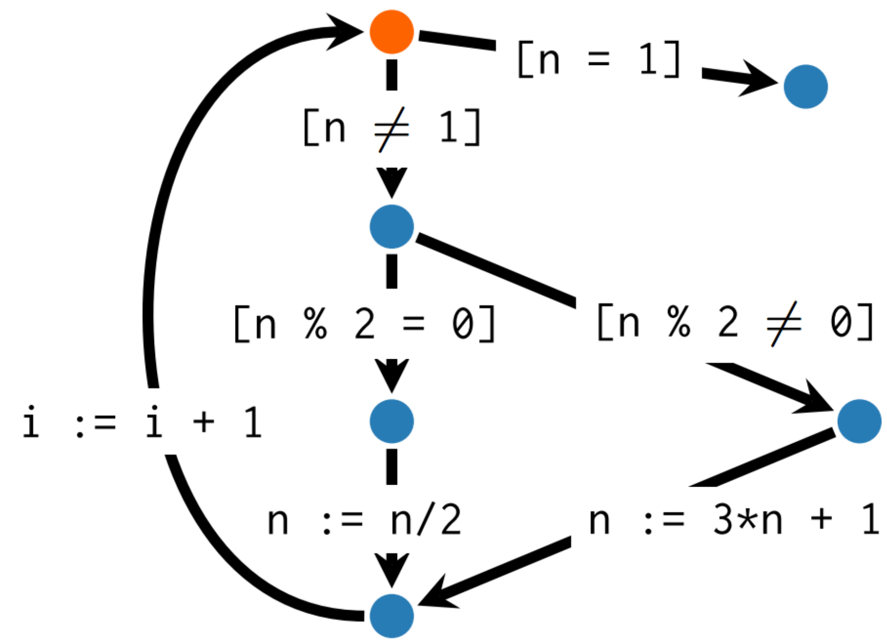


- Semantics could be defined as collections of **paths**

Control-flow Graphs

- A traditional approach to **separate** data and control-flow

```
while(n ≠ 1){
  if(n % 2 == 0)
    n := n/2;
  else
    n := 3*n+1;
  i := i+1;
}
```



- Semantics could be defined as collections of **paths**

- What about probabilistic programs?

Probabilistic Programs

Probabilistic Programs

- Paths are **<u>not</u>** independent

Probabilistic Programs

- Paths are **not** independent
- A program specifies **probability distributions** over paths

Probabilistic Programs

- Paths are **not** independent
- A program specifies **probability distributions** over paths
- Need to reason about collections of paths!

Hyper-Graphs

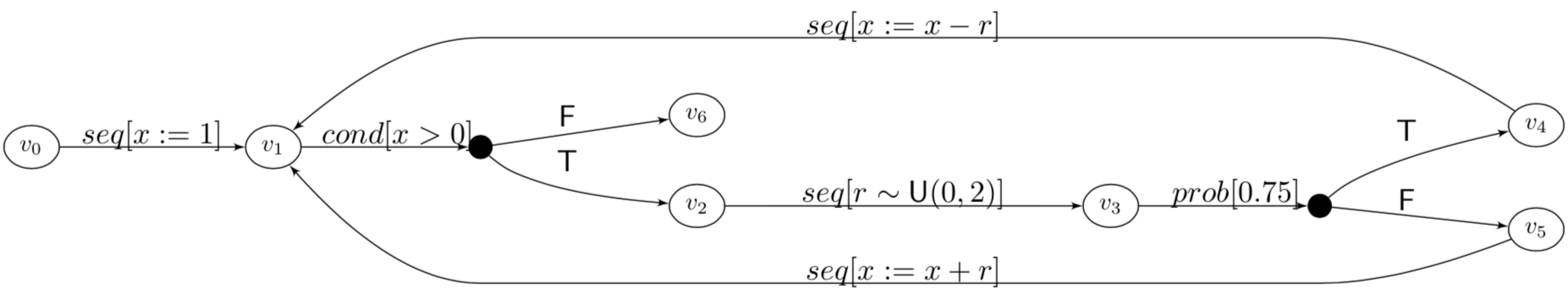
```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

- Edges have **one** source and **multiple** destinations
- cond. choices & prob. choices are modeled by hyper-edges with two destinations

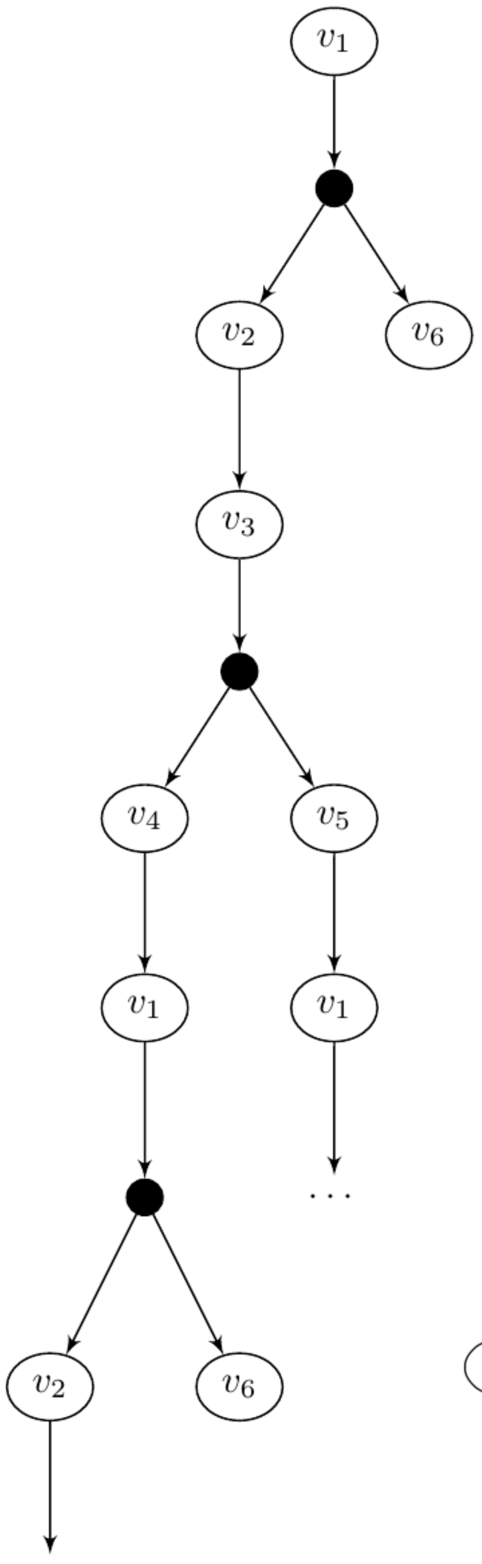
Hyper-Graphs

```
x := 1;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi
od
```

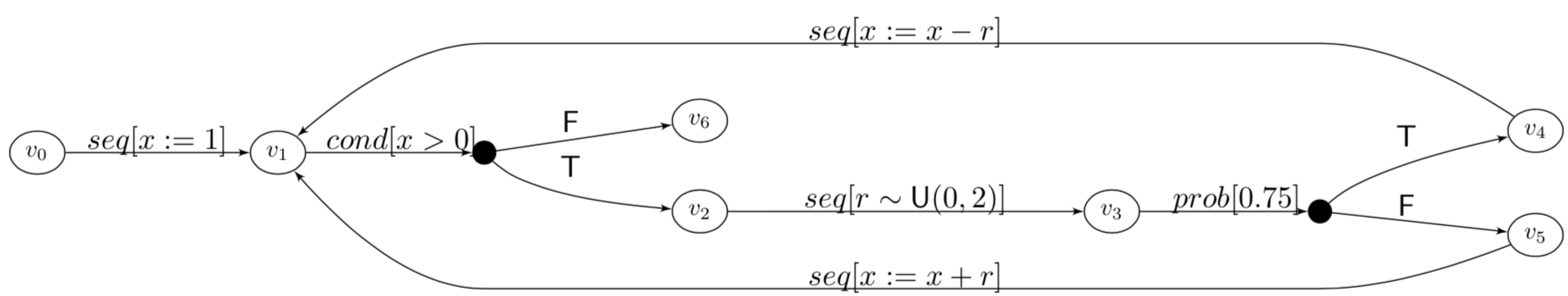
- Edges have **one** source and **multiple** destinations
- cond. choices & prob. choices are modeled by hyper-edges with two destinations



Hyper-Paths



- A hyper-path is made up of hyper-edges
- A hyper-path represents a collection of paths
- Distribution w.r.t. a hyper-path
- Nondeterminism — sets of hyper-paths



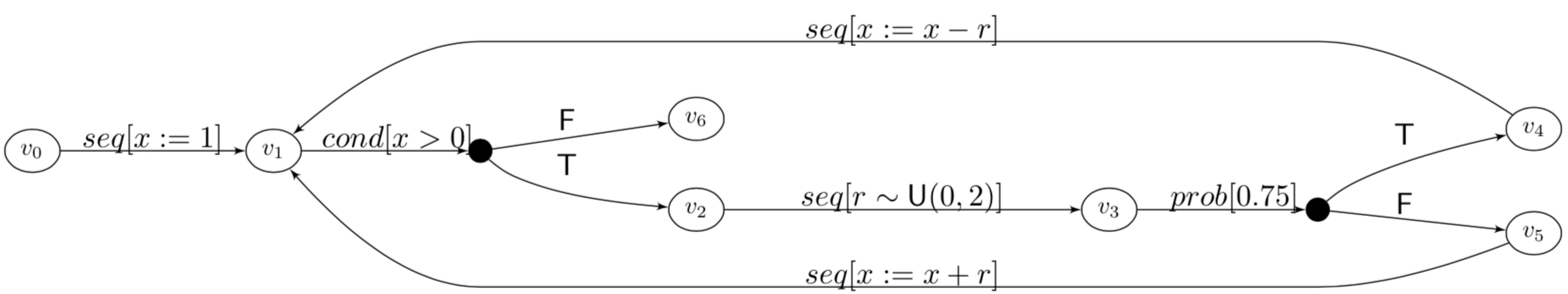
Forward Assertions

Forward Assertions

- Traditional static analyses can compute either **forward** or **backward** assertions

Forward Assertions

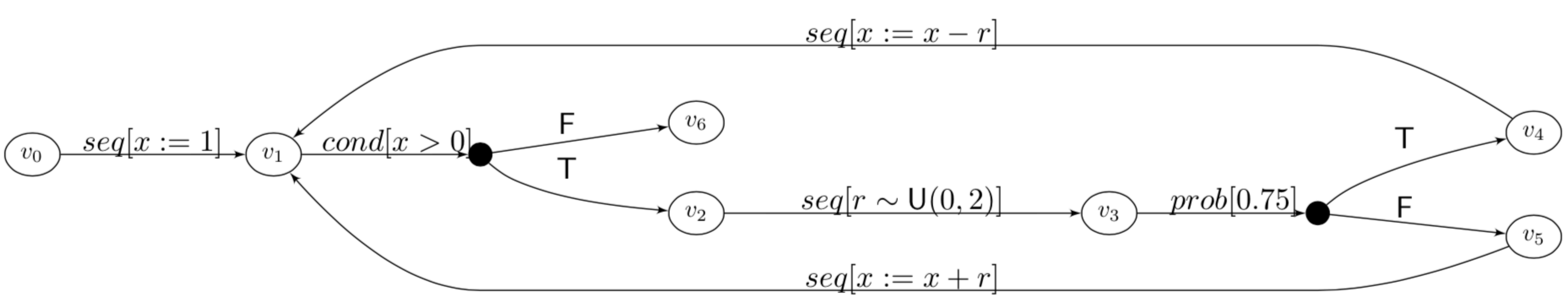
- Traditional static analyses can compute either **forward** or **backward** assertions
- Hyper-edges have **one** source and **multiple** destinations



Forward Assertions

- Traditional static analyses can compute either **forward** or **backward** assertions

- Hyper-edges have **one** source and **multiple** destinations
- Asymmetry!

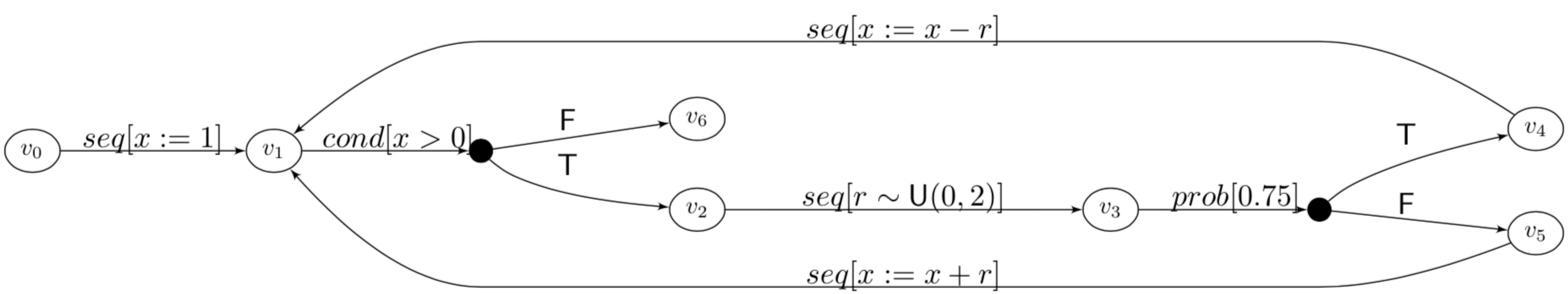


Forward Assertions

- Traditional static analyses can compute either **forward** or **backward** assertions

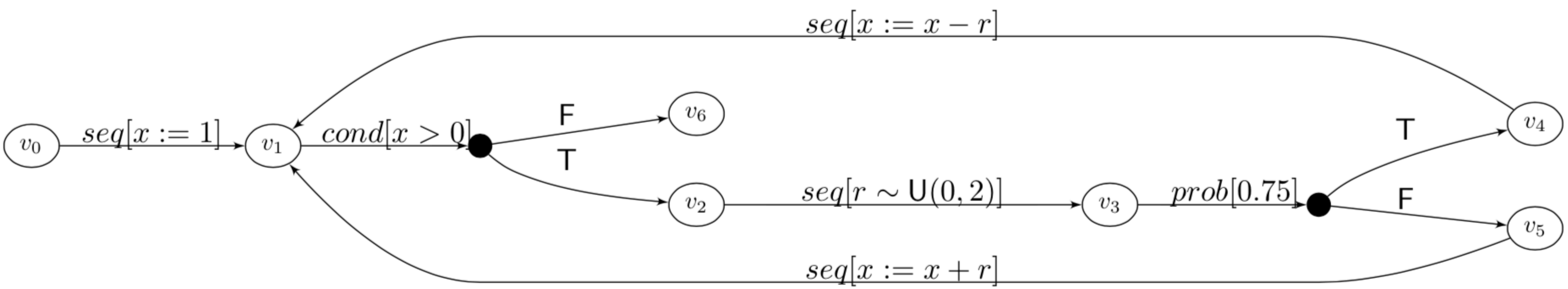
- Hyper-edges have **<u>one</u>** source and **<u>multiple</u>** destinations
- Asymmetry!

- Hyper-graphs prefer **forward** assertions
  - the semantics of a node v represents the computation that can continue from v



Forward Assertions

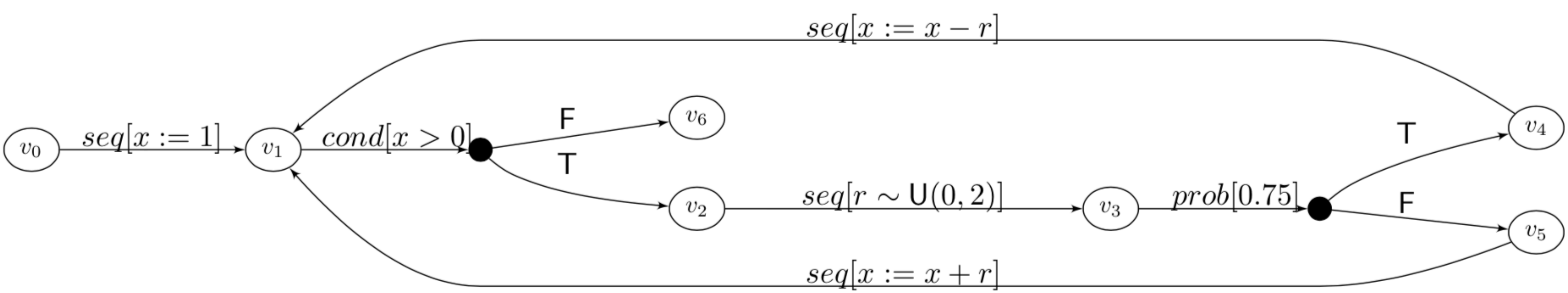
```
x := 1; t := 0;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi;
  t := t + 1
od
```



Forward Assertions

```
x := 1; t := 0;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi;
  t := t + 1
od
```

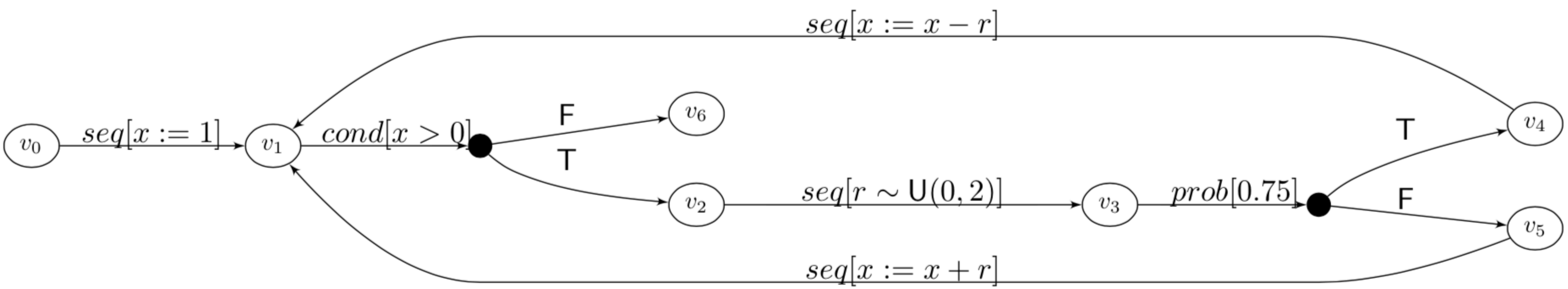
- Assertions assigned to v6:
  - E[x’]=x, E[r’]=r, E[t’]=t



Forward Assertions

```
x := 1; t := 0;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi;
  t := t + 1
od
```

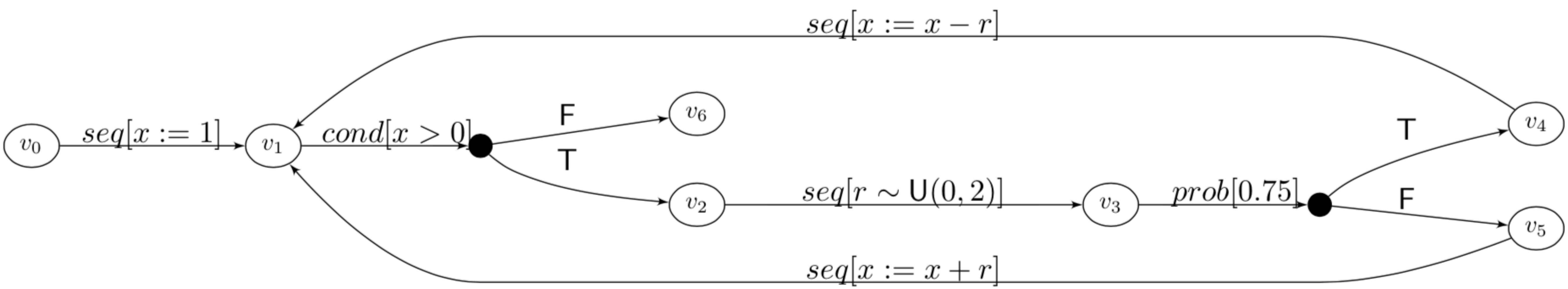
- Assertions assigned to v6:
  - E[x’]=x, E[r’]=r, E[t’]=t
- Assertions assigned to v1:
  - E[2x’+t’]=2x+t, E[x’]>=-2



Forward Assertions

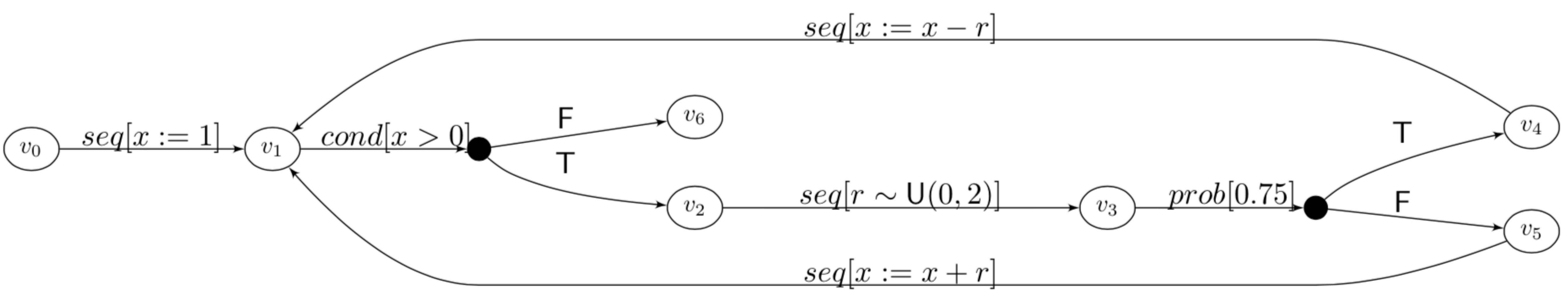
```
x := 1; t := 0;
while x > 0 do
  r ~ Uniform(0,2);
  if prob(0.75) then
    x := x - r
  else
    x := x + r
  fi;
  t := t + 1
od
```

- Assertions assigned to v6:
  - E[x’]=x, E[r’]=r, E[t’]=t
- Assertions assigned to v1:
  - E[2x’+t’]=2x+t, E[x’]>=-2
- Assertions assigned to v0:
  - E[t’]<=t+6



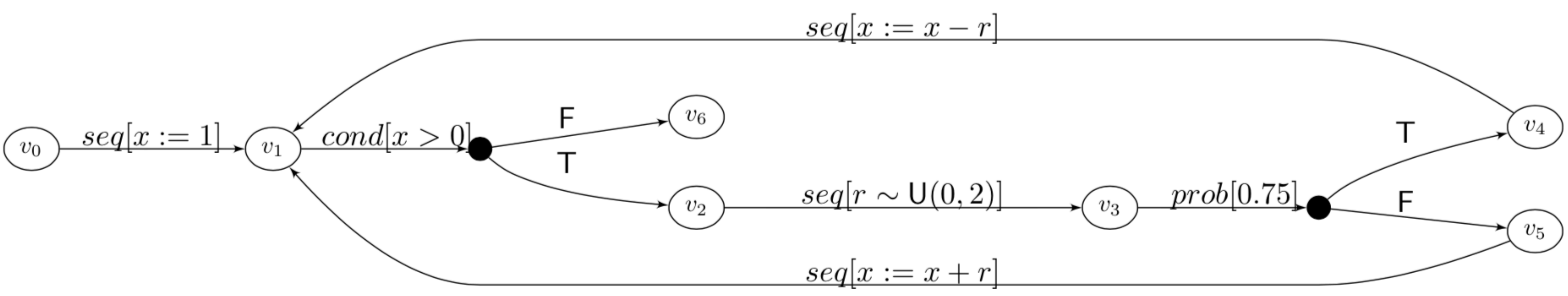
Backward Analysis

- The meaning of v4: E[2x'+t']=2(x-1)+(t+1)=2x+t-1
- The meaning of v5: E[2x'+t']=2(x+1)+(t+1)=2x+t+3



Backward Analysis

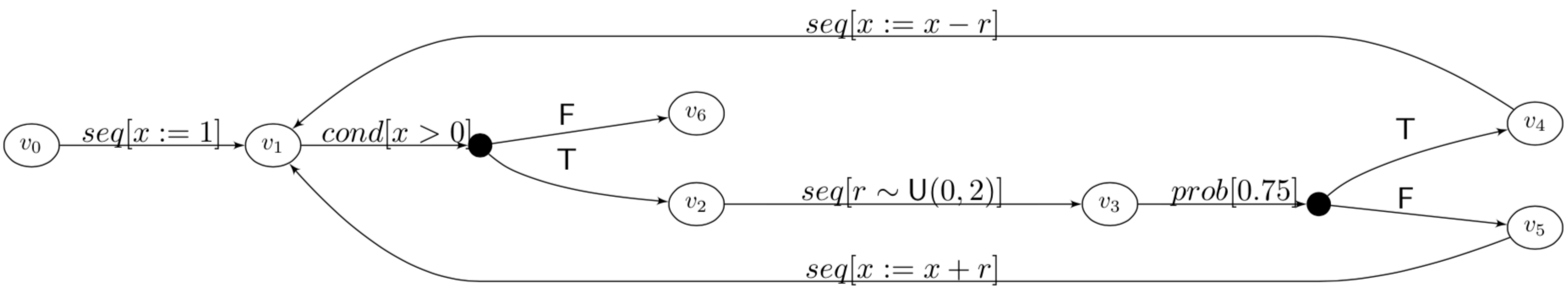
- The meaning of v4: E[2x'+t']=2(x-1)+(t+1)=2x+t-1
- The meaning of v5: E[2x'+t']=2(x+1)+(t+1)=2x+t+3
- We can compute the meaning of v3 by "**combining**" two:
    - E[2x'+t']=0.75(2x+t-1)+0.25(2x+t+3)=2x+t



Backward Analysis

- The meaning of v4: E[2x'+t']=2(x-1)+(t+1)=2x+t-1
- The meaning of v5: E[2x'+t']=2(x+1)+(t+1)=2x+t+3
- We can compute the meaning of v3 by "**combining**" two:
  - E[2x'+t']=0.75(2x+t-1)+0.25(2x+t+3)=2x+t

- A hyper-edge is a **transformer** that computes properties of source as a function of properties of destinations



Interprocedural Analysis

Interprocedural Analysis

- Two-vocabulary program properties
  - $P[x=5]=0.3$ is a **one-vocabulary** property
  - $E[2x'+t']=2x+t$ is a **two-vocabulary** expectation invariant

Interprocedural Analysis

- Two-vocabulary program properties
  - $P[x=5]=0.3$ is a **one-vocabulary** property
  - $E[2x'+t']=2x+t$ is a **two-vocabulary** expectation invariant
- One-vocabulary properties specify **states**

Interprocedural Analysis

- Two-vocabulary program properties
  - $P[x=5]=0.3$ is a **one-vocabulary** property
  - $E[2x'+t']=2x+t$ is a **two-vocabulary** expectation invariant
- One-vocabulary properties specify **states**
- Two-vocabulary properties specify **state transformers**

Interprocedural Analysis

- Two-vocabulary program properties
  - P[x=5]=0.3 is a **one-vocabulary** property
  - E[2x'+t']=2x+t is a **two-vocabulary** expectation invariant
- One-vocabulary properties specify **states**
- Two-vocabulary properties specify **state transformers**
- Two-vocabulary properties can be used as procedure summaries

An Algebraic Approach

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\perp}, \underline{1} \rangle$$

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\perp}, \underline{1} \rangle$$

Property universe

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\bot}, \underline{1} \rangle$$

Property universe

Approx. order

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\perp}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\diamond, {}_{p}\oplus, \Cup, \underline{\bot}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

Cond.-choice

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\perp}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

Cond.-choice

Prob.-choice

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
    - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\perp}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

Cond.-choice

Prob.-choice

Nondet.-choice

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\Diamond, {}_{p}\oplus, \Cup, \underline{\bot}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

Cond.-choice

Prob.-choice

Nondet.-choice

Bottom

An Algebraic Approach

- Any static analysis method performs reasoning in some space of program **properties** and property **operations**; such property operations should obey **algebraic laws**
  - `skip` should be interpreted as the identity element

$$\langle M, \sqsubseteq, \otimes, {}_{\varphi}\diamond, {}_{p}\oplus, \Cup, \underline{\bot}, \underline{1} \rangle$$

Property universe

Approx. order

Seq.

Cond.-choice

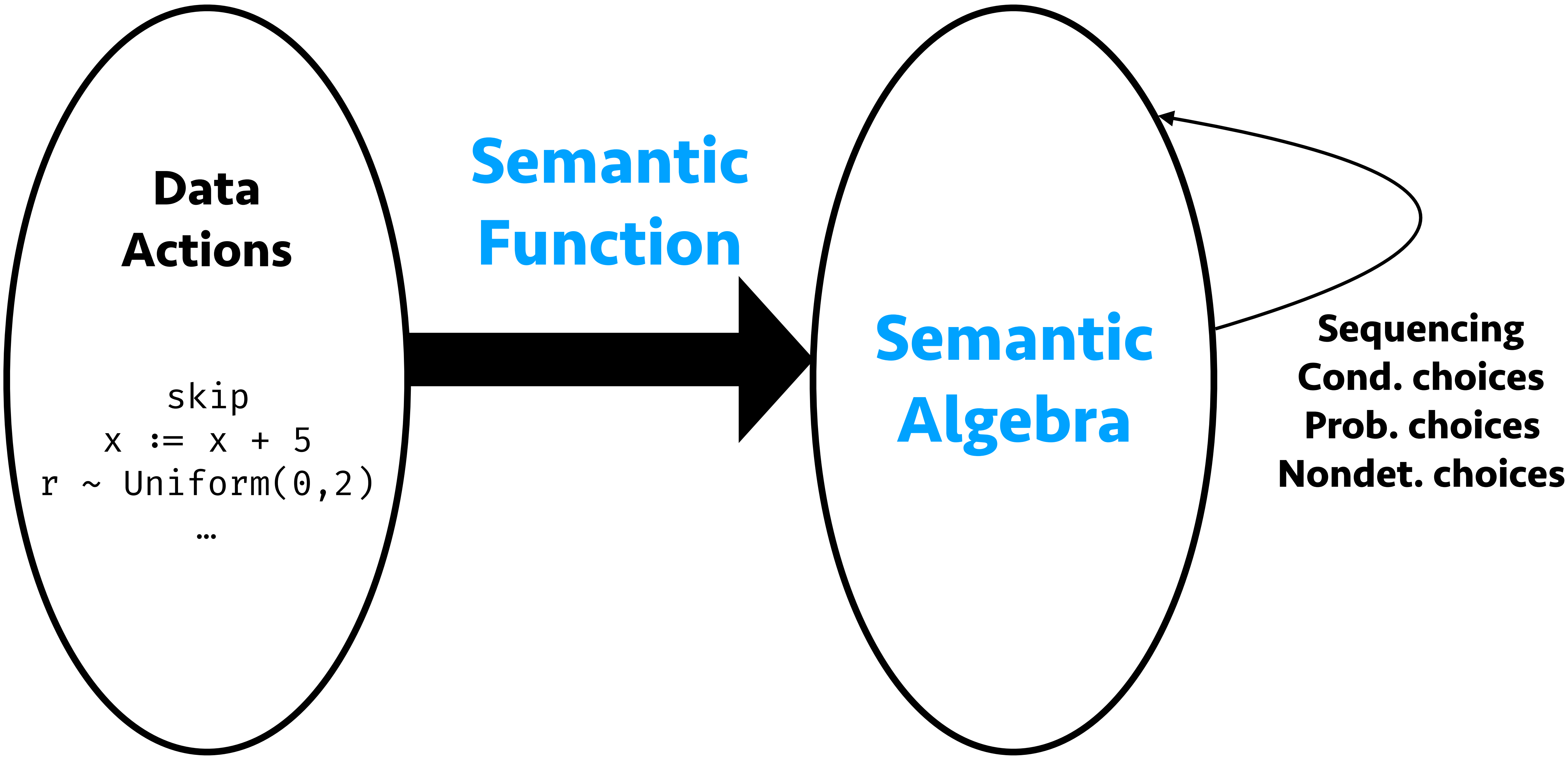
Prob.-choice

Nondet.-choice

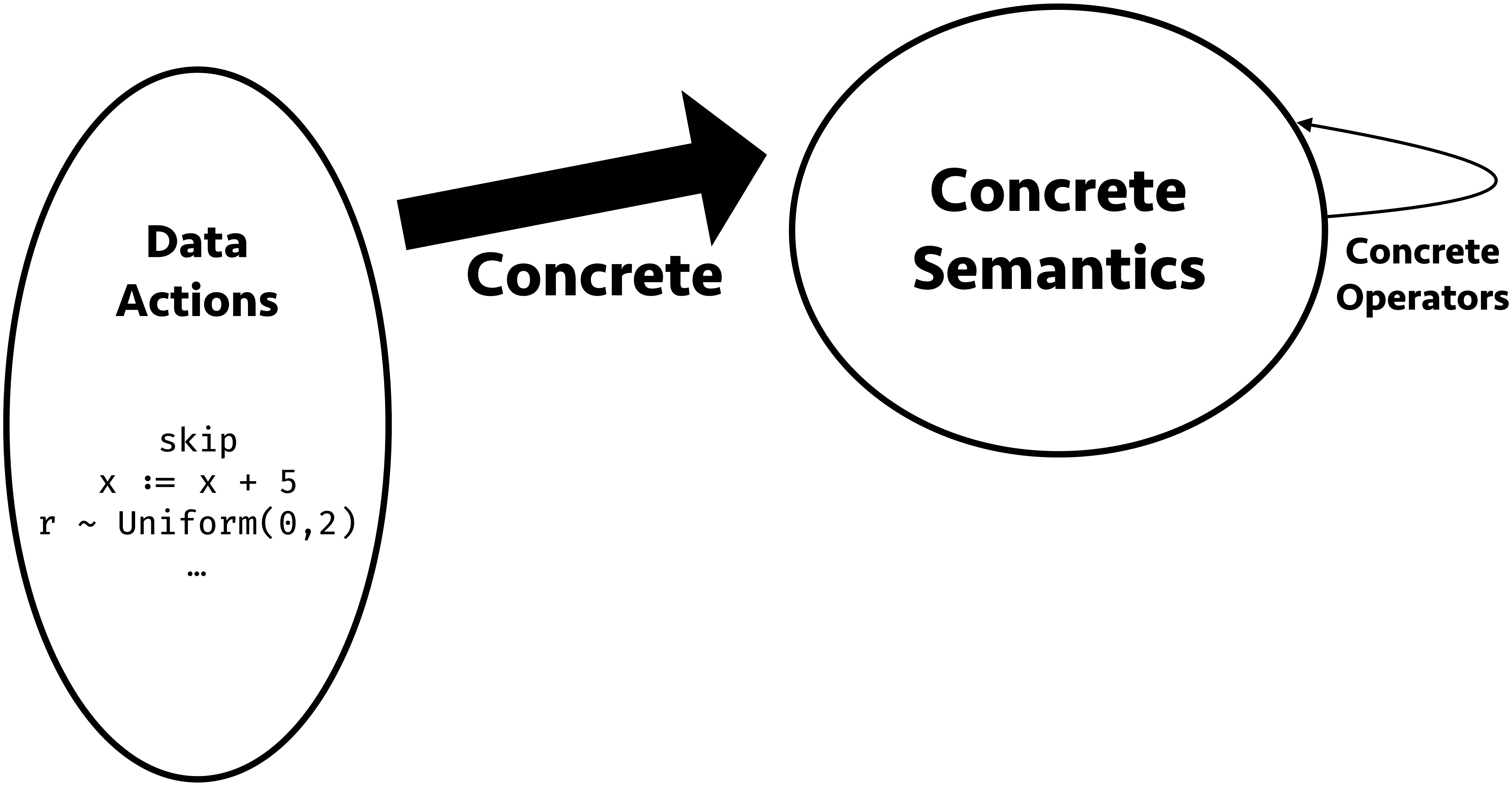
Bottom

Identity

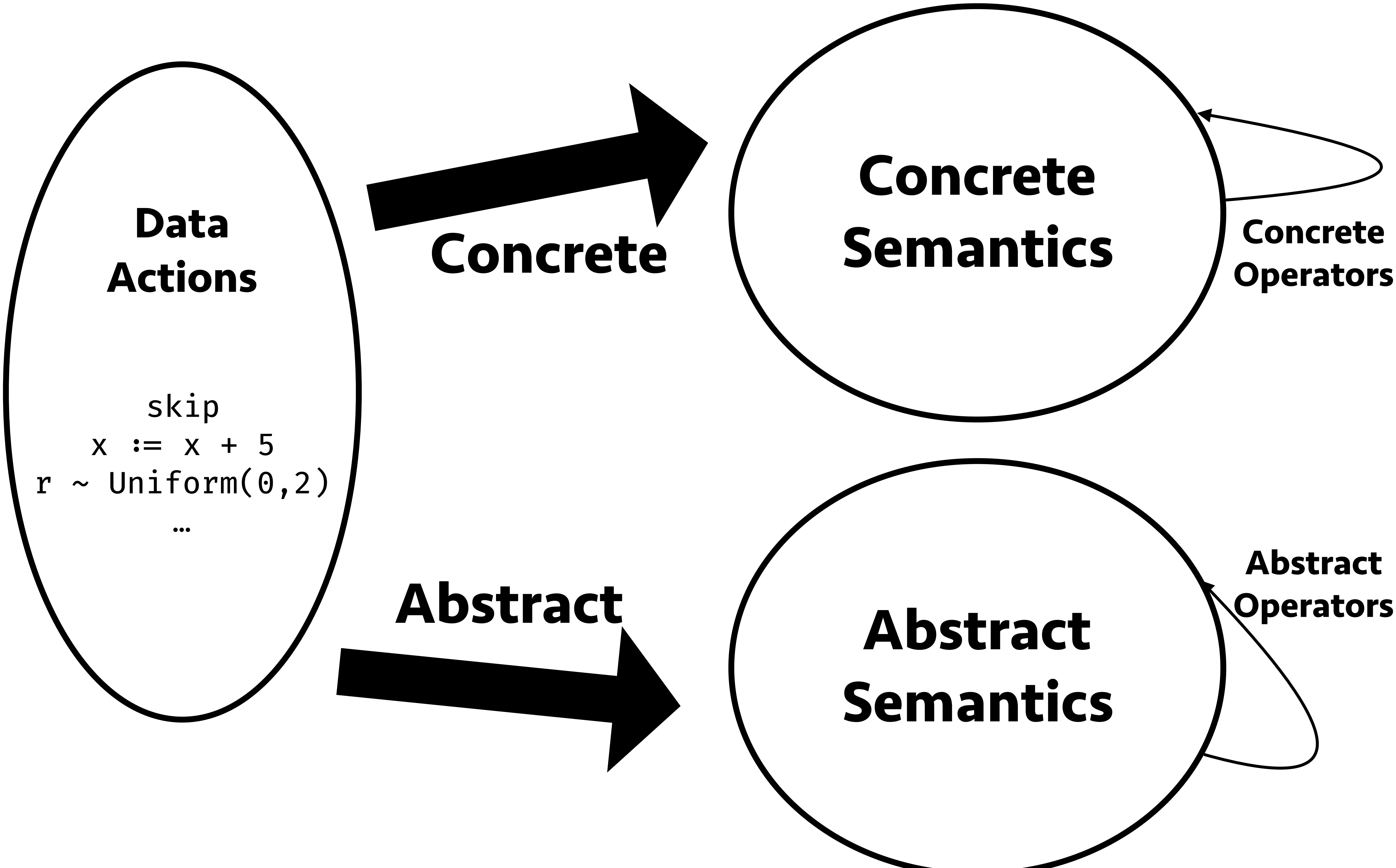
An Algebraic Approach



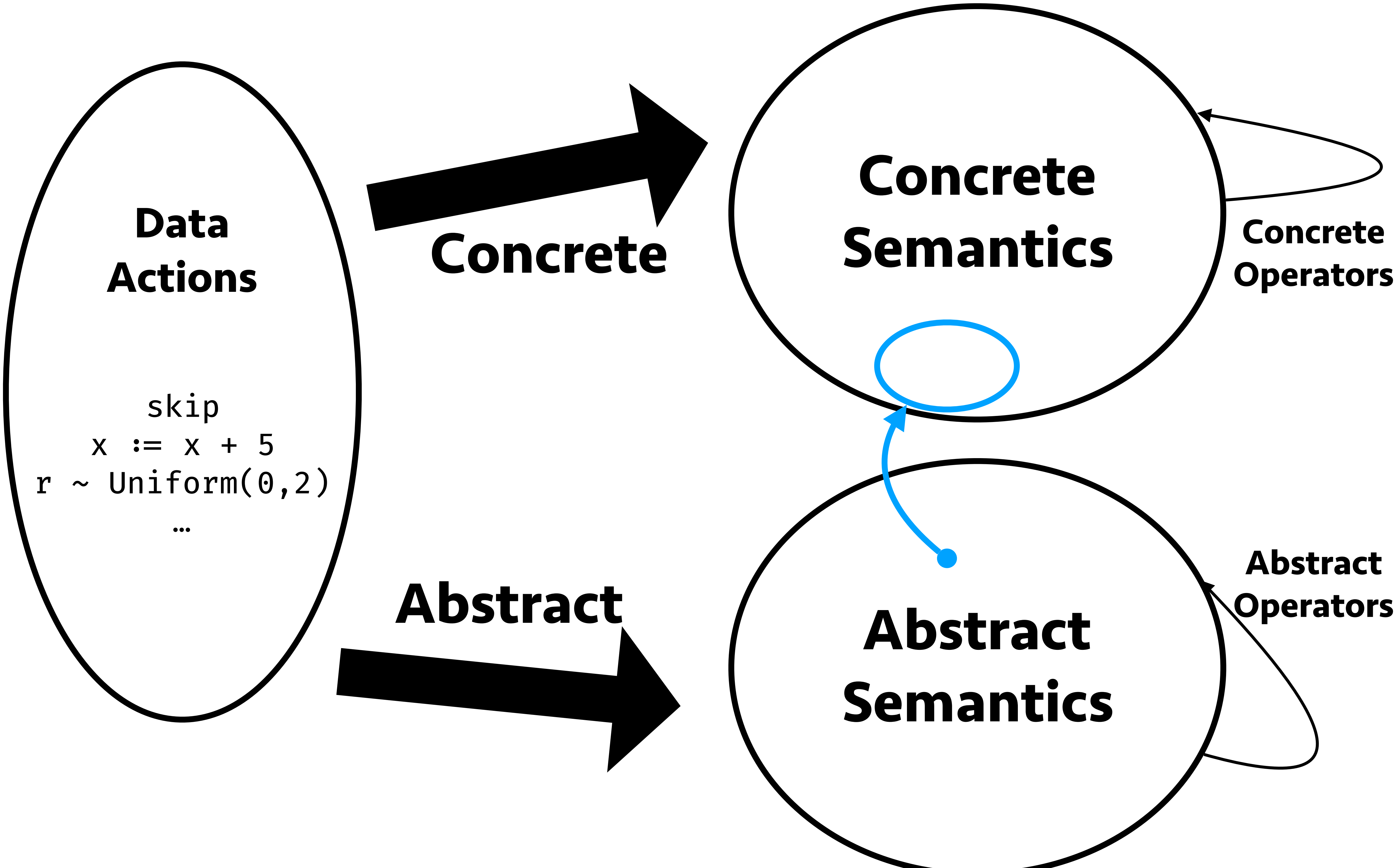
Approximation



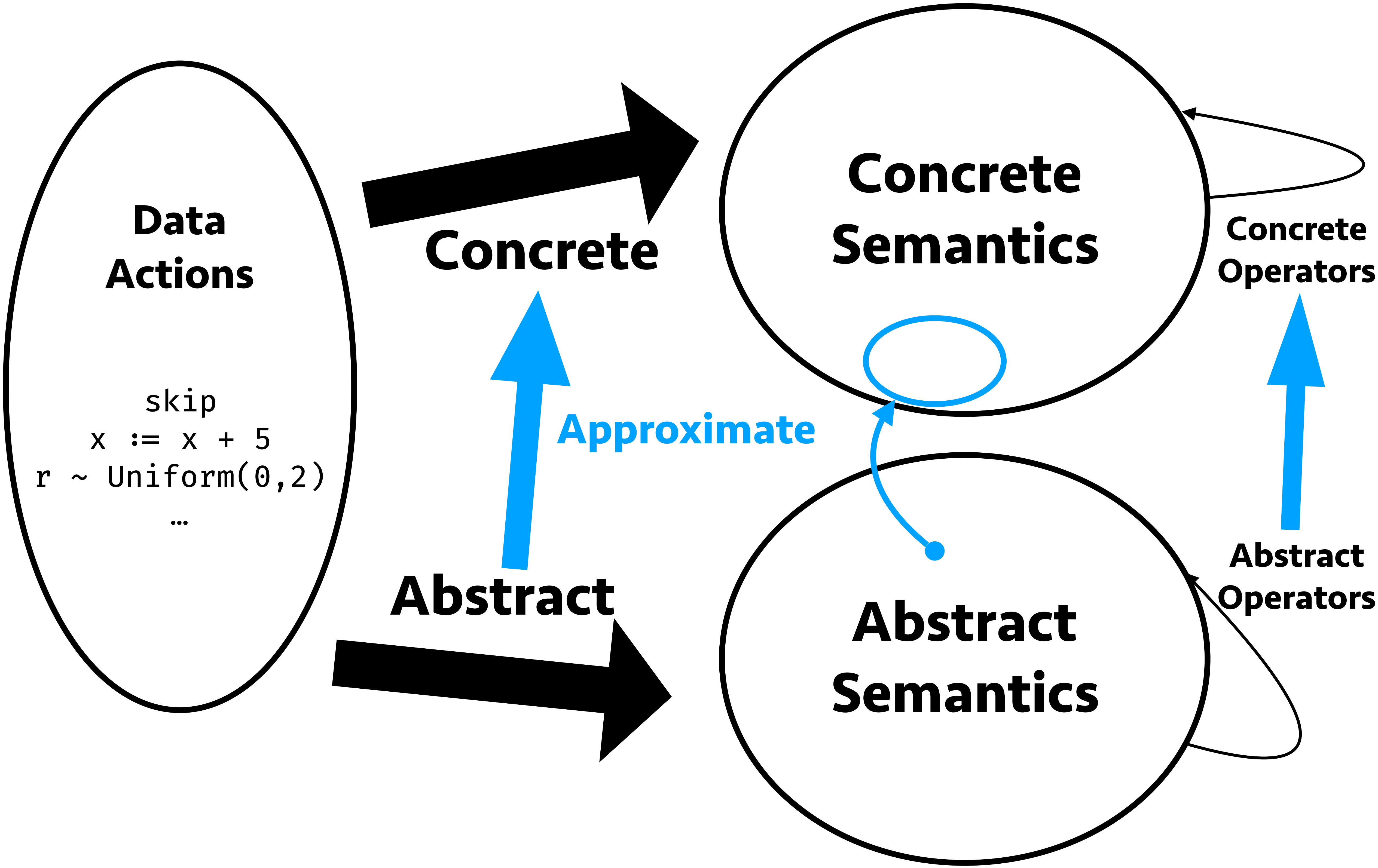
Approximation



Approximation



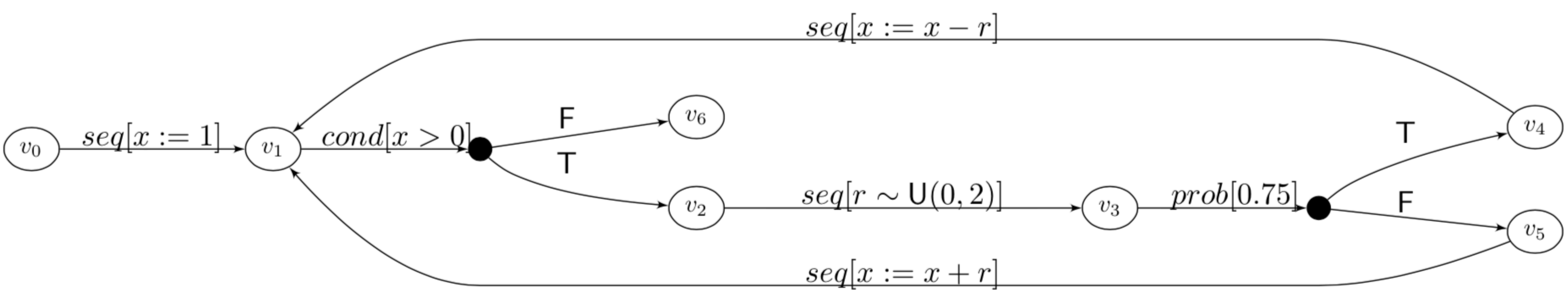
Approximation



General Analysis Algorithm

- Solve an equation system extracted from the control-flow hyper-graph
- Chaotic-iteration strategy
- Widening
- The **framework** furnishes the analysis implementation

```
S[v0] ⩾ seq[x:=1](S[v1])
S[v1] ⩾ cond[x>0](S[v2],S[v6])
S[v2] ⩾ seq[r~U(0,2)](S[v3])
S[v3] ⩾ prob[0.75](S[v4],S[v5])
S[v4] ⩾ seq[x:=x-r](S[v1])
S[v5] ⩾ seq[x:=x+r](S[v1])
S[v6] ⩾ 1
```



Technical Summary

- A blending of ideas from prior work on
  - static analysis of single-procedure probabilistic programs
  - interprocedural dataflow analysis of standard programs
- Especially
  - the separation of data & control-flow randomness
  - backward analysis on control-flow hyper-graphs
  - two-vocabulary program properties
  - an algebraic approach

Instantiations

- **Bayesian inference**: compute the posterior distribution
  - abstract programs as distribution transformers matrices
- **Markov decision problem**: compute the optimal expected reward
  - abstract programs as real numbers (reward gain)
- **Linear expectation-invariant analysis**
  - abstract programs as pairs of polyhedra (relational domain)

Future Work

- Design more efficient analysis algorithms to exploit all algebraic laws
- Find useful coarser abstractions for Bayesian inference by analogy with the techniques for predicate abstraction
- Use the framework to design new analysis for expected resource analysis and side-channel attack analysis