Nested Queries in Nondeterministic and Probablistic Programming Languages

Jeffrey Mark Siskind, qobi@purdue.edu

PURDUE UNIVERSITY

NEPLS, Tuesday 10 November 2015

Joint work with Frank Wood and Hongseok Yang

Prolog

```
?- member(X, [a, b, c]).
X = a
```

Colmerauer, Alain, Henry Kanoui, Robert Pasero, and Philippe Roussel, 'Un système de communication en français, rapport préliminaire de fin de contrat IRIA,' Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Oct. 1972.

Prolog

```
?- member(X, [a, b, c]).
X = a ;
X = b
```

Colmerauer, Alain, Henry Kanoui, Robert Pasero, and Philippe Roussel, 'Un système de communication en français, rapport préliminaire de fin de contrat IRIA,' Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Oct. 1972.

Prolog

```
?- member(X, [a, b, c]).
X = a ;
X = b ;
X = c.
?-
```

Colmerauer, Alain, Henry Kanoui, Robert Pasero, and Philippe Roussel, 'Un système de communication en français, rapport préliminaire de fin de contrat IRIA,' Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Oct. 1972.

Prolog

```
?- member(X, [a, b, c]).
X = a ;
X = b ;
X = c.
?- bagof(X, member(X, [a, b, c]), Xs).
Xs = [a, b, c].
?-
```

Colmerauer, Alain, Henry Kanoui, Robert Pasero, and Philippe Roussel, 'Un système de communication en français, rapport préliminaire de fin de contrat IRIA,' Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Oct. 1972.

Warren, David H. D., 'Higher-order extensions to Prolog: are they needed?' *Machine Intelligence*, 10:441-454, 1982.

Nondeterministic Lisp/Schemer/Screamer

```
> (define (choose l)
    (if (null? l) (fail) (amb (first l) (choose (rest l)))))
> (choose '(a b c))
a
>
```

Chapman, David, 'Planning For Conjunctive Goals,' MIT AI Lab TR-802, 1985.

Nondeterministic Lisp/Schemer/Screamer

```
> (define (choose l)
    (if (null? l) (fail) (amb (first l) (choose (rest l)))))
> (choose '(a b c))
a
> (fail)
b
>
```

Chapman, David, 'Planning For Conjunctive Goals,' MIT AI Lab TR-802, 1985.

Nondeterministic Lisp/Schemer/Screamer

```
> (define (choose l)
    (if (null? l) (fail) (amb (first l) (choose (rest l)))))
> (choose '(a b c))
a
> (fail)
b
> (fail)
c
>
```

Chapman, David, 'Planning For Conjunctive Goals,' MIT AI Lab TR-802, 1985.

Nondeterministic Lisp/Schemer/Screamer

```
> (define (choose l)
    (if (null? l) (fail) (amb (first l) (choose (rest l)))))
> (choose '(a b c))
a
> (fail)
b
> (fail)
c
> (fail)
>>Error: Top-level fail
>
```

Chapman, David, 'Planning For Conjunctive Goals,' MIT AI Lab TR-802, 1985.

Nondeterministic Lisp/Schemer/Screamer

```
> (define (choose l)
    (if (null? l) (fail) (amb (first l) (choose (rest l)))))
> (choose '(a b c))
a
> (fail)
b
> (fail)
c
> (fail)
>>Error: Top-level fail
> (set (choose '(a b c)))
(a b c)
>
```

Chapman, David, 'Planning For Conjunctive Goals,' MIT AI Lab TR-802, 1985.

Siskind, Jeffrey Mark, 'Screamning Yellow Zonkers,' MIT, 1991.

Stochastic Lisp/Probabilistic Scheme/Church

```
> (sample '((a . 0.5) (b . 0.25) (c . 0.25)))
a
Probability is: 0.5
>
```

Koller Daphne, McAllester David, and Pfeffer Avi, 'Effective Bayseian Inference for Stochastic Programs,' *AAAI*, pp. 740–747, 1997.

Stochastic Lisp/Probabilistic Scheme/Church

```
> (sample '((a . 0.5) (b . 0.25) (c . 0.25)))
a
Probability is: 0.5
> (fail)
b
Probability is: 0.25
>
```

Koller Daphne, McAllester David, and Pfeffer Avi, 'Effective Bayseian Inference for Stochastic Programs,' *AAAI*, pp. 740–747, 1997.

Stochastic Lisp/Probabilistic Scheme/Church

```
> (sample '((a . 0.5) (b . 0.25) (c . 0.25)))
a
Probability is: 0.5
> (fail)
b
Probability is: 0.25
> (fail)
c
Probability is: 0.25
>
```

Koller Daphne, McAllester David, and Pfeffer Avi, 'Effective Bayseian Inference for Stochastic Programs,' *AAAI*, pp. 740–747, 1997.

Stochastic Lisp/Probabilistic Scheme/Church

```
> (sample '((a . 0.5) (b . 0.25) (c . 0.25)))
a
Probability is: 0.5
> (fail)
b
Probability is: 0.25
> (fail)
c
Probability is: 0.25
> (fail)
>>Error: Top-level fail
>
```

Koller Daphne, McAllester David, and Pfeffer Avi, 'Effective Bayseian Inference for Stochastic Programs,' *AAAI*, pp. 740–747, 1997.

Stochastic Lisp/Probabilistic Scheme/Church

```
> (sample '((a . 0.5) (b . 0.25) (c . 0.25)))
a
Probability is: 0.5
> (fail)
b
Probability is: 0.25
> (fail)
c
Probability is: 0.25
> (fail)
>>Error: Top-level fail
> (distribution (sample '((a . 0.5) (b . 0.25) (c . 0.25))))
((a . 0.5) (b . 0.25) (c . 0.25))
>
```

Koller Daphne, McAllester David, and Pfeffer Avi, 'Effective Bayseian Inference for Stochastic Programs,' *AAAI*, pp. 740–747, 1997.

Radul, Alexey, 'Report on the Probabilistic Language Scheme,' *Symposium on Dynamic Languages*, pp. 2–10, 2007.

Dichotomy

Dichotomy

`choose` turns a set into a nondeterministic process

Dichotomy

`choose` turns a set into a nondeterministic process

`set` turns a nondeterministic process into a set

Dichotomy

`choose` turns a set into a nondeterministic process

`set` turns a nondeterministic process into a set

`sample` turns a distribution into a stochastic process

Dichotomy

`choose` turns a set into a nondeterministic process

`set` turns a nondeterministic process into a set

`sample` turns a distribution into a stochastic process

`distribution` turns a stochastic process into a distribution

Dichotomy

`choose` turns a set into a nondeterministic process

`set` turns a nondeterministic process into a set

`sample` turns a distribution into a stochastic process

`distribution` turns a stochastic process into a distribution

`(choose (set` e`))`	$=$	e	
`(set (choose` e`))`	$=$	e	when e is deterministic
`(sample (distribution` e`))`	$=$	e	
`(distribution (sample` e`))`	$=$	e	when e is deterministic

Implementation

```
(define (nondeterministic-boolean) (error #f "Top-level nondeterministic-boolean"))

(define (nondeterministic-fail) (error #f "Top-level nondeterministic-fail"))

(define (choose x)
 (let loop ((x x)      )
  (cond ((or (null?  x)           ) (nondeterministic-fail))
        ((nondeterministic-boolean                      ) (first x))
        (else (loop (rest x)                      )))))
```


Wand, Mitchell, 'Continuation-based multiprocessing,' *LFP*, pp. 19–28, 1980.

Carlsson, M. and Kahn, Kenneth M., 'LM-Prolog User Manual,' UPMAIL Technical Report No. 24, Uppsala University, Sweden, 1983.

Haynes, C.T., Friedman, D.P., and Wand, M., 'Continuations and Coroutines,' Indiana University Computer Science TR 158, 1984.

Stickel, Mark E., 'A Prolog Technology Theorem Prover,' *New Generation Computing*, 2:371–383, 1984.

Felleisen, Matthias, 'Transliterating Prolog into Scheme,' Indiana University Computer Science TR 85-182, 1985.

Haynes, Christopher T., 'Logic Continuations,' *ICLP*, 1986.

Implementation

```
(define (probabilistic-boolean alpha) (error #f "Top-level probabilistic-boolean"))

(define (probabilistic-fail) (error #f "Top-level probabilistic-fail"))

(define (sample x)
 (let loop ((x x) (p 1))
  (cond ((or (null?  x) (zero?  p)) (probabilistic-fail))
        ((probabilistic-boolean (/ (cdr (first x)) p)) (car (first x)))
        (else (loop (rest x) (- p (cdr (first x))))))))
```


Wand, Mitchell, 'Continuation-based multiprocessing,' *LFP*, pp. 19–28, 1980.

Carlsson, M. and Kahn, Kenneth M., 'LM-Prolog User Manual,' UPMAIL Technical Report No. 24, Uppsala University, Sweden, 1983.

Haynes, C.T., Friedman, D.P., and Wand, M., 'Continuations and Coroutines,' Indiana University Computer Science TR 158, 1984.

Stickel, Mark E., 'A Prolog Technology Theorem Prover,' *New Generation Computing*, 2:371–383, 1984.

Felleisen, Matthias, 'Transliterating Prolog into Scheme,' Indiana University Computer Science TR 85-182, 1985.

Haynes, Christopher T., 'Logic Continuations,' *ICLP*, 1986.

Implementation

```
(define-syntax set
 (syntax-rules ()
  ((set e)
   (call-with-current-continuation
    (lambda (c)
     (let ((values '())
           (saved-nondeterministic-boolean nondeterministic-boolean)
           (saved-nondeterministic-fail nondeterministic-fail)
                )
      (set! nondeterministic-boolean
            (lambda ()

             (call-with-current-continuation
              (lambda (c)
               (let ((saved-nondeterministic-fail nondeterministic-fail)
                                  )
                (set! nondeterministic-fail
                      (lambda ()
                       (set! nondeterministic-fail saved-nondeterministic-fail)

                       (c #f)))

                #t)))))
      (set! nondeterministic-fail
            (lambda ()
             (set! nondeterministic-boolean saved-nondeterministic-boolean)
             (set! nondeterministic-fail saved-nondeterministic-fail)
             (c (reverse values))))
      (set! values (cons e values))
      (nondeterministic-fail)))))))
```

Implementation

```
(define-syntax distribution
 (syntax-rules ()
  ((distribution e)
   (call-with-current-continuation
    (lambda (c)
     (let ((values '())
           (saved-probabilistic-boolean probabilistic-boolean)
           (saved-probabilistic-fail probabilistic-fail)
           (p 1))
      (set! probabilistic-boolean
            (lambda (alpha)
             (unless (<= 0 alpha 1) (error #f "Alpha not probability"))
             (call-with-current-continuation
              (lambda (c)
               (let ((saved-probabilistic-fail probabilistic-fail)
                     (saved-p p))
                (set! probabilistic-fail
                      (lambda ()
                       (set! probabilistic-fail saved-probabilistic-fail)
                       (set! p (* (- 1 alpha) saved-p))
                       (c #f)))
                (set! p (* alpha p))
                #t)))))
      (set! probabilistic-fail
            (lambda ()
             (set! probabilistic-boolean saved-probabilistic-boolean)
             (set! probabilistic-fail saved-probabilistic-fail)
             (c (reverse values))))
      (set! values (cons (cons e p) values))
      (probabilistic-fail)))))))
```

Utility of a Query or Aggregation Construct

Utility of a Query or Aggregation Construct

- satisfiability

```
(not (null? (set e)))
```

Utility of a Query or Aggregation Construct

- satisfiability

```
(not (null? (set e)))
```

- model counting

```
(length (set e))
```

Utility of a Query or Aggregation Construct

- satisfiability

```
(not (null? (set e)))
```

- model counting

```
(length (set e))
```

- optimization

```
(fold max minus-infinity (map f (set e)))
```

Utility of a Query or Aggregation Construct

Utility of a Query or Aggregation Construct

- marginal probablity

```
(fold + 0 (map cdr (remove-if-not car (distribution e))))
```

Utility of a Query or Aggregation Construct

- marginal probablity

```
(fold + 0 (map cdr (remove-if-not car (distribution e))))
```

- expectation

```
(fold +
      0
      (map (lambda (pair) (* (car pair) (cdr pair)))
           (distribution e)))
```

Utility of a Query or Aggregation Construct

- marginal probablity

```
(fold + 0 (map cdr (remove-if-not car (distribution e))))
```

- expectation

```
(fold +
      0
      (map (lambda (pair) (* (car pair) (cdr pair)))
           (distribution e)))
```

- maximum likelihood

```
(fold max minus-infinity (map cdr (distribution e)))
```

Utility of a Query or Aggregation Construct

- marginal probablity

```
(fold + 0 (map cdr (remove-if-not car (distribution e))))
```

- expectation

```
(fold +
      0
      (map (lambda (pair) (* (car pair) (cdr pair)))
           (distribution e)))
```

- maximum likelihood

```
(fold max minus-infinity (map cdr (distribution e)))
```

- entropy

```
(fold +
      0
      (map (lambda (pair) (* (cdr pair) (log (cdr pair))))
           (distribution e)))
```

Utility of a Query or Aggregation Construct

- marginal probablity

```
(fold + 0 (map cdr (remove-if-not car (distribution e))))
```

- expectation

```
(fold +
      0
      (map (lambda (pair) (* (car pair) (cdr pair)))
           (distribution e)))
```

- maximum likelihood

```
(fold max minus-infinity (map cdr (distribution e)))
```

- entropy

```
(fold +
      0
      (map (lambda (pair) (* (cdr pair) (log (cdr pair))))
           (distribution e)))
```

- mutual information
- KL divergence

Nesting Query or Aggregation Constructs

Notice that this also makes it quite all right for set expressions to be nested. For example one possible way to express the query:

"Which people drink each beverage?"

is:

```
?- setof(Beverage-People, setof(X, X drinks Beverage, People), S).
```

Warren (1982)

Nesting Query or Aggregation Constructs

```
(set (let ((beverage (choose beverages)))
       (cons beverage
             (set (let ((person (choose people)))
                    (if (drinks? person beverage)
                        person
                        (nondeterministic-fail))))))))
```

Nesting Query or Aggregation Constructs

```
(probability
 (let ((my-move (sample my-moves)))
  (> (probability
      (let ((your-move (sample your-moves)))
       (you-win? my-move your-move)))
     0.5)))
```

What is the probability that the probability that you will win is greater than 0.5?

Nesting Query or Aggregation Constructs

```
(possible?
 (let ((my-move (choose my-moves)))
  (> (probability
      (let ((your-move (sample your-moves)))
       (you-win? my-move your-move)))
     0.5)))
```

Is it possible that the probability that you will win is greater than 0.5?

Nesting Query or Aggregation Constructs

```
(probability
 (let ((my-move (sample my-moves)))
  (possible?
   (let ((your-move (choose your-moves)))
    (you-win? my-move your-move)))))
```

What is the probability that it is possible for you to win?

Nesting Query or Aggregation Constructs

Nesting Query or Aggregation Constructs

```
(possible?
 (let ((my-move (choose my-moves)))
  (> (probability
      (let ((your-move (sample your-moves)))
       (you-win? my-move your-move)))
     0.5)))

(probability
 (let ((my-move (sample my-moves)))
  (possible?
   (let ((your-move (choose your-moves)))
    (you-win? my-move your-move)))))
```

Nesting Query or Aggregation Constructs

```
(possible?
 (let ((my-move (choose my-moves)))
  (> (probability
      (let ((your-move (sample your-moves)))
       (you-win? my-move your-move)))
     0.5)))

(probability
 (let ((my-move (sample my-moves)))
  (possible?
   (let ((your-move (choose your-moves)))
    (you-win? my-move your-move)))))

(possible?
 (> (probability (you-win? (choose my-moves) (sample your-moves)))
    0.5))

(probability
 (possible? (you-win? (sample my-moves) (choose your-moves))))
```

Nesting Query or Aggregation Constructs

```
(possible?
 (let ((my-move (choose my-moves)))
  (> (probability
      (let ((your-move (sample your-moves)))
       (you-win? my-move your-move)))
     0.5)))

(probability
 (let ((my-move (sample my-moves)))
  (possible?
   (let ((your-move (choose your-moves)))
    (you-win? my-move your-move)))))

(possible?
 (> (probability (you-win? (choose my-moves) (sample your-moves)))
    0.5))

(probability
 (possible? (you-win? (sample my-moves) (choose your-moves))))

(possible-that-it-is-probable? (you-win? (playing-a-game)))

(probability-that-it-is-possible (you-win? (playing-a-game)))
```

The Rub

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
```

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
> (distribution (distribution (sample '((#t . 0.5) (#f . 0.5)))))
((((#t . 0.5) (#f . 0.5)) . 1))
```

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
> (distribution (distribution (sample '((#t . 0.5) (#f . 0.5)))))
((((#t . 0.5) (#f . 0.5)) . 1))
> (distribution (set (choose '(#t #f))))
(((#t #f) . 1))
```

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
> (distribution (distribution (sample '((#t . 0.5) (#f . 0.5)))))
((((#t . 0.5) (#f . 0.5)) . 1))
> (distribution (set (choose '(#t #f))))
(((#t #f) . 1))
> (set (distribution (sample '((#t . 0.5) (#f . 0.5)))))
(((#t . 0.5) (#f . 0.5)))
```

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
> (distribution (distribution (sample '((#t . 0.5) (#f . 0.5)))))
((((#t . 0.5) (#f . 0.5)) . 1))
> (distribution (set (choose '(#t #f))))
(((#t #f) . 1))
> (set (distribution (sample '((#t . 0.5) (#f . 0.5)))))
(((#t . 0.5) (#f . 0.5)))
> (set (distribution (choose '(#t #f))))
Unhandled exception
 Condition components:
   1. &error
   2. &message:  "Top-level probabilistic-fail"
   3. &irritants:  ()
```

The Rub

```
> (set (set (choose '(#t #f))))
((#t #f))
> (distribution (distribution (sample '((#t . 0.5) (#f . 0.5)))))
((((#t . 0.5) (#f . 0.5)) . 1))
> (distribution (set (choose '(#t #f))))
(((#t #f) . 1))
> (set (distribution (sample '((#t . 0.5) (#f . 0.5)))))
(((#t . 0.5) (#f . 0.5)))
> (set (distribution (choose '(#t #f))))
Unhandled exception
 Condition components:
   1. &error
   2. &message:  "Top-level probabilistic-fail"
   3. &irritants:  ()
> (distribution (set (sample '((#t . 0.5) (#f . 0.5)))))
Unhandled exception
 Condition components:
   1. &error
   2. &message:  "Top-level nondeterministic-fail"
   3. &irritants:  ()
```

The Untyped Nondeterministic Lambda Calculus

$$E ::= x \mid \lambda x.e \mid e_0\ e_1 \mid \text{choose } e \mid \text{set } e$$

The Untyped Nondeterministic Lambda Calculus

$$
\begin{aligned}
\mathcal{S}\ v &= \{v\} \\
\mathcal{M}\ f\ \{v_1, \ldots, v_n, \ldots\} &= \{(f\ v_1), \ldots, (f\ v_n), \ldots\} \\
\bigcup \{\{v_{11}, \ldots, v_{1n}, \ldots\}, \ldots, \{v_{m1}, \ldots, v_{mn}, \ldots\}, \ldots\} &= \{v_{11}, \ldots, v_{1n}, \ldots, \ldots, v_{m1}, \ldots, v_{mn}, \ldots, \ldots\}
\end{aligned}
$$

The Untyped Nondeterministic Lambda Calculus

$$
\begin{aligned}
\mathcal{E}\ \rho\ x &= \mathcal{S}\ (\rho\ x) \\
\mathcal{E}\ \rho\ \lambda x.e &= \mathcal{S}\ \lambda v.\mathcal{E}\ \rho[x \mapsto v]\ e \\
\mathcal{E}\ \rho\ (e_0\ e_1) &= \bigcup\ (\mathcal{M}\ (\lambda v_0.\bigcup\ (\mathcal{M}\ (\lambda v_1.v_0\ v_1)\ (\mathcal{E}\ \rho\ e_1)))\ (\mathcal{E}\ \rho\ e_0)) \\
\mathcal{E}\ \rho\ (\mathsf{choose}\ e) &= \bigcup\ (\mathcal{E}\ \rho\ e) \\
\mathcal{E}\ \rho\ (\mathsf{set}\ e) &= \mathcal{S}\ (\mathcal{E}\ \rho\ e)
\end{aligned}
$$

The Untyped Stratified Nondeterministic Lambda Calculus

$$E ::= x \mid \lambda x.e \mid e_0\ e_1 \mid \mathsf{choose}_1\ e \mid \mathsf{set}_1\ e \mid \mathsf{choose}_2\ e \mid \mathsf{set}_2\ e$$

The Untyped Stratified Nondeterministic Lambda Calculus

$$\left\{ \begin{array}{cccc} v_{11} & \cdots & v_{1n} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{m1} & \cdots & v_{mn} & \cdots \\ \vdots & & \vdots & \ddots \end{array} \right\}$$

The Untyped Stratified Nondeterministic Lambda Calculus

$$\mathcal{S}\, v \;=\; \{\, v \,\}$$

$$\mathcal{M}\, f \begin{Bmatrix} v_{11} & \cdots & v_{1n} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{m1} & \cdots & v_{mn} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} = \begin{Bmatrix} (f\ v_{11}) & \cdots & (f\ v_{1n}) & \cdots \\ \vdots & \ddots & \vdots & \\ (f\ v_{m1}) & \cdots & (f\ v_{mn}) & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix}$$

$$\bigcup \begin{Bmatrix} \begin{Bmatrix} v_{1111} & \cdots & v_{111j} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{11i1} & \cdots & v_{11ij} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} & \cdots & \begin{Bmatrix} v_{1u11} & \cdots & v_{1u1l} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{1uk1} & \cdots & v_{1ukl} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} & \cdots \\ \vdots & \ddots & \vdots & \\ \begin{Bmatrix} v_{t111} & \cdots & v_{t11n} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{t1m1} & \cdots & v_{t1mn} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} & \cdots & \begin{Bmatrix} v_{tu11} & \cdots & v_{tu1s} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{tur1} & \cdots & v_{turs} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} & \cdots \\ \vdots & & \vdots & \ddots \end{Bmatrix} = \begin{Bmatrix} v_{1111} & \cdots & v_{111j} & \cdots & v_{1u11} & \cdots & v_{1u1l} & \cdots \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots & \\ v_{11i1} & \cdots & v_{11ij} & \cdots & v_{1uk1} & \cdots & v_{1ukl} & \cdots \\ \vdots & & \vdots & \ddots & \vdots & & \vdots & \ddots \\ v_{t111} & \cdots & v_{t11n} & \cdots & v_{tu11} & \cdots & v_{tu1s} & \cdots \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots & \\ v_{t1m1} & \cdots & v_{t1mn} & \cdots & v_{tur1} & \cdots & v_{turs} & \cdots \\ \vdots & & \vdots & \ddots & \vdots & & \vdots & \ddots \end{Bmatrix}$$

Note that $\bigcup$ is defined only when the row (column) cardinalities of all of the inner two-axis sets in each column (row) of a given row (column) in the outer two-axis set are the same.

The Untyped Stratified Nondeterministic Lambda Calculus

$$\mathbf{inject}_1\ \{v_1, \ldots, v_n, \ldots\} = \left\{ \begin{array}{c} v_1 \\ \vdots \\ v_n \\ \vdots \end{array} \right\}$$

$$\mathbf{inject}_2\ \{v_1, \ldots, v_n, \ldots\} = \{\ v_1 \quad \ldots \quad v_n \quad \ldots\ \}$$

The Untyped Stratified Nondeterministic Lambda Calculus

$$\mathbf{project}_1 \left\{ \begin{array}{c} v_1 \\ \vdots \\ v_n \\ \vdots \end{array} \right\} = \{v_1, \ldots, v_n, \ldots\}$$

$$\mathbf{project}_2 \left\{ \begin{array}{cccc} v_1 & \ldots & v_n & \ldots \end{array} \right\} = \{v_1, \ldots, v_n, \ldots\}$$

The Untyped Stratified Nondeterministic Lambda Calculus

$$\mathcal{S}_1 \left\{ \begin{array}{cccc} v_{11} & \cdots & v_{1n} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{m1} & \cdots & v_{mn} & \cdots \\ \vdots & & \vdots & \ddots \end{array} \right\} = \left\{ \left\{ \begin{array}{c} v_{11} \\ \vdots \\ v_{m1} \\ \vdots \end{array} \right\} \cdots \left\{ \begin{array}{c} v_{1n} \\ \vdots \\ v_{mn} \\ \vdots \end{array} \right\} \cdots \right\}$$

$$\mathcal{S}_2 \left\{ \begin{array}{cccc} v_{11} & \cdots & v_{1n} & \cdots \\ \vdots & \ddots & \vdots & \\ v_{m1} & \cdots & v_{mn} & \cdots \\ \vdots & & \vdots & \ddots \end{array} \right\} = \left\{ \begin{array}{ccccc} \{ & v_{11} & \cdots & v_{1n} & \cdots \ \} \\ & & \vdots & & \\ \{ & v_{m1} & \cdots & v_{mn} & \cdots \ \} \\ & & \vdots & & \end{array} \right\}$$

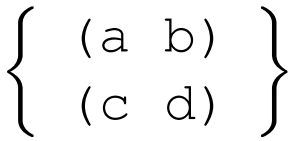
The Untyped Stratified Nondeterministic Lambda Calculus

$$
\begin{aligned}
\mathcal{E}\ \rho\ x &= \mathcal{S}\ (\rho\ x) \\
\mathcal{E}\ \rho\ \lambda x.e &= \mathcal{S}\ \lambda v.\mathcal{E}\ \rho[x \mapsto v]\ e \\
\mathcal{E}\ \rho\ (e_0\ e_1) &= \bigcup\ (\mathcal{M}\ (\lambda v_0.\bigcup\ (\mathcal{M}\ (\lambda v_1.v_0\ v_1)\ (\mathcal{E}\ \rho\ e_1)))\ (\mathcal{E}\ \rho\ e_0)) \\
\mathcal{E}\ \rho\ (\mathsf{choose}_1\ e) &= \bigcup\ (\mathcal{M}\ \mathbf{inject}_1\ (\mathcal{E}\ \rho\ e)) \\
\mathcal{E}\ \rho\ (\mathsf{set}_1\ e) &= \mathcal{M}\ \mathbf{project}_1\ (\mathcal{S}_1\ (\mathcal{E}\ \rho\ e)) \\
\mathcal{E}\ \rho\ (\mathsf{choose}_2\ e) &= \bigcup\ (\mathcal{M}\ \mathbf{inject}_2\ (\mathcal{E}\ \rho\ e)) \\
\mathcal{E}\ \rho\ (\mathsf{set}_2\ e) &= \mathcal{M}\ \mathbf{project}_2\ (\mathcal{S}_2\ (\mathcal{E}\ \rho\ e))
\end{aligned}
$$

One Issue

One Issue

```
> (choose1 '((a b) (c d)))
```



One Issue

```
> (choose1 '((a b) (c d)))
{ (a b) }
{ (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b }
{ c  d }
```

One Issue

```
> (choose1 '((a b) (c d)))
{ (a b) }
{ (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b }
{ c  d }
> (set2 (choose2 (choose1 '((a b) (c d)))))
{ (a b) }
{ (c d) }
```

One Issue

```
> (choose1 '((a b) (c d)))
{ (a b) }
{ (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b }
{ c  d }
> (set2 (choose2 (choose1 '((a b) (c d)))))
{ (a b) }
{ (c d) }
> (set1 (set2 (choose2 (choose1 '((a b) (c d))))))
((a b) (c d))
```

One Issue

```
> (choose1 '((a b) (c d)))
{ (a b)
  (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b
  c  d }
> (set2 (choose2 (choose1 '((a b) (c d)))))
{ (a b)
  (c d) }
> (set1 (set2 (choose2 (choose1 '((a b) (c d))))))
((a b) (c d))
> (set1 (choose2 (choose1 '((a b) (c d)))))
{ (a c)  (b d) }
```

One Issue

```
> (choose1 '((a b) (c d)))
{ (a b) }
{ (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b }
{ c  d }
> (set2 (choose2 (choose1 '((a b) (c d)))))
{ (a b) }
{ (c d) }
> (set1 (set2 (choose2 (choose1 '((a b) (c d))))))
((a b) (c d))
> (set1 (choose2 (choose1 '((a b) (c d)))))
{ (a c)  (b d) }
> (set2 (set1 (choose2 (choose1 '((a b) (c d))))))
((a c) (b d))
```

One Issue

```
> (choose1 '((a b) (c d)))
{ (a b) }
{ (c d) }
> (choose2 (choose1 '((a b) (c d))))
{ a  b }
{ c  d }
> (set2 (choose2 (choose1 '((a b) (c d)))))
{ (a b) }
{ (c d) }
> (set1 (set2 (choose2 (choose1 '((a b) (c d))))))
((a b) (c d))
> (set1 (choose2 (choose1 '((a b) (c d)))))
{ (a c)  (b d) }
> (set2 (set1 (choose2 (choose1 '((a b) (c d))))))
((a c) (b d))
> (set2 (set1 (choose2 (choose1 '((a) (c d))))))
>>Error: Invalid argument to ⋃
```

The Untyped Stratified Probabilistic Lambda Calculus

$$\mathcal{S}_1 \left\{ \begin{array}{cccc} \langle v_{11}, p^1_{11}, p^2_{11} \rangle & \cdots & \langle v_{1n}, p^1_{1n}, p^2_{1n} \rangle & \cdots \\ \vdots & \ddots & \vdots & \\ \langle v_{m1}, p^1_{m1}, p^2_{m1} \rangle & \cdots & \langle v_{mn}, p^1_{mn}, p^2_{mn} \rangle & \cdots \\ \vdots & & \vdots & \ddots \end{array} \right\} = \left\{ \begin{array}{ccc} \left\langle \left\{ \begin{array}{c} \langle v_{11}, p^1_{11}, p^2_{11} + \cdots + p^2_{1n} \rangle \\ \vdots \\ \langle v_{m1}, p^1_{m1}, p^2_{m1} + \cdots + p^2_{mn} \rangle \\ \vdots \end{array} \right\}, 1, p^2_{11} \right\rangle & \cdots & \left\langle \left\{ \begin{array}{c} \langle v_{1n}, p^1_{1n}, p^2_{11} + \cdots + p^2_{1n} \rangle \\ \vdots \\ \langle v_{mn}, p^1_{mn}, p^2_{m1} + \cdots + p^2_{mn} \rangle \\ \vdots \end{array} \right\}, 1, p^2_{1n} \right\rangle \quad \cdots \end{array} \right\}$$

$$\mathcal{S}_2 \left\{ \begin{array}{cccc} \langle v_{11}, p^1_{11}, p^2_{11} \rangle & \cdots & \langle v_{1n}, p^1_{1n}, p^2_{1n} \rangle & \cdots \\ \vdots & \ddots & \vdots & \\ \langle v_{m1}, p^1_{m1}, p^2_{m1} \rangle & \cdots & \langle v_{mn}, p^1_{mn}, p^2_{mn} \rangle & \cdots \\ \vdots & & \vdots & \ddots \end{array} \right\} = \left\{ \begin{array}{ccc} \langle \{ \langle v_{11}, p^1_{11} + \cdots + p^1_{m1}, p^2_{11} \rangle & \cdots & \langle v_{1n}, p^1_{1n} + \cdots + p^1_{mn}, p^2_{1n} \rangle \quad \cdots \} , p^1_{11}, 1 \rangle \\ & & \vdots \\ \langle \{ \langle v_{m1}, p^1_{11} + \cdots + p^1_{m1}, p^2_{m1} \rangle & \cdots & \langle v_{mn}, p^1_{1n} + \cdots + p^1_{mn}, p^2_{mn} \rangle \quad \cdots \} , p^1_{m1}, 1 \rangle \\ & & \vdots \end{array} \right\}$$

Note that $\mathcal{S}_1$ is defined only when the row vectors of column probabilities are the same across all rows, and $\mathcal{S}_2$ is defined only when the columns vectors of row probabilities are the same across all columns.

Another Issue

Another Issue

```
> (sample1
   '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))
```

$$\left\{ \begin{array}{l} \langle \texttt{((a . 0.6) (b . 0.4))}, 0.9, 1 \rangle \\ \langle \texttt{((c . 0.8) (d . 0.2))}, 0.1, 1 \rangle \end{array} \right\}$$

Another Issue

```
> (sample1
   '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))
```

$$\left\{ \begin{array}{l} \langle \texttt{((a . 0.6) (b . 0.4))}, 0.9, 1 \rangle \\ \langle \texttt{((c . 0.8) (d . 0.2))}, 0.1, 1 \rangle \end{array} \right\}$$

```
> (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \texttt{a}, 0.9, 0.6 \rangle & \langle \texttt{b}, 0.9, 0.4 \rangle \\ \langle \texttt{c}, 0.1, 0.8 \rangle & \langle \texttt{d}, 0.1, 0.2 \rangle \end{array} \right\}$$

Another Issue

```
> (sample1
   '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))
```

$$\left\{ \begin{array}{l} \langle \texttt{((a . 0.6) (b . 0.4))}, 0.9, 1 \rangle \\ \langle \texttt{((c . 0.8) (d . 0.2))}, 0.1, 1 \rangle \end{array} \right\}$$

```
> (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \texttt{a}, 0.9, 0.6 \rangle & \langle \texttt{b}, 0.9, 0.4 \rangle \\ \langle \texttt{c}, 0.1, 0.8 \rangle & \langle \texttt{d}, 0.1, 0.2 \rangle \end{array} \right\}$$

```
> (distribution2
    (sample2
     (sample1
      '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))))
```

$$\left\{ \begin{array}{l} \langle \texttt{((a . 0.6) (b . 0.4))}, 0.9, 1 \rangle \\ \langle \texttt{((c . 0.8) (d . 0.2))}, 0.1, 1 \rangle \end{array} \right\}$$

Another Issue

```
> (sample1
   '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))
```

$$\left\{ \begin{array}{l} \langle ((\mathtt{a}\ .\ 0.6)\ (\mathtt{b}\ .\ 0.4)), 0.9, 1\rangle \\ \langle ((\mathtt{c}\ .\ 0.8)\ (\mathtt{d}\ .\ 0.2)), 0.1, 1\rangle \end{array} \right\}$$

```
> (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \mathtt{a}, 0.9, 0.6\rangle & \langle \mathtt{b}, 0.9, 0.4\rangle \\ \langle \mathtt{c}, 0.1, 0.8\rangle & \langle \mathtt{d}, 0.1, 0.2\rangle \end{array} \right\}$$

```
> (distribution2
    (sample2
     (sample1
      '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))))
```

$$\left\{ \begin{array}{l} \langle ((\mathtt{a}\ .\ 0.6)\ (\mathtt{b}\ .\ 0.4)), 0.9, 1\rangle \\ \langle ((\mathtt{c}\ .\ 0.8)\ (\mathtt{d}\ .\ 0.2)), 0.1, 1\rangle \end{array} \right\}$$

```
> (distribution1
    (distribution2
     (sample2
      (sample1
       '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))))
((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))
```

Another Issue

```
> (sample1
   '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))
{ ⟨((a . 0.6) (b . 0.4)),0.9,1⟩ }
{ ⟨((c . 0.8) (d . 0.2)),0.1,1⟩ }
> (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))
{ ⟨a,0.9,0.6⟩ ⟨b,0.9,0.4⟩ }
{ ⟨c,0.1,0.8⟩ ⟨d,0.1,0.2⟩ }
> (distribution2
    (sample2
     (sample1
      '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))))
{ ⟨((a . 0.6) (b . 0.4)),0.9,1⟩ }
{ ⟨((c . 0.8) (d . 0.2)),0.1,1⟩ }
> (distribution1
    (distribution2
     (sample2
      (sample1
       '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))))
((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))
> (distribution1
    (sample2
     (sample1
      '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1)))))
>>Error: Invalid argument to S1
{ ⟨((a . 0.9) (c . 0.1)),1,?⟩ ⟨((b . 0.9) (d . 0.1)),1,?⟩ }
```

Another Issue

Another Issue

```
> (sample2
   (sample1
    '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.6) (d . 0.4)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \mathtt{a}, 0.9, 0.6 \rangle & \langle \mathtt{b}, 0.9, 0.4 \rangle \\ \langle \mathtt{c}, 0.1, 0.6 \rangle & \langle \mathtt{d}, 0.1, 0.6 \rangle \end{array} \right\}$$

Another Issue

```
> (sample2
   (sample1
    '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.6) (d . 0.4)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \mathrm{a}, 0.9, 0.6 \rangle & \langle \mathrm{b}, 0.9, 0.4 \rangle \\ \langle \mathrm{c}, 0.1, 0.6 \rangle & \langle \mathrm{d}, 0.1, 0.6 \rangle \end{array} \right\}$$

```
> (distribution1
   (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.6) (d . 0.4)) . 0.1)))))
```

$$\{ \langle \texttt{((a . 0.9) (c . 0.1))}, 1, 0.6 \rangle \ \langle \texttt{((b . 0.9) (d . 0.1))}, 1, 0.4 \rangle \}$$

Another Issue

```
> (sample2
   (sample1
    '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.6) (d . 0.4)) . 0.1))))
```

$$\left\{ \begin{array}{ll} \langle \mathrm{a}, 0.9, 0.6 \rangle & \langle \mathrm{b}, 0.9, 0.4 \rangle \\ \langle \mathrm{c}, 0.1, 0.6 \rangle & \langle \mathrm{d}, 0.1, 0.6 \rangle \end{array} \right\}$$

```
> (distribution1
   (sample2
    (sample1
     '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.6) (d . 0.4)) . 0.1)))))
```

$$\{ \langle \texttt{((a . 0.9) (c . 0.1))}, 1, 0.6 \rangle \ \langle \texttt{((b . 0.9) (d . 0.1))}, 1, 0.4 \rangle \}$$

```
> (distribution2
   (distribution1
    (sample2
     (sample1
      '((((a . 0.6) (b . 0.4)) . 0.9) (((c . 0.8) (d . 0.2)) . 0.1))))))
((((a . 0.9) (c . 0.1)) . 0.6) (((b . 0.9) (d . 0.1)) . 0.4))
```

Coherency Conditions

Coherency Conditions

1. The number of choices at every stratum must be the same for all samples/choices of all other strata.

Coherency Conditions

1. The number of choices at every stratum must be the same for all samples/choices of all other strata.
2. The distribution at every stratum must be the same for all samples/choices of all other strata.