

Mlite : Adding Matrices to Clite

Andrew Dreher and Jeremy Stober

Final Presentation
Programming Languages

Why Matrices?

- Matrices are pervasive in computer science
 - Computer graphics
 - Machine learning
 - Scientific computing/modeling
- Matrix semantics have much in common with *IntValue* and *FloatValue*
 - Certain types of matrices form groups, rings, and fields
- Some key differences – operations are not always defined
 - Matrix dimensions may not agree
 - Many matrices may have no inverse
- Typing matrices is difficult – parametric in dimension and element type

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

Initialization

```
int main () {
    matrix A,B,C,D,E,F,G,M,N;
    float x;

    x = 89.5;
    M = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]];
    N = [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
    A = M * N;
    B = M - N;
    B = M + M;
    C = 4.3 * M - 0.7;
    G = (\ elem -> 4.3 * elem - 0.7; x == 1.0) N;
    D = M*4.3;

    E = M;
    E[1,1] = 75.0;

    F = M - [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
}
```

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

Matrix-Matrix Operations

```
int main () {
    matrix A,B,C,D,E,F,G,M,N;
    float x;

    x = 89.5;
    M = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]];
    N = [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
    A = M * N;
    B = M - N;
    B = M + M;
    C = 4.3 * M - 0.7;
    G = (\ elem -> 4.3 * elem - 0.7; x == 1.0) N;
    D = M*4.3;

    E = M;
    E[1,1] = 75.0;

    F = M - [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
}
```

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

Matrix-Matrix Operations

```
int main () {
    matrix A,B,C,D,E,F,G,M,N;
    float x;

    x = 89.5;
    M = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]];
    N = [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
    A = M * N;
    B = M - N;
    B = M + M;
    C = 4.3 * M - 0.7;
    G = (\ elem -> 4.3 * elem - 0.7; x == 1.0) N;
    D = M*4.3;

    E = M;
    E[1,1] = 75.0;

    F = M - [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
}
```

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

Lambda Operations

```
int main () {
    matrix A,B,C,D,E,F,G,M,N;
    float x;

    x = 89.5;
    M = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]];
    N = [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
    A = M * N;
    B = M - N;
    B = M + M;
    C = 4.3 * M - 0.7;
    G = (\ elem -> 4.3 * elem - 0.7; x == 1.0) N;
    D = M*4.3;

    E = M;
    E[1,1] = 75.0;

    F = M - [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
}
```

Matrix Features

- Matrix entries are expressions
 - Must evaluate to FloatValue - checked at runtime
- Natural initialization syntax
- Support for common Matrix-Matrix operations: $+$, $-$, \times
- Support for common Scalar-Matrix operations: $+$, $-$, \times , $/$
- *Lambda* syntax for element by element function application
- Access to individual elements via *MatrixCell* syntax

MatrixCell Access

```
int main () {
    matrix A,B,C,D,E,F,G,M,N;
    float x;

    x = 89.5;
    M = [[1.0, 0.0, 0.0],[0.0, 1.0, 0.0],[0.0, 0.0, 1.0]];
    N = [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
    A = M * N;
    B = M - N;
    B = M + M;
    C = 4.3 * M - 0.7;
    G = (\ elem -> 4.3 * elem - 0.7; x == 1.0) N;
    D = M*4.3;

    E = M;
    E[1,1] = 75.0;

    F = M - [[0.0, 2.0, 0.0],[1.0, 1.0, 7.0],[5.0, 4.0, 1.0]];
}
```

An Example: Fibonacci

```
int main() {
    matrix f, r;
    int position, i;
    float value;

    f = [[1.0, (5.0-4.0)], [1.0, 0.0]];
    r = f;
    i = 1;
    position = 3;
    while(i < position) {
        r = r*f;
        i = i+1;
    }
    value = r[(5-4), ((i*3 + 4)*0)];
}
```

- Fibonacci number can be expressed in matrix form as:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

An Example: Floyd-Warshall

```
float min(float a, float b) {
    if(a < b)
        return a;
    else
        return b;
}

int main() {
    matrix path;
    int k, i, j, n;

    path = [[1.0, 2.0, 3.0, 1.0],
            [-1.0, 1.0, 2.0, 1.0],
            [3.0, 3.0, 1.0, 2.0],
            [1.0, 3.0, 3.0, 1.0]];
    n = 4;

    k = 0; i = 0; j = 0;
    for k to n-1 {
        for i to n-1 {
            for j to n-1 {
                path[i,j] = min( path[i,j], path[i,k] + path[k,j] );
            }
        }
    }
}
```

An Example: Probabilistic Model Checking

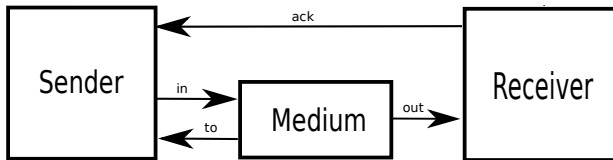


Figure: Communication over an unreliable medium.¹

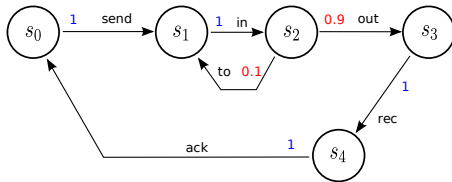


Figure: The behavior of Parrow's Protocol.

¹Taken from *A Framework for Reasoning about Time and Reliability* by Hans Hansson and Bengt Jonsson.

A Question

Will “rec” appear in the first 6 timesteps at least 99% of the time?

- We can answer questions like this using probabilistic model checking.

Example Paths

- Compute the probability weight of the paths (product rule)
- Example One:

$$s_0 \xrightarrow{1.0} s_1 \xrightarrow{1.0} s_2 \xrightarrow{0.9} s_3 \xrightarrow{1.0} s_4$$

- Example Two:

$$s_0 \xrightarrow{1.0} s_1 \xrightarrow{1.0} s_2 \xrightarrow{0.1} s_1 \xrightarrow{1.0} s_2 \xrightarrow{0.9} s_3 \xrightarrow{1.0} s_4$$

As Matrix Multiplication

- T is the *transition matrix*:

| | s_0 | s_1 | s_2 | s_3 | s_4 |
|-------|-------|-------|-------|-------|-------|
| s_0 | 0 | 1 | 0 | 0 | 0 |
| s_1 | 0 | 0 | 1 | 0 | 0 |
| s_2 | 0 | 0.1 | 0 | 0.9 | 0 |
| s_3 | 0 | 0 | 0 | 0 | 1 |
| s_4 | 0 | 0 | 0 | 0 | 1 |

- We have the following identity:

$$P(s, t) = T^t P(s, 0)$$

- We can find the probability of paths $s_0 \rightsquigarrow^{\leq 6} s_4$:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0.1 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^6 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.99 \\ 0.99 \\ 0.999 \\ 1 \\ 1 \end{pmatrix}$$

An Example: Probabilistic Model Checking

```
int main () {
    matrix T,P,R;

    T = [[0.0, 1.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 1.0, 0.0, 0.0],
[0.0, 0.1, 0.0, 0.9, 0.0],
[0.0, 0.0, 0.0, 0.0, 1.0],
[0.0, 0.0, 0.0, 0.0, 1.0]];

    P = [[0.0],
[0.0],
[0.0],
[0.0],
[1.0]];

    R = T * T * T * T * T * T * T * P;
}
```

An Example: Approximating Stationary Distributions

```
float max(matrix M) {
  int i,j;
  float m;
  m = 0.0;
  i = 0;
  j = 0;
  while (i < 3) {
    j = 0;
    while (j < 3) {
      if (M[i,j] > 0.0){
        if (m < M[i,j]){
          m = M[i,j];
        }
      } else {
        if (m < - M[i,j]){
          m = - M[i,j];
        }
      }
      j = j + 1;
    }
    i = i + 1;
  }
  return m;
}
```

```
int main () {
  matrix M,N;
  float x;
  M = [[0.1,0.6,0.3],[0.25,0.05,0.7],[0.7,0.3,0.0]];
  N = [[0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0]];
  while(max(M - N) > 0.01){
    N = M;
    M = M * M;
  }
}
```