Faster Reachability Analysis for LR(1) Parsers

Frédéric Bour  
Tarides & Inria  
Paris, France

François Pottier  
Inria  
Paris, France
Plan

- The reachability problem for LR(1) automata
- State-of-the-art solution & performance comparison
- Main ideas of our contribution
- Conclusion
Reachability in LR(1) automata
**What is the problem?**

“Can the automaton reach a configuration \((s, z)\)?”

- \(s\) is the current state
- \(z\) is the first unconsumed symbol
What is the problem?

“Can the automaton reach a configuration \((s,z)\)?”

- \(s\) is the current state
- \(z\) is the first unconsumed symbol

In practice, we also want a **minimal sentence** that reaches this configuration.
Why solve it?
Why solve it?

- Test case generation
Why solve it?

• Test case generation

  • **Negative test cases** (our main focus)
    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)
Why solve it?

• Test case generation

• **Negative test cases** (our main focus)
  Enumerate sentences that cause errors in all states that can fail
  (Jeffery 2003, Pottier 2016)

Assistance to write error message:

```
translation_unit_file: INT PRE_NAME VAR_NAME EQ XOR_ASSIGN
## Ends in an error in state: 561.

*Ill-formed init declarator.*
*At this point, an initializer is expected.*
```
Why solve it?

- Test case generation
  - **Negative test cases** (our main focus)
    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)
  - Positive test cases
    Cover all reductions for regression testing,
    check compatibility between different grammar versions,
    ...
Why solve it?

• Test case generation
  • **Negative test cases** (our main focus)
    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)
  • Positive test cases
    Cover all reductions for regression testing,
    check compatibility between different grammar versions,
    ...

• Syntactic completion, syntactic error recovery, ...
State-of-the-art solution & performance comparison
Pottier’s algorithm (2016)
Pottier’s algorithm (2016)

• Implemented in the Menhir parser generator
Pottier’s algorithm (2016)

- Implemented in the Menhir parser generator
- Applied to CompCert to obtain high-quality error messages
Pottier’s algorithm (2016)

- Implemented in the Menhir parser generator
- Applied to CompCert to obtain high-quality error messages

But it does not scale well!
Pottier’s algorithm (2016)

A few data points:
- CompCert (C): 25s and 529MB
- Unicon: 566s and 8.5GB

Problems:
- Too slow for interactive use
- Painful for grammar maintainers
Pottier’s algorithm (2016)

The algorithm works in two steps:
Pottier’s algorithm (2016)

The algorithm works in two steps:

1. For each transition, find the shortest input that allows taking it (while satisfying constraints on lookahead tokens)
Pottier’s algorithm (2016)

The algorithm works in two steps:

1. For each transition, find the shortest input that allows taking it (while satisfying constraints on lookahead tokens)
2. Generate minimal sentences by taking consecutive transitions
Pottier’s algorithm (2016)

The algorithm works in two steps:

1. For each transition, find the shortest input that allows taking it (while satisfying constraints on lookahead tokens)
2. Generate minimal sentences by taking consecutive transitions

The bottleneck by far is step 1.
We propose a new algorithm to solve it.
Our contribution: speeding up the analysis!

Original algorithm.
Our contribution: speeding up the analysis!

First step: a “naïve” matrix-based formulation (faster! but memory hungry)
Our contribution: speeding up the analysis!

Second step: compact matrices, two to three orders of magnitude better, in time and space.
Our contribution: speeding up the analysis!

Updated data points:

- CompCert (C): **0.10s and 12MB** (was 25s and 529MB)
- Unicon: **0.28s and 32MB** (was 566s and 8.5GB)
Our contribution: speeding up the analysis!

Updated data points:

- CompCert (C): **0.10s and 12MB** (was 25s and 529MB)
- Unicon: **0.28s and 32MB** (was 566s and 8.5GB)

Can still take some time: a “rich” C++ grammar that takes 56s and 2.7GB.

(grammer from “Diff/TS: A tool for fine-grained structural change analysis” by Hashimoto and Mori)
Idea #1: costs with matrices
An example grammar

Let’s consider this LR(1) grammar:

\[
S ::= T \ a \\
| T \ b \ b \\
\]

\[
T ::= a \\
| a \ a \\
| a \ a \\
\]
The automaton

It turns into the following LR(1) automaton, with one SHIFT/REDUCE conflict
The automaton

It turns into the following LR(1) automaton, with one SHIFT/REDUCE conflict.
The automaton

It turns into the following LR(1) automaton, with one SHIFT/REDUCE conflict.
The automaton

It turns into the following LR(1) automaton, with one SHIFT/REDUCE conflict

```
S ::= . T a
   | . T b b

S ::= T . a
   | T . b b

T ::= a .
   | a . a

S ::= T b . b

S ::= T . a
   reduce S ::= T a

S ::= T b b
   reduce S ::= T b b

S ::= T b b
   reduce S ::= T b b

T ::= a a.
   reduce T ::= a a
```

**s0**

**s1**

**s2**

**s3**

**s4**

**s5**

**s6**
Let's say we decide to **SHIFT**
Conflict resolution

Let’s say we decide to SHIFT

```
S ::= . T a
  | . T b b
S ::= T . a
  | T . b b
T ::= a .
  | a . a
S ::= T a .
S ::= T b . b
S ::= T b b .
```

```
s0
  s1
  T

T ::= a a .
  | a . a
S ::= T b b .
S ::= T a
  reduce S ::= T a
  reduce S ::= T b b
  reduce S ::= T a
  reduce T ::= a
  reduce T ::= a a
```
Conflict resolution

Let’s say we decide to SHIFT
Cost equations

A first attempt at finding costs
Cost equations

A first attempt at finding costs
Cost equations

A first attempt at finding costs

\[
\begin{align*}
s0 & \rightarrow s1 \\
S ::= & . T a \mid . T b b & s1 \\
S ::= & T . a \mid T . b b & s2 \\
T ::= & a . & s3 \\
S ::= & T . a \rightarrow reduce S ::= T a & s4 \\
S ::= & T b . b & s5 \\
S ::= & T b b \rightarrow reduce S ::= T b b & s6 \\
T ::= & a a . & s7 \\
T ::= & a . a \rightarrow reduce T ::= a & s8 \\
T ::= & a a \rightarrow reduce T ::= a a & s9 
\end{align*}
\]
Cost equations

A first attempt at finding costs

<table>
<thead>
<tr>
<th>Equation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost(s0, a)</td>
<td>1</td>
</tr>
<tr>
<td>cost(s1, a)</td>
<td>1</td>
</tr>
<tr>
<td>cost(s5, a)</td>
<td>1</td>
</tr>
<tr>
<td>cost(s1, b)</td>
<td>1</td>
</tr>
<tr>
<td>cost(s3, b)</td>
<td>1</td>
</tr>
</tbody>
</table>
**Cost equations**

A first attempt at finding costs

\[
\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1
\end{align*}
\]
Cost equations

A first attempt at finding costs

cost(s0, a) = 1
cost(s1, a) = 1
cost(s5, a) = 1
cost(s1, b) = 1
cost(s3, b) = 1

cost(s0, T)
Cost equations

A first attempt at finding costs
Cost equations

A first attempt at finding costs

cost(s0, a) = 1
cost(s1, a) = 1
cost(s5, a) = 1

T ::= . T a
| . T . b b

S ::= T . a
| T . b b

T ::= a .
| a . a

S ::= T b . b

S ::= T a .
reduce S ::= T a

S ::= T b b .
reduce S ::= T b b

T ::= a a .
reduce T ::= a a

T ::= a a

S ::= T a
reduce S ::= T a

S ::= T b b
reduce S ::= T b b

T ::= a
reduce T ::= a

T ::= a
reduce T ::= a

T ::= . T a
| . T . b b

S ::= T . a
| T . b b

T ::= a .
| a . a

S ::= T b . b

S ::= T a .
reduce S ::= T a

S ::= T b b .
reduce S ::= T b b

T ::= a a .
reduce T ::= a a

T ::= a a
reduce T ::= a a

T ::= a
reduce T ::= a

T ::= a
reduce T ::= a
Cost equations

A first attempt at finding costs

cost(s0, a) = 1
cost(s1, a) = 1
cost(s5, a) = 1

cost(s0, T) = \min\{ cost(s0, a) + cost(s5, a) \}
Cost equations

A first attempt at finding costs

\[
\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1 \\
\text{cost}(s_0, T) &= \min \bigg\{ \text{cost}(s_0, a) + \text{cost}(s_5, a) \bigg\} \\
&= 1
\end{align*}
\]
Cost equations

A first attempt at finding costs

\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1 \\
\text{cost}(s_0, T) &= \min \left\{ \text{cost}(s_0, a), \text{cost}(s_0, a) + \text{cost}(s_5, a) \right\} = 1 \\
\text{cost}(s_0, S) &= \min \\
\end{align*}
Cost equations

A first attempt at finding costs

\[
\begin{align*}
cost(s_0, a) &= 1 \\
cost(s_1, a) &= 1 \\
cost(s_5, a) &= 1 \\
cost(s_1, b) &= 1 \\
cost(s_3, b) &= 1
\end{align*}
\]

\[
\begin{align*}
cost(s_0, T) &= \min\left\{ cost(s_0, a) \right\} \\
&= 1 \\
cost(s_0, S) &= \min\left\{ cost(s_0, T) + cost(s_1, a) \right\} \\
&= \min\left\{ cost(s_0, T) + cost(s_1, b) + cost(s_3, b) \right\}
\end{align*}
\]
Cost equations

A first attempt at finding costs

cost(s0, a) = 1
cost(s1, a) = 1
cost(s5, a) = 1
cost(s1, b) = 1
cost(s3, b) = 1

cost(s0, T) = min\{
  \text{cost(s0, a)} + \text{cost(s5, a)}
\} = 1

cost(s0, S) = min\{
  \text{cost(s0, T)} + \text{cost(s1, a)} + \text{cost(s3, b)}
\} = 2
Cost equations

A first attempt at finding costs

\[
\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1 \\
\text{cost}(s_0, T) &= \min \left\{ \text{cost}(s_0, a), \text{cost}(s_0, a) + \text{cost}(s_5, a) \right\} = 1 \\
\text{cost}(s_0, S) &= \min \left\{ \text{cost}(s_0, T) + \text{cost}(s_1, a), \text{cost}(s_0, T) + \text{cost}(s_1, b) + \text{cost}(s_3, b) \right\} = 2
\end{align*}
\]
Cost equations

A first attempt at finding costs

\[
\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1 \\
\text{cost}(s_0, T) &= \min \{ \text{cost}(s_0, a) + \text{cost}(s_5, a) \} \\
&= 1 \\
\text{cost}(s_0, S) &= \min \{ \text{cost}(s_0, T) + \text{cost}(s_1, a) + \text{cost}(s_3, b) \} \\
&= 2
\end{align*}
\]
Cost equations

A first attempt at finding costs

\begin{align*}
\text{cost}(s_0, a) &= 1 \\
\text{cost}(s_1, a) &= 1 \\
\text{cost}(s_5, a) &= 1 \\
\text{cost}(s_1, b) &= 1 \\
\text{cost}(s_3, b) &= 1 \\
\end{align*}

\begin{align*}
\text{cost}(s_0, T) &= \min \{ \text{cost}(s_0, a) + \text{cost}(s_5, a) \} \\
&= 1 \\
\text{cost}(s_0, S) &= \min \{ \text{cost}(s_0, T) + \text{cost}(s_1, a) + \text{cost}(s_3, b) \} \\
&= 2
\end{align*}
Cost equations

A first attempt at finding costs

\[
\begin{align*}
cost(s_0, a) &= 1 \\
cost(s_1, a) &= 1 \\
cost(s_5, a) &= 1 \\
cost(s_1, b) &= 1 \\
cost(s_3, b) &= 1 \\

cost(s_0, T) &= \min \left\{ \begin{array}{c} cost(s_0, a) \\ cost(s_0, a) + cost(s_5, a) \end{array} \right\} = 1 \\
cost(s_0, S) &= \min \left\{ \begin{array}{c} cost(s_0, T) + cost(s_1, a) \\ cost(s_0, T) + cost(s_1, b) + cost(s_3, b) \end{array} \right\} = 2
\end{align*}
\]
Cost matrices in the \((\text{min}, +)\) semiring

A single integer per edge is not sufficient to carry the cost information.

We use matrices indexed by terminals:
- the \textbf{row} index represents the \textit{lookahead token before} taking the transition
- the \textbf{column} index represents the \textit{lookahead token after} taking the transition
Cost matrices in the \((\text{min}, +)\) semiring

A single integer per edge is not sufficient to carry the cost information.

We use matrices indexed by terminals:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition
Cost matrices in the \((\min, +)\) semiring

A single integer per edge is not sufficient to carry the cost information.

We use matrices indexed by terminals:

- the row index represents the lookahead token before taking the transition
- the column index represents the lookahead token after taking the transition
Cost matrices in the \((\text{min, +})\) semiring

A single integer per edge is not sufficient to carry the cost information.

We use matrices indexed by terminals:

- the row index represents the lookahead token before taking the transition
- the column index represents the lookahead token after taking the transition
Cost matrices in the \((\text{min}, +)\) semiring

A single integer per edge is not sufficient to carry the cost information.

We use matrices indexed by terminals:

- the row index represents the lookahead token before taking the transition
- the column index represents the lookahead token after taking the transition
Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:
- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

In the $(\min, +)$ semiring, **matrix product** represents the cost of a **sequence**.

![Diagram](image)
LR(1) matrix-based cost equations

Computing costs with matrices:
LR(1) matrix-based cost equations

Computing costs with matrices:

\[ \text{cost}(s0, a) = \text{cost}(s1, a) = \text{cost}(s5, a) = \]
LR(1) matrix-based cost equations

Computing costs with matrices:

cost(s0, a) = cost(s1, a) = cost(s5, a) =

\[
\begin{pmatrix}
1 & 1 \\
\infty & \infty
\end{pmatrix}
\]
**LR(1) matrix-based cost equations**

Computing costs with matrices:

\[
\begin{align*}
\text{cost}(s_0, a) &= \text{cost}(s_1, a) = \text{cost}(s_5, a) = \\
\text{cost}(s_1, b) &= \text{cost}(s_3, b) = \\
\end{align*}
\]

\[
\begin{bmatrix}
1 & 1 \\
\infty & \infty \\
\end{bmatrix}
\]
LR(1) matrix-based cost equations

Computing costs with matrices:

\[
\begin{align*}
\text{cost}(s_0, a) &= \text{cost}(s_1, a) = \text{cost}(s_5, a) = \\
\text{cost}(s_1, b) &= \text{cost}(s_3, b) = \\
\text{cost}(s_0, T) &= \min \left\{ \begin{array}{l}
\text{cost}(s_0, a) \\
\text{cost}(s_0, a) \cdot \text{cost}(s_5, a)
\end{array} \right\} = \\
\end{align*}
\]
**LR(1) matrix-based cost equations**

Computing costs with matrices:

\[
\begin{align*}
\text{cost}(s_0, a) &= \text{cost}(s_1, a) = \text{cost}(s_5, a) = \\
\text{cost}(s_1, b) &= \text{cost}(s_3, b) = \\
\text{cost}(s_0, T) &= \min \left\{ \text{cost}(s_0, a), \text{cost}(s_0, a) \cdot \text{cost}(s_5, a) \right\} = \\
\text{cost}(s_0, S) &= \min \left\{ \text{cost}(s_0, T) \cdot \text{cost}(s_1, a), \text{cost}(s_0, T) \cdot \text{cost}(s_1, b) \cdot \text{cost}(s_3, b) \right\} =
\end{align*}
\]

\[
\begin{array}{c c}
\text{a} & \text{b} \\
\infty & \infty \\
\infty & \infty \\
\end{array}
\]

\[
\begin{array}{c c}
\text{a} & \text{b} \\
\infty & \infty \\
1 & 1 \\
\end{array}
\]

\[
\begin{array}{c c}
\text{a} & \text{b} \\
2 & 1 \end{array}
\]
Big matrices?

A $|T| \times |T|$ matrix for each transition and reduction step consumes a lot of space.
Big matrices?

A $|T| \times |T|$ matrix for each transition and reduction step consumes a lot of space.

It is often wasteful: lookahead terminals often behave the same. This leads to identical columns in the cost matrix.
Big matrices?

A $|T| \times |T|$ matrix for each transition and reduction step consumes a lot of space.

It is often wasteful: lookahead terminals often behave the same. This leads to **identical columns** in the cost matrix.

We can characterize and group lookahead tokens with identical behavior.
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  • \(T := a\) reduced when lookahead is \(b\) \(\rightarrow \{b\}\)
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  • \(T := a\) reduced when lookahead is \(b\) \(\rightarrow\) \(\{b\}\)
  • \(T := a\ a\) always reduced \(\rightarrow\) \(\{a,b\}\)
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  • \(T := a\) reduced when lookahead is \(b\) \(\rightarrow\) \(\{b\}\)
  • \(T := a\ a\) always reduced \(\rightarrow\) \(\{a,b\}\)

• The cases to consider are given by the **coarsest refinement** of \(\{a,b\}\) and \(\{b\}\):
  \(\{\{a\},\{b\}\}\)
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  - \(T := a\) reduced when lookahead is \(b\) \(\rightarrow\) \(\{b\}\)
  - \(T := a\ a\) always reduced \(\rightarrow\) \(\{a, b\}\)

• The cases to consider are given by the coarsest refinement of \(\{a, b\}\) and \(\{b\}\):
  \[
  \{\{a\},\{b\}\}\]
  \[
  \{T := a\ a\}\]
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  - \(T \ := \ a\) reduced when lookahead is \(b\) \(\rightarrow\) \{b\}
  - \(T \ := \ a\ a\) always reduced \(\rightarrow\) \{a,b\}

• The cases to consider are given by the coarsest refinement of \{a,b\} and \{b\}:
  \[
  \{\{a\},\{b\}\} \\
  \{T \ := \ a\ a\} \\
  \{T \ := \ a, T \ := \ a\ a\} 
  \]
Classifying terminals

• The goto transition \((s_0, T)\) is followed when one of its productions is reduced:
  • \(T := a\) reduced when lookahead is \(b\) \(\rightarrow\) \{\(b\)\}
  • \(T := a\ a\) always reduced \(\rightarrow\) \{\(a\), \(b\)\}

• The cases to consider are given by the coarsest refinement of \{\(a\), \(b\)\} and \{\(b\)\}:
  \[
  \{\{a\}\}, \{b\}\} \quad \{T := a\ a\} \quad \{T := a, T := a\ a\}
  \]

Before starting to compute costs, we know what lookahead symbols to distinguish!
Idea #2: compacting matrices
**Compacting columns**

Let’s assume that:

- we want to compact a matrix $m$
- we have 4 terminals, $a$, $b$, $c$ and $d$
- our characterization found the partition $\\{\{a,b\}, \{c\}, \{d\}\}$.

$$
\begin{array}{cccc}
  & a & b & c & d \\
 a & 1 & 1 & 2 & 3 \\
 b & \infty & \infty & \infty & \infty \\
 c & \infty & \infty & 3 & \infty \\
 d & 1 & 1 & 1 & 1 \\
\end{array}
$$
## Compacting columns

\[
m = \begin{array}{cccc}
a & b & c & d \\
a & 1 & 1 & 2 & 3 \\
b & \infty & \infty & \infty & \infty \\
c & \infty & \infty & 3 & \infty \\
d & 1 & 1 & 1 & 1 \\
\end{array}
\]
## Compacting columns

$$m = \begin{array}{cccc}
\text{a} & \text{b} & \text{c} & \text{d} \\
\hline
\text{a} & 1 & 1 & 2 & 3 \\
\text{b} & \infty & \infty & \infty & \infty \\
\text{c} & \infty & \infty & 3 & \infty \\
\text{d} & 1 & 1 & 1 & 1 \\
\end{array}$$
### Compacting columns

$$m = \begin{array}{cccc}
  & a & b & c & d \\
  a & 1 & 1 & 2 & 3 \\
  b & \infty & \infty & \infty & \infty \\
  c & \infty & \infty & 3 & \infty \\
  d & 1 & 1 & 1 & 1 \\
\end{array}$$
## Compacting columns

\[
m = \begin{bmatrix}
1 & 2 & 3 \\
\infty & \infty & \infty \\
\infty & 3 & \infty \\
1 & 1 & 1 \\
\end{bmatrix}
\]
## Compacting columns

\[
m = \begin{array}{c|ccc}
\{a, b\} & \{c\} & \{d\} \\
\hline
a & 1 & 2 & 3 \\
b & \infty & \infty & \infty \\
c & \infty & 3 & \infty \\
d & 1 & 1 & 1 \\
\end{array}
\]
Compacting rows

\[
\begin{array}{|c|c|c|}
\hline
{a, b} & \{c\} & \{d\} \\
\hline
a & m_{aa} & \ldots & \ldots \\
& m_{ab} & & \\
\hline
b & \ldots & \ldots & \ldots \\
\hline
c & \ldots & \ldots & \ldots \\
\hline
d & \ldots & \ldots & \ldots \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
a & b & c & d \\
\hline
a & n_{aa} & \ldots & \ldots & \ldots \\
\hline
b & n_{ba} & \ldots & \ldots & \ldots \\
\hline
c & n_{ca} & \ldots & \ldots & \ldots \\
\hline
d & n_{da} & \ldots & \ldots & \ldots \\
\hline
\end{array}
\]

\[= \mathbf{r}\]
## Compacting rows

\[
\begin{bmatrix}
\{a,b\} & \{c\} & \{d\} \\
\hline
a & m_{aa} & m_{ab} & \ldots & \ldots \\
\hline
b & \ldots & \ldots & \ldots \\
\hline
c & \ldots & \ldots & \ldots \\
\hline
d & \ldots & \ldots & \ldots \\
\end{bmatrix}
\begin{bmatrix}
a \\
\hline
b \\
\hline
c \\
\hline
d \\
\end{bmatrix}
\begin{bmatrix}
a & b & c & d \\
\hline
n_{aa} & \ldots & \ldots & \ldots \\
\hline
n_{ba} & \ldots & \ldots & \ldots \\
\hline
n_{ca} & \ldots & \ldots & \ldots \\
\hline
n_{da} & \ldots & \ldots & \ldots \\
\end{bmatrix}
= r
\]

\[r_{aa} = \]
Compacting rows

\[
\begin{align*}
\{a, b\} & \quad \{c\} & \quad \{d\} \\
\begin{array}{ccc}
 a & m_{aa} & \ldots \\
 b & \ldots & \ldots \\
 c & \ldots & \ldots \\
 d & \ldots & \ldots \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccc}
 a & b & c & d \\
 n_{aa} & \ldots & \ldots & \ldots \\
 n_{ba} & \ldots & \ldots & \ldots \\
 n_{ca} & \ldots & \ldots & \ldots \\
 n_{da} & \ldots & \ldots & \ldots \\
\end{array}
\end{align*}
\]

\[
= \quad r
\]

\[
r_{aa} = (m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]

\[
x \land y = \min\{x, y\}
\]
Compacting rows

\[ \{a, b\} \{c\} \{d\} \]

\[
\begin{array}{ccc}
a & m_{aa} & \cdots & \cdots \\
m_{ab} & \cdots & \cdots & \cdots \\
b & \cdots & \cdots & \cdots \\
c & \cdots & \cdots & \cdots \\
d & \cdots & \cdots & \cdots \\
\end{array}
\]

\[ \cdot \]

\[
\begin{array}{cccc}
a & n_{aa} & \cdots & \cdots \\
b & n_{ba} & \cdots & \cdots \\
c & n_{ca} & \cdots & \cdots \\
d & n_{da} & \cdots & \cdots \\
\end{array}
\]

\[ = r \]

\[
r_{aa} = (m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]
## Compacting rows

\[
\begin{array}{ccc}
\{a,b\} & \{c\} & \{d\} \\
\hline
a & m_{aa} & ... & ... \\
   & m_{ab} &   &   \\
b & ... & ... & ... \\
c & ... & ... & ... \\
d & ... & ... & ... \\
\end{array}
\quad \cdot \quad
\begin{array}{cccc}
a & n_{aa} & ... & ... & ... \\
b & n_{ba} & ... & ... & ... \\
c & n_{ca} & ... & ... & ... \\
d & n_{da} & ... & ... & ... \\
\end{array}
\quad = \quad r
\]

\[
r_{aa} = (m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]

\[
(m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) = m_{aa} + (n_{aa} \land n_{ba})
\]
Compacting rows

\[ r_{aa} = (m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da}) \]

\[ r_{aa} = (m_{aa} + n_{aa} \land n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da}) \]
Compacting rows

\[
\begin{array}{cccc}
\{a,b\} & \{c\} & \{d\} \\
\hline
a & m_{aa} & \ldots & \ldots \\
 & m_{ab} & \ldots & \ldots \\
b & \ldots & \ldots & \ldots \\
 & \ldots & \ldots & \ldots \\
c & \ldots & \ldots & \ldots \\
 & \ldots & \ldots & \ldots \\
d & \ldots & \ldots & \ldots \\
\end{array}
\]  
\[\cdot\]
\[
\begin{array}{cccc}
a & n_{aa} & \ldots & \ldots \\
b & n_{ba} & \ldots & \ldots \\
c & n_{ca} & \ldots & \ldots \\
d & n_{da} & \ldots & \ldots \\
\end{array}
\]
\[
= r
\]

\[
r_{aa} = (m_{aa} + n_{aa}) \land (m_{ab} + n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]

\[
r_{aa} = (m_{aa} + [n_{aa} \land n_{ba}]) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]
Compacting rows

\[ r_{aa} = (m_{aa} + n_{aa} \land n_{ba}) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da}) \]
Compacting rows

\[
\begin{array}{c|c|c|c|c}
\{a,b\} & \{c\} & \{d\} \\
\hline
a & m_{aa} & \ldots & \ldots \\
   & m_{ab} & \ldots & \ldots \\
\hline
b & \ldots & \ldots & \ldots \\
\hline
c & \ldots & \ldots & \ldots \\
\hline
d & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\{a,b\} & \{c\} & \{d\} \\
\hline
a & n_{aa} & \ldots & \ldots & \ldots \\
   & \^ & \^ & \^ & \^ \\
\hline
b & \ldots & \ldots & \ldots \\
\hline
c & n_{ca} & \ldots & \ldots & \ldots \\
\hline
d & n_{da} & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
r_{aa} = (m_{aa} + (n_{aa} \land n_{ba})) \land (m_{ac} + n_{ca}) \land (m_{ad} + n_{da})
\]
Compacting rows

\[
\begin{array}{ccc}
\{a, b\} & \{c\} & \{d\} \\
\hline
a & m_{aa} & \ldots & \ldots \\
& m_{ab} & \ldots & \ldots \\
b & \ldots & \ldots & \ldots \\
c & \ldots & \ldots & \ldots \\
d & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{cccc}
a & b & c & d \\
\hline
\{a, b\} & n_{aa} & \ldots & \ldots & \ldots \\
& \wedge & \wedge & \wedge & \wedge \\
\{c\} & n_{ca} & \ldots & \ldots & \ldots \\
\{d\} & n_{da} & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{ccc}
\cdots & \ldots & \ldots \\
\cdots & \ldots & \ldots \\
\cdots & \ldots & \ldots \\
\cdots & \ldots & \ldots \\
\end{array}
\]

\[
r_{aa} = (m_{aa} + n_{aa} \wedge n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})
\]
Compacting rows

On average, compaction of rows and columns reduces space consumption of matrices by 5 orders of magnitude!
In practice...
In practice...

- Grammars are recursive objects. We need to solve mutually recursive and monotonous equations.
In practice...

- Grammars are **recursive** objects. We need to solve **mutually recursive** and monotonous equations.
- Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed.
In practice...

• Grammars are **recursive** objects. We need to solve **mutually recursive** and monotonous equations.
• Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed
• Re-ordering of matrix product chains
• Maximal sharing of intermediate matrices
• ...

In practice...

- Grammars are **recursive** objects. We need to solve **mutually recursive** and monotonous equations.
- Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed.
- Re-ordering of matrix product chains
- Maximal sharing of intermediate matrices
- ...

You will find these explained in our paper!
Conclusion
A new algorithm that provides a significant speed-up to LR(1) reachability by:

• Reframing the problem. Solving a set of mutually recursive equations on matrices.
• Compacting matrices in a sound way.

The implementation gives good results and is available in the current release of Menhir.