An Agda proof of the correctness of Valiant's algorithm for CF parsing

Thomas Bååth Sjöblom

Chalmers University of Technology

- Valiant's algorithm

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.

Motivation

- Valiant's algorithm
    - 1974: Prove that parsing can be done as quickly as matrix multiplication.
    - 2013: Provide a practical (?) algorithm for parsing in parallel.

- Valiant's algorithm
    - 1974: Prove that parsing can be done as quickly as matrix multiplication.
    - 2013: Provide a practical (?) algorithm for parsing in parallel.
    - Can be derived from matrix algebra.

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof
  - Bug free code is important!

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof
  - Bug free code is important!
  - Valiant's algorithm is complicated enough to need it.

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof
  - Bug free code is important!
  - Valiant's algorithm is complicated enough to need it.
- Agda

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof
  - Bug free code is important!
  - Valiant's algorithm is complicated enough to need it.
- Agda
  - Familiar if you know Haskell.

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as quickly as matrix multiplication.
  - 2013: Provide a practical (?) algorithm for parsing in parallel.
  - Can be derived from matrix algebra.
- Formal proof
  - Bug free code is important!
  - Valiant's algorithm is complicated enough to need it.
- Agda
  - Familiar if you know Haskell.
  - Not much math in it (yet!).

Analysing the structure of a string

Analysing the structure of a string

"She eats a fish with a fork"

Analysing the structure of a string

"She eats a fish with a fork"

Grammar:

Analysing the structure of a string

"She eats a fish with a fork"

Grammar:

$$\Sigma = \{\text{She}, \text{eats}, \text{a}, \text{fish}, \text{with}, \text{fork}\}$$

Analysing the structure of a string

"She eats a fish with a fork"

Grammar:

$\Sigma = \{\text{She}, \text{eats}, \text{a}, \text{fish}, \text{with}, \text{fork}\}$

$N = \{S, N, N_p, V, V_p, D, P, P_p\}$

Analysing the structure of a string

$$\text{``She eats a fish with a fork''}$$

Grammar:

$$\Sigma = \{\text{She}, \text{eats}, \text{a}, \text{fish}, \text{with}, \text{fork}\}$$
$$N = \{S, N, N_p, V, V_p, D, P, P_p\}$$
$$P = \left\{ \begin{array}{l|l|l} S \to N_p V_p & P_p \to P N_p & N_p \to \text{She} \\ V_p \to V_p P_p & N_p \to DN & N \to \text{fish} \\ V_p \to V N_p & V \to \text{eats} & N \to \text{fork} \\ V_p \to \text{eats} & P \to \text{with} & D \to \text{a} \end{array} \right\}$$

Analysing the structure of a string

"She eats a fish with a fork"

Grammar:

$$\Sigma = \{\text{She}, \text{eats}, \text{a}, \text{fish}, \text{with}, \text{fork}\}$$

$$N = \{S, N, N_p, V, V_p, D, P, P_p\}$$

$$P = \left\{ \begin{array}{l|l|l}
S \rightarrow N_p V_p & P_p \rightarrow P N_p & N_p \rightarrow \text{She} \\
V_p \rightarrow V_p P_p & N_p \rightarrow DN & N \rightarrow \text{fish} \\
V_p \rightarrow V N_p & V \rightarrow \text{eats} & N \rightarrow \text{fork} \\
V_p \rightarrow \text{eats} & P \rightarrow \text{with} & D \rightarrow \text{a}
\end{array} \right\}$$

$$S = S$$

Parsing

"She eats a fish with a fork"

Parsing

"She eats a fish with a fork"

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \to$ She, $V \to$ eats, $V_p \to$ eats, $D \to$ a, etc. Defines C.

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \rightarrow$ She, $V \rightarrow$ eats, $V_p \rightarrow$ eats, $D \rightarrow$ a, etc. Defines C.

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \rightarrow$ She, $V \rightarrow$ eats, $V_p \rightarrow$ eats, $D \rightarrow$ a, etc. Defines C.

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \rightarrow$ She,
    $V \rightarrow$ eats, $V_p \rightarrow$ eats,
    $D \rightarrow$ a, etc. Defines C.
- The rest:

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \rightarrow$ She, $V \rightarrow$ eats, $V_p \rightarrow$ eats, $D \rightarrow$ a, etc. Defines C.
- The rest:
  - $x \cdot y = \{A \mid B \in x, C \in y, A \rightarrow BC \in P\}$

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \to$ She, $V \to$ eats, $V_p \to$ eats, $D \to$ a, etc. Defines C.
- The rest:
  - $x \cdot y = \{A \mid B \in x, C \in y, A \to BC \in P\}$

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & ? & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \to$ She, $V \to$ eats, $V_p \to$ eats, $D \to$ a, etc. Defines C.
- The rest:
  - $x \cdot y = \{A \mid B \in x, C \in y, A \to BC \in P\}$

$$
\begin{pmatrix}
0 & ? & ? & ? & ? & ? & ? & ? \\
  & 0 & ? & ? & ? & ? & ? & ? \\
  &   & 0 & ? & ? & ? & ? & ? \\
  &   &   & 0 & ? & ? & ? & ? \\
  &   &   &   & 0 & ? & ? & ? \\
  &   &   &   &   & 0 & ? & ? \\
  &   &   &   &   &   & 0 & ? \\
  &   &   &   &   &   &   & 0
\end{pmatrix}
$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
    - Fill with $N_p \to$ She,
      $V \to$ eats, $V_p \to$ eats,
      $D \to$ a, etc. Defines C.
- The rest:
    - $x \cdot y = \{A \mid B \in x, C \in y,$
      $A \to BC \in P\}$

$$
\begin{pmatrix}
0 & ? & ? & ? & ? & ? & ? & ? \\
  & 0 & ? & ? & ? & ? & ? & ? \\
  &   & 0 & ? & ? & ? & ? & ? \\
  &   &   & 0 & ? & ? & ? & ? \\
  &   &   &   & 0 & ? & ? & ? \\
  &   &   &   &   & 0 & ? & ? \\
  &   &   &   &   &   & 0 & ? \\
  &   &   &   &   &   &   & 0
\end{pmatrix}
$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \rightarrow$ She, $V \rightarrow$ eats, $V_p \rightarrow$ eats, $D \rightarrow$ a, etc. Defines C.
- The rest:
  - $x \cdot y = \{A \mid B \in x, C \in y, A \rightarrow BC \in P\}$
  - $X_{ij} = \sum_k X_{ik} \cdot X_{kj}$

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & ? & ? & \textcolor{red}{?} & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & ? & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonal:
  - Fill with $N_p \to$ She, $V \to$ eats, $V_p \to$ eats, $D \to$ a, etc. Defines C.
- The rest:
  - $x \cdot y = \{A \mid B \in x, C \in y, A \to BC \in P\}$
  - $X_{ij} = \sum_k X_{ik} \cdot X_{kj}$
- $X = XX + C$

$$
\begin{pmatrix}
0 & ? & ? & ? & ? & ? & ? & ? \\
  & 0 & ? & ? & ? & ? & ? & ? \\
  &   & 0 & ? & ? & ? & ? & ? \\
  &   &   & 0 & ? & ? & ? & ? \\
  &   &   &   & 0 & ? & ? & ? \\
  &   &   &   &   & 0 & ? & ? \\
  &   &   &   &   &   & 0 & ? \\
  &   &   &   &   &   &   & 0
\end{pmatrix}
$$

Save substring parses in X.

$$X = XX + C$$

C upper triangular. Find X.

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.
- C a $2^n \times 2^n$ matrix:

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.
- C a $2^n \times 2^n$ matrix:
  1. Split: $C = \begin{pmatrix} C_U & C_R \\ & C_L \end{pmatrix}$, $X = \begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix}$

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.
- C a $2^n \times 2^n$ matrix:
    1. Split: $C = \begin{pmatrix} C_U & C_R \\ & C_L \end{pmatrix}$, $X = \begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix}$
    2. $\begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix} = \begin{pmatrix} X_U X_U + C_U & X_U X_R + X_R X_L + C_R \\ & X_L X_L + C_L \end{pmatrix}$

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.
- C a $2^n \times 2^n$ matrix:
    1. Split: $C = \begin{pmatrix} C_U & C_R \\ & C_L \end{pmatrix}$, $X = \begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix}$
    2. $\begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix} = \begin{pmatrix} X_U X_U + C_U & X_U X_R + X_R X_L + C_R \\ & X_L X_L + C_L \end{pmatrix}$
    3. Recursively compute X_U, X_L.

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

- C a 1×1 matrix $\implies C = (0)$. $X = (0)$.
- C a $2^n \times 2^n$ matrix:
    1. Split: $C = \begin{pmatrix} C_U & C_R \\ & C_L \end{pmatrix}$, $X = \begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix}$
    2. $\begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix} = \begin{pmatrix} X_U X_U + C_U & X_U X_R + X_R X_L + C_R \\ & X_L X_L + C_L \end{pmatrix}$
    3. Recursively compute X_U, X_L.
    4. Find X_R: $X_R = X_U X_R + X_R X_L + C_R$.

$$X_R = X_U X_R + X_R X_L + C_R$$

Valiant's Algorithm

$$X_R = X_U X_R + X_R X_L + C_R$$

- C_R a 1×1 matrix $\implies X_U = X_L = (0)$. $X_R = C_R$.

Valiant's Algorithm

$$X_R = X_U X_R + X_R X_L + C_R$$

- C_R a 1×1 matrix $\implies X_U = X_L = (0)$. $X_R = C_R$.
- C_R a $2^n \times 2^n$ matrix:

$$X_R = X_U X_R + X_R X_L + C_R$$

- C_R a 1×1 matrix $\implies X_U = X_L = (0)$. $X_R = C_R$.
- C_R a $2^n \times 2^n$ matrix:
  - Split $C_R = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$, $X_R = \begin{pmatrix} X_1 & X_2 \\ X_3 & X_4 \end{pmatrix}$,
    $X_U = \begin{pmatrix} X_{U_U} & X_{U_R} \\ & X_{U_L} \end{pmatrix}$ and $X_L = \begin{pmatrix} X_{L_U} & X_{L_R} \\ & X_{L_L} \end{pmatrix}$

Valiant's Algorithm

$$X_R = X_U X_R + X_R X_L + C_R$$

- C_R a 1×1 matrix $\implies X_U = X_L = (0)$. $X_R = C_R$.
- C_R a $2^n \times 2^n$ matrix:
  - Split $C_R = \begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix}$, $X_R = \begin{pmatrix} X_1 & X_2 \\ X_3 & X_4 \end{pmatrix}$,

    $X_U = \begin{pmatrix} X_{U_U} & X_{U_R} \\ & X_{U_L} \end{pmatrix}$ and $X_L = \begin{pmatrix} X_{L_U} & X_{L_R} \\ & X_{L_L} \end{pmatrix}$
  - Multiplying together gives
    $$X_1 = X_{U_U} X_1 + X_1 X_{L_U} + X_{U_R} X_3 + C_1$$
    $$X_2 = X_{U_U} X_2 + X_2 X_{L_L} + X_{U_R} X_4 + X_1 X_{L_R} + C_2$$
    $$X_3 = X_{U_L} X_3 + X_3 X_{L_U} + C_3$$
    $$X_4 = X_{U_L} X_4 + X_4 X_{L_L} + X_3 X_{L_R} + C_4$$

Agda

Need:

Agda

Need:
- Matrix datatype

Need:

- Matrix datatype
- Upper triangular matrix datatype

Need:

- Matrix datatype
- Upper triangular matrix datatype
- Addition and multiplication

Need:

- Matrix datatype
- Upper triangular matrix datatype
- Addition and multiplication
- Equality

```
data Mat : ℕ → Set where
  sing : R → Mat 0
  quad : ∀ { n } → Mat n → Mat n
                 → Mat n → Mat n
                 → Mat (suc n)
```

Agda

```
data Mat : ℕ → Set where
  sing  : R → Mat 0
  quad  : ∀ { n } → Mat n → Mat n
                  → Mat n → Mat n
                  → Mat (suc n)


data Tri : ℕ → Set where
  zer : Tri 0
  tri : ∀ { n } → Tri n → Mat n
                → Tri n
                → Tri (suc n)
```

Agda

```
_+_ : ∀ {n} → Mat n → Mat n → Mat n
sing x        + sing x'            = sing (x +ᴿ x')
quad A B C D + quad A' B' C' D' =
  quad (A + A') (B + B')
       (C + C') (D + D')
```

Agda

```
_+_ : ∀ {n} → Mat n → Mat n → Mat n
sing x        + sing x'              = sing (x +_R x')
quad A B C D + quad A' B' C' D' =
  quad (A + A') (B + B')
       (C + C') (D + D')


_*_ : ∀ {n} → Mat n → Mat n → Mat n
sing x        * sing x'              = sing (x *_R x')
quad A B C D * quad A' B' C' D' =
  quad (A * A' + B * C') (A * B' + B * D')
       (C * A' + D * C') (C * B' + D * D')
```

Agda

```
overlap : ∀ { n } → Tri n → Mat n → Tri n → Mat n
overlap zer (sing x) zer                      = sing x
overlap (tri U₁' R₁' L₁') (quad A B C D)
                     (tri U₂' R₂' L₂') = quad A' B'
                                              C' D'
  where
    C' = overlap L₁' C U₂'
    A' = overlap U₁' (A + R₁' * C') U₂'
    D' = overlap L₁' (D + C' * R₂') L₂'
    B' = overlap U₁' (B + R₁' * D' + A' * R₂') L₂'
```

Agda

```
valiant : ∀ {n} → Tri n → Tri n
valiant zer          = zer
valiant (tri U R L) = tri U⁺ (overlap U⁺ R L⁺) L⁺
  where U⁺ = valiant U
        L⁺ = valiant L
```

Agda

- Equality:

$$\_\approx\_ : \forall \{n\} \to \text{Tri } n \to \text{Tri } n \to \text{Set}$$
$$\text{zer} \approx \text{zer} = \text{tt}$$
$$(\text{tri } U \; R \; L) \approx (\text{tri } U' \; R' \; L') = U \approx U', R \approx R'$$
$$, L \approx L'$$

Agda

- Equality:

$$\_\approx\_ : \forall \{n\} \to \text{Tri } n \to \text{Tri } n \to \text{Set}$$
$$\text{zer} \approx \text{zer} = \text{tt}$$
$$(\text{tri U R L}) \approx (\text{tri U' R' L'}) = U \approx U', R \approx R'$$
$$, L \approx L'$$

- To prove the correctness, find an element of type

$$\forall \{n\} \{C : \text{Tri } n\} \to$$
$$(\text{valiant C}) \approx (\text{valiant C}) * (\text{valiant C}) + C$$

Sketch of proof:

- We pattern match on the Tri.
    - zer, should have type

        $$zer \approx zer * zer + zer$$

    - tri, we get that the overlap function should satisfy

        $$R' \approx U * R' + R' * L + R,$$

        but this was the specification we used to derive it.

Thank you!

Thank you!