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

Thomas Bååth Sjöblom

Chalmers University of Technology

Motivation

Motivation

- Valiant's algorithm

Motivation

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

Motivation

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

Motivation

- Valiant's algorithm
  - 1974: Prove that parsing can be done as fast 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 fast 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 fast 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 fast 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 fast 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 fast 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 fast 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

"She eats a fish with a fork"

Grammar:

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

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

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

Analysing the structure of a string

"She eats a fish with a fork"

Grammar:

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

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

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

$$S = S$$

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"

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

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"

- Superdiagonals:
  - 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"

- Superdiagonals:
  - 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"

- Superdiagonals:
  - 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"

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

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

Save substring parses in X.

Parsing

"She eats a fish with a fork"

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

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

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonals:
  - 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"

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

$$\begin{pmatrix} 0 & ? & ? & ? & ? & ? & ? & ? \\ & 0 & ? & ? & ? & ? & ? & ? \\ & & 0 & ? & {\color{blue}?} & ? & {\color{red}?} & ? \\ & & & 0 & ? & ? & ? & ? \\ & & & & 0 & ? & {\color{green}?} & ? \\ & & & & & 0 & ? & ? \\ & & & & & & 0 & ? \\ & & & & & & & 0 \end{pmatrix}$$

Save substring parses in X.

Parsing

"She eats a fish with a fork"

- Superdiagonals:
  - 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"

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

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

Save substring parses in X.

Parsing

"She eats a fish with a fork"

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

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

Save substring parses in X.

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

Valiant's Algorithm

$$X = XX + C$$

C upper triangular. Find X.

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

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:

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: $C = \begin{pmatrix} C_U & C_R \\ & C_L \end{pmatrix}$, $X = \begin{pmatrix} X_U & X_R \\ & X_L \end{pmatrix}$

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: $X_R = X_U X_R + X_R X_L + C_R$.

Valiant's Algorithm

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

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:

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 $X_L = \begin{pmatrix} X_{L_U} & X_{L_R} \\ & X_{L_L} \end{pmatrix}$

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

$$\begin{aligned}
X_1 &= X_{U_U} X_1 + X_1 X_{L_U} + X_{U_R} X_3 + C_1 \\
X_2 &= X_{U_U} X_2 + X_2 X_{L_L} + X_{U_R} X_4 + X_1 X_{L_R} + C_2 \\
X_3 &= X_{U_L} X_3 + X_3 X_{L_U} + C_3 \\
X_4 &= X_{U_L} X_4 + X_4 X_{L_L} + X_3 X_{L_R} + C_4
\end{aligned}$$

Agda

Need:

Agda

Need:

- Matrix datatype

Agda

Need:

- Matrix datatype
- Upper triangular matrix datatype

Agda

Need:

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

Agda

Need:

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

Agda

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Mat}\ :\ \mathbb{N} \to \mathsf{Set}\ \textbf{where} \\
&\quad \mathsf{sing}\ :\ \mathsf{R} \to \mathsf{Mat}\ 0 \\
&\quad \mathsf{quad}\ :\ \forall\ \{\mathsf{n}\} \to \mathsf{Mat}\ \mathsf{n} \to \mathsf{Mat}\ \mathsf{n} \\
&\qquad\qquad\qquad\quad \to \mathsf{Mat}\ \mathsf{n} \to \mathsf{Mat}\ \mathsf{n} \\
&\qquad\qquad\qquad\quad \to \mathsf{Mat}\ (\mathsf{suc}\ \mathsf{n})
\end{aligned}
$$

Agda

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Mat} : \mathbb{N} \to \mathsf{Set}\ \textbf{where}\\
&\quad \mathsf{sing} : \mathsf{R} \to \mathsf{Mat}\ 0\\
&\quad \mathsf{quad} : \forall \{ \mathsf{n} \} \to \mathsf{Mat}\ \mathsf{n} \to \mathsf{Mat}\ \mathsf{n}\\
&\qquad\qquad\qquad \to \mathsf{Mat}\ \mathsf{n} \to \mathsf{Mat}\ \mathsf{n}\\
&\qquad\qquad\qquad \to \mathsf{Mat}\ (\mathsf{suc}\ \mathsf{n})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Tri} : \mathbb{N} \to \mathsf{Set}\ \textbf{where}\\
&\quad \mathsf{zer} : \mathsf{Tri}\ 1\\
&\quad \mathsf{tri} : \forall \{ \mathsf{n} \} \to \mathsf{Tri}\ \mathsf{n} \to \mathsf{Mat}\ \mathsf{n}\\
&\qquad\qquad\qquad\qquad\quad \to \mathsf{Tri}\ \mathsf{n}\\
&\qquad\qquad\qquad\qquad\quad \to \mathsf{Tri}\ (\mathsf{suc}\ \mathsf{n})
\end{aligned}
$$

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

$$
\begin{array}{l}
\text{overlap} : \forall \{n\} \to \text{Tri } n \to \text{Mat } n \to \text{Tri } n \to \text{Mat } n \\
\text{overlap zer (sing x) zer} \qquad\qquad\qquad\qquad = \text{sing x} \\
\text{overlap (tri } U_1\text{' } R_1\text{' } L_1\text{') (quad A B C D)} \\
\qquad\qquad\qquad\qquad \text{(tri } U_2\text{' } R_2\text{' } L_2\text{')} = \text{quad A' B'} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{C' D'}
\end{array}
$$

where

$$
\begin{aligned}
\text{C'} &= \text{overlap } L_1\text{' C } U_2\text{'} \\
\text{A'} &= \text{overlap } U_1\text{' (A} + R_1\text{'} * \text{C') } U_2\text{'} \\
\text{D'} &= \text{overlap } L_1\text{' (D} + \text{C'} * R_2\text{') } L_2\text{'} \\
\text{B'} &= \text{overlap } U_1\text{' (B} + R_1\text{'} * \text{D'} + \text{A'} * R_2\text{') } L_2\text{'}
\end{aligned}
$$

Agda

$$
\begin{array}{lcl}
\mathsf{valiant} \ : \ \forall \{ \mathsf{n} \} \rightarrow \mathsf{Tri}\ \mathsf{n} \rightarrow \mathsf{Tri}\ \mathsf{n} & & \\
\mathsf{valiant}\ \mathsf{zer} & = & \mathsf{zer} \\
\mathsf{valiant}\ (\mathsf{tri}\ \mathsf{U}\ \mathsf{R}\ \mathsf{L}) & = & \mathsf{tri}\ \mathsf{U}^{+}\ (\mathsf{overlap}\ \mathsf{U}^{+}\ \mathsf{R}\ \mathsf{L}^{+})\ \mathsf{L}^{+} \\
\quad \mathbf{where}\ \mathsf{U}^{+} \ = \ \mathsf{valiant}\ \mathsf{U} & & \\
\qquad\qquad\quad \mathsf{L}^{+} \ = \ \mathsf{valiant}\ \mathsf{L} & &
\end{array}
$$

Agda

- Equality:

$$
\begin{aligned}
&\_\approx\_ \ : \forall \{ n \} \rightarrow \mathsf{Tri}\ n \rightarrow \mathsf{Tri}\ n \rightarrow \mathsf{Set} \\
&\mathsf{zer} \approx \mathsf{zer} \ = \ \mathsf{tt} \\
&(\mathsf{tri}\ U\ R\ L) \approx (\mathsf{tri}\ U'\ R'\ L') \ = \ U \approx U', R \approx R' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad , L \approx L'
\end{aligned}
$$

Agda

- Equality:

$$
\begin{aligned}
&\_\approx\_ \; : \forall \{ n \} \to \mathsf{Tri}\ n \to \mathsf{Tri}\ n \to \mathsf{Set} \\
&\mathsf{zer} \approx \mathsf{zer} \;=\; \mathsf{tt} \\
&(\mathsf{tri}\ U\ R\ L) \approx (\mathsf{tri}\ U'\ R'\ L') \;=\; U \approx U', R \approx R' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad , L \approx L'
\end{aligned}
$$

- To prove the correctness, find an element of type

$$
\begin{aligned}
&\forall \{ n \} \{ C : \mathsf{Tri}\ n \} \to \\
&\quad (\mathsf{valiant}\ C) \approx (\mathsf{valiant}\ C) * (\mathsf{valiant}\ C) + C
\end{aligned}
$$

Agda

Sketch of proof:

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

    $$\text{zer} \approx \text{zer} * \text{zer} + \text{zer}$$

  - tri, we get that the overlap function should satisfy

    $$\text{R'} \approx \text{U} * \text{R'} + \text{R'} * \text{L} + \text{R},$$

    but this was the specification we used to derive it.

Thank you!

Thank you!