Nested Word Automata

Jens Stimpfle

30.6.2014

Nested Words

Nested Words

- Theoretically and practically pleasant model for the representation of data with both:
  - a **linear ordering**
  - a **hierarchically nested matching**

Nested Words

- Theoretically and practically pleasant model for the representation of data with both:
  - a **linear ordering**
  - a **hierarchically nested matching**
- Applications in software verification and document processing

Nested Words

- Theoretically and practically pleasant model for the representation of data with both:
  - a **linear ordering**
  - a **hierarchically nested matching**
- Applications in software verification and document processing
- This is the last list item

Structure of this talk

1. Motivation
2. Nested words
3. Nested word automata

Section 1

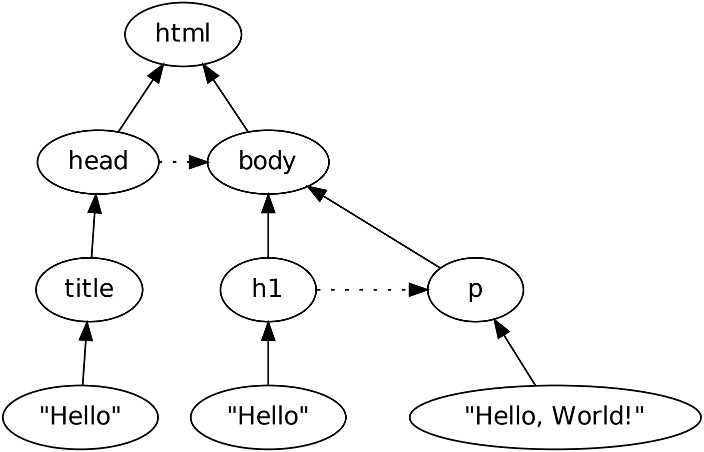
Motivation

Subsection 1

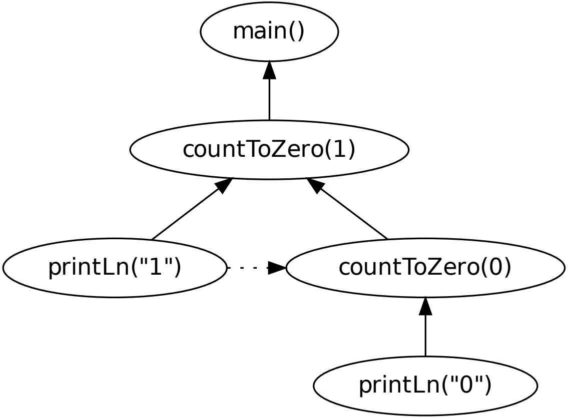
Data with both linear ordering and hierarchically nested matching

1. Document trees (e.g. HTML)
2. Executions of structured programs (with call-return semantics)

Document trees (e.g. HTML)



Executions of structured programs (with call-return semantics)



Subsection 2

Formal Languages

- Regular Languages
- Context-Free Languages

Regular Languages

Regular language over an alphabet Σ

- Most easily explained as generated by a regular expression (RE)
  - Example RE: `0|[123456789][0123456789]*`

Regular Languages

Regular language over an alphabet Σ

- Most easily explained as generated by a regular expression (RE)
  - Example RE: `0|[123456789][0123456789]*`
  - Typical implementation: DFA (Deterministic Finite Automaton)

“Problems” with Regular Languages

- Can’t express arbitrarily deep nesting

Context-free Languages

Context-free language over Σ

- Superset of Regular Languages

Context-free Languages

Context-free language over Σ

- Superset of Regular Languages
- Most easily explained as generated by a Context-free Grammar (CFG)
  - terminal symbols Σ and non-terminal symbols V
  - start symbol $S \in V$
  - Productions $\subset V \times (V \cup \Sigma)^*$

Context-free Languages

Context-free language over Σ

- Superset of Regular Languages
- Most easily explained as generated by a Context-free Grammar (CFG)
  - terminal symbols Σ and non-terminal symbols V
  - start symbol $S \in V$
  - Productions $\subset V \times (V \cup \Sigma)^*$
  - Example for real world usage:

```
HTML : "<html>" BODY "</html>"
BODY : "<body>" CONTENT "</html>"
CONTENT : "Hello, world!" | "Hallo, Welt!"
```

Context-free Languages

Context-free language over Σ

- Superset of Regular Languages
- Most easily explained as generated by a Context-free Grammar (CFG)
  - terminal symbols Σ and non-terminal symbols V
  - start symbol $S \in V$
  - Productions $\subset V \times (V \cup \Sigma)^*$
  - Example for real world usage:

```
HTML : "<html>" BODY "</html>"
BODY : "<body>" CONTENT "</html>"
CONTENT : "Hello, world!" | "Hallo, Welt!"
```

- Typical implementation: Pushdown Automaton

“Problems” with Context-free Languages

- Not closed under intersection
- Not closed under complementation
- Not closed under difference

“Problems” with Context-free Languages

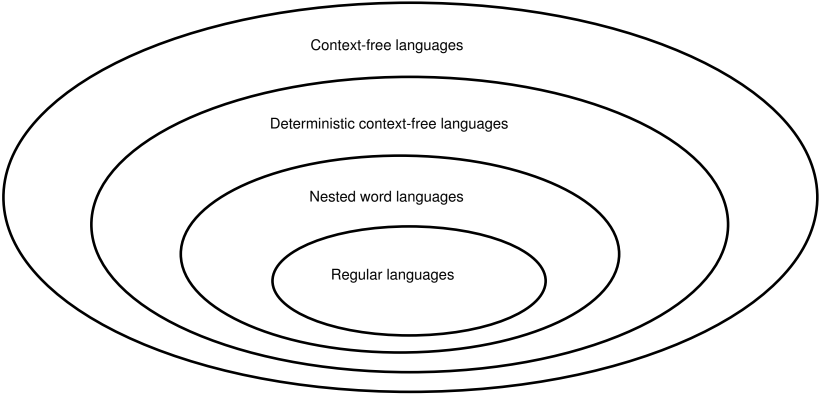
- Not closed under intersection
- Not closed under complementation
- Not closed under difference
- Can’t decide inclusion
- Can’t decide equivalence

“Problems” with Context-free Languages

- Not closed under intersection
- Not closed under complementation
- Not closed under difference
- Can’t decide inclusion
- Can’t decide equivalence
- Not determinizable (Deterministic Context-free languages are a strict subset of Context-free languages)

Nested words

- Nested words were constructed to overcome the limitations of Context-free and Regular languages
- The class of nested word languages lies properly between deterministic context-free languages and Regular languages



Section 2

Nested words

Nested words are ordinary words with extra information:

The nesting structure is explicitly contained in the input.

$\Rightarrow$ automata for nested words need not parse the nesting.

Definition: Nested word

- Later!
- For now: *well-matched* nested words

Definition: Well-matched nested word

A *well-matched nested word* over an alphabet Σ is a **pair** $(a_1 \ldots a_n, \rightsquigarrow)$

Definition: Well-matched nested word

A *well-matched nested word* over an alphabet Σ is a **pair** $(a_1 \dots a_n, \rightsquigarrow)$

- $a_1 \dots a_n \in \Sigma^*$ is a word over Σ

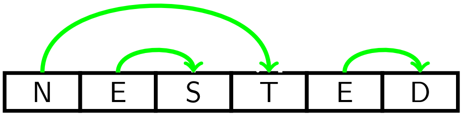
Definition: Well-matched nested word

A *well-matched nested word* over an alphabet Σ is a **pair** $(a_1 \dots a_n, \rightsquigarrow)$

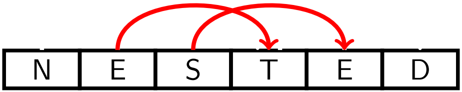
- $a_1 \dots a_n \in \Sigma^*$ is a word over Σ
- The *matching* $\rightsquigarrow$ matches "start tags" with their "end tags":
  - $\rightsquigarrow \subset [1..n] \times [1..n]$
  - Given $(i,j) \neq (k,l)$ elements of $\rightsquigarrow$, **either** $i < j < k < l$ **or** $i < k < l < j$

For $(i,j) \in \rightsquigarrow$, i is a *call position* and j is a *return position*

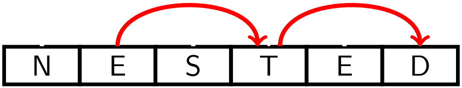
Well-matched



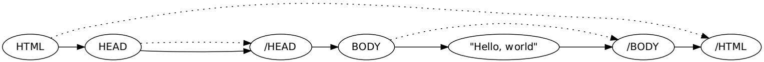
Not well-matched



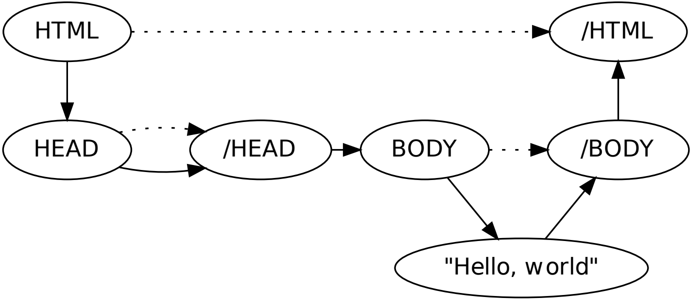
Not well-matched



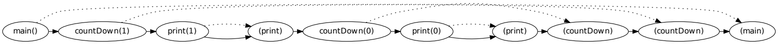
Example: Simple HTML tree



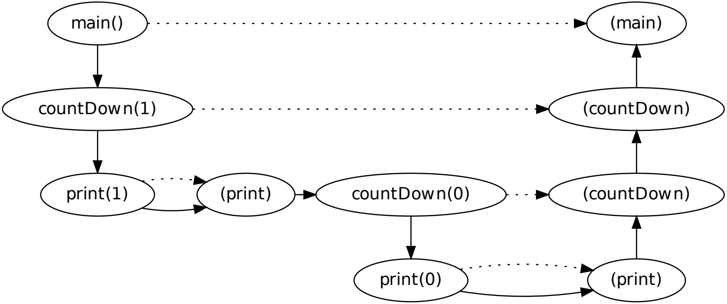
Example: Simple HTML tree



Example: Process trace



Example: Process trace



Section 3

Nested Word Automata (NWA)

A Nested Word Automaton takes a nested word as input and (as automatons do) accepts or rejects it.

A Nested Word Automaton takes a nested word as input and (as automatons do) accepts or rejects it.

Nested word automata have much of the power of Pushdown Automata, but can take advantage of the fact that their inputs carry a “pre-parsed” hierarchical structure.

Definition: Deterministic Nested word automaton

Definition: A *deterministic nested word automaton* (*DNWA*) over an alphabet Σ is a structure

$$
\begin{array}{llll}
(\; Q, & Q_0, & Q_f & \text{// linear states, initial, accepting} \\
, \; P, & P_0, & P_f & \text{// hierarchical states, initial, accepting} \\
, \; \delta_c, & \delta_i, & \delta_r & \text{// transitions: call, internal, return} \\
)
\end{array}
$$

where Q and P are sets of symbols,

Definition: Deterministic Nested word automaton

Definition: A *deterministic nested word automaton* (*DNWA*) over an alphabet Σ is a structure

$$
\begin{array}{llll}
(\; Q, & Q_0, & Q_f & \text{// linear states, initial, accepting} \\
, \; P, & P_0, & P_f & \text{// hierarchical states, initial, accepting} \\
, \; \delta_c, & \delta_i, & \delta_r & \text{// transitions: call, internal, return} \\
)
\end{array}
$$

where Q and P are sets of symbols, $Q_0 \in Q$, $P_0 \in P$, $Q_f \subset Q$, $P_f \subset P$,

Definition: Deterministic Nested word automaton

Definition: A *deterministic nested word automaton* (*DNWA*) over an alphabet Σ is a structure

$$
\begin{array}{llll}
(\ Q, & Q_0, & Q_f & \text{// linear states, initial, accepting} \\
,\ P, & P_0, & P_f & \text{// hierarchical states, initial, accepting} \\
,\ \delta_c, & \delta_i, & \delta_r & \text{// transitions: call, internal, return} \\
)
\end{array}
$$

where Q and P are sets of symbols, $Q_0 \in Q$, $P_0 \in P$, $Q_f \subset Q$, $P_f \subset P$, and the three δ are transition functions

$$
\begin{array}{lll}
\delta_c \subset & (\Sigma \times Q) & \mapsto (Q \times P) \\
\delta_i \subset & (\Sigma \times Q) & \mapsto Q \\
\delta_r \subset & (\Sigma \times Q \times P) & \mapsto Q
\end{array}
$$

Definition: DNWA: Run

The *run* of a DNWA over a nested word $(a_1..a_n, \rightsquigarrow)$ is defined as

- A sequence q_i for $i \in [1, n]$
- And a sequence p_i for all call positions i

Definition: DNWA: Run

The *run* of a DNWA over a nested word $(a_1..a_n, \rightsquigarrow)$ is defined as

- A sequence q_i for $i \in [1, n]$
- And a sequence p_i for all call positions i

so that for $i \in [1, n]$ it holds that:

- if i is a call position, then $\delta_c(a_i, q_{i-1}) = (q_i, p_i)$
- else if i is an internal position, then $\delta_i(a_i, q_{i-1}) = q_i$
- else if i is a return position (let h be its corresponding call position), then $\delta_r(a_i, q_{i-1}, p_h) = q_i$

Definition: DNWA: Run

The *run* of a DNWA over a nested word $(a_1..a_n, \rightsquigarrow)$ is defined as

- A sequence q_i for $i \in [1, n]$
- And a sequence p_i for all call positions i

so that for $i \in [1, n]$ it holds that:

- if i is a call position, then $\delta_c(a_i, q_{i-1}) = (q_i, p_i)$
- else if i is an internal position, then $\delta_i(a_i, q_{i-1}) = q_i$
- else if i is a return position (let h be its corresponding call position), then $\delta_r(a_i, q_{i-1}, p_h) = q_i$

Informally: q_i is the *linear trace* and p_i the *hierarchical trace*.

Definition: DNWA: Run

The *run* of a DNWA over a nested word $(a_1..a_n, \rightsquigarrow)$ is defined as

- A sequence q_i for $i \in [1, n]$
- And a sequence p_i for all call positions i

so that for $i \in [1, n]$ it holds that:

- if i is a call position, then $\delta_c(a_i, q_{i-1}) = (q_i, p_i)$
- else if i is an internal position, then $\delta_i(a_i, q_{i-1}) = q_i$
- else if i is a return position (let h be its corresponding call position), then $\delta_r(a_i, q_{i-1}, p_h) = q_i$

Informally: q_i is the *linear trace* and p_i the *hierarchical trace*.

The run is always uniquely and well-defined (after adding transitions to a black hole state where the transition functions are undefined)

Definition: DNWA: Acceptance

A DNWA accepts a nested word if the run over it ends in an accepting linear state:

Let A be a DNWA with accepting linear states Q_f, and let $(q_{1..n}, p_{1..m})$ be the run of A over a nested word w.
Then A *accepts* w **iff** $q_n \in Q_f$.

Example: Nested word automaton

Task: Given $\Sigma = \{[, (,],)\}$, build an acceptor for the language of properly balanced parentheses.

Example: Nested word automaton

Task: Given $\Sigma = \{[, (,],)\}$, build an acceptor for the language of properly balanced parentheses.

$$\begin{aligned} Q &= \{q\} \\ Q_0 &= q \\ Q_f &= \{q\} \end{aligned}$$

Example: Nested word automaton

Task: Given $\Sigma = \{[, (,],)\}$, build an acceptor for the language of properly balanced parentheses.

$$\begin{aligned} Q &= \{q\} \\ Q_0 &= q \\ Q_f &= \{q\} \\ P &= \Sigma \mathbin{\dot\cup} \{\perp\} \\ P_0 &= \perp \\ P_f &= \{\perp\} \end{aligned}$$

Example: Nested word automaton

Task: Given $\Sigma = \{[, (,],)\}$, build an acceptor for the language of properly balanced parentheses.

$$
\begin{array}{lcl}
Q & = & \{q\} \\
Q_0 & = & q \\
Q_f & = & \{q\} \\
P & = & \Sigma \mathbin{\dot\cup} \{\perp\} \\
P_0 & = & \perp \\
P_f & = & \{\perp\} \\
\delta_c & = & \{\ ((, q) \mapsto (q, (), \quad ([, q) \mapsto (q, [)\ \} \\
\delta_r & = & \{\ (), q, () \mapsto q, \quad (], q, [) \mapsto q\ \} \\
\delta_i & = & \emptyset
\end{array}
$$

Remarks

The last example is actually a showcase example for Pushdown automata.

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.
- The NWA has only an implicit stack. The stack manipulation type (push, pop, nothing) at each input symbol is only a function of the nesting structure.

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.
- The NWA has only an implicit stack. The stack manipulation type (push, pop, nothing) at each input symbol is only a function of the nesting structure.
- Push: exactly one element per call position.

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.
- The NWA has only an implicit stack. The stack manipulation type (push, pop, nothing) at each input symbol is only a function of the nesting structure.
- Push: exactly one element per call position.
- Pop: exactly one element per return position.

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.
- The NWA has only an implicit stack. The stack manipulation type (push, pop, nothing) at each input symbol is only a function of the nesting structure.
- Push: exactly one element per call position.
- Pop: exactly one element per return position.
- Reading the stack only when popping (returning).

Remarks

The last example is actually a showcase example for Pushdown automata.

Important differences of NWAs:

- The nesting structure is in the input nested word, **not** parsed at run-time.
- The NWA has only an implicit stack. The stack manipulation type (push, pop, nothing) at each input symbol is only a function of the nesting structure.
- Push: exactly one element per call position.
- Pop: exactly one element per return position.
- Reading the stack only when popping (returning).

With these restrictions, processing a nested word takes place in fixed **linear time and space**.

This is the last slide.