Linear Parsing Expression Grammars

○Nariyoshi Chida and Kimio Kuramitsu
Yokohama National University, Japan
chida-nariyoshi-pg@ynu.jp
kimio@ynu.ac.jp
@LATA2017, March 8, 2017

Parsing Expression Grammars (PEGs)

- A formal grammar introduced by B. Ford in 2004.

- PEGs are used for **parser generators**
  - PEG.js : a parser generator for JavaScript
  - Rats! (PLDI 2006)
  - Nez (Onward! 2016)
  - …

Parsing Expression Grammars (PEGs)

Example

- A PEG which recognizes a simple mathematical expression.

$$\begin{aligned} Expression &\leftarrow Sum \\ Sum &\leftarrow Product((+/-)Product)* \\ Product &\leftarrow Value((\times/\div)Value)* \\ Value &\leftarrow [0-9]* \end{aligned}$$

Parsing Expression Grammars (PEGs)

Example

- A PEG which recognizes a simple mathematical expression.

$$Expression \leftarrow Sum$$

$$Sum \leftarrow Product((+/-)Product)*$$

$$Product \leftarrow Value((\times/\div)Value)*$$

A nonterminal.

An expression.

$$Value \leftarrow [0-9]*$$

Parsing Expression Grammars (PEGs)

Example

- A PEG which recognizes a simple mathematical expression.

$$Expression \leftarrow Sum$$

$$Sum \;\ldots\; ct((+/-)Product)*$$

$$Produc \;\ldots\; (\times/\div)Value)*$$

$$Value \leftarrow [0-9]*$$

A choice in PEG.
Ordered choice.

Parsing Expression Grammars (PEGs)

Example

- A PEG which recognizes a simple mathematical expression.

$$Expression \leftarrow Sum$$

$$Sum \leftarrow P \ldots t)*$$

$$Product \leftarrow Value \ldots (\times/\div)Value)*$$

$$Value \leftarrow [0-9]*$$

A repetition in PEG.

Quiz:

- Are these PEGs convertible to DFAs?

1. $A \leftarrow a\ A\ b\ /\ c$

2. $A \leftarrow a\ A\ a\ /\ aa$

3. $A \leftarrow (\ a\ /\ ab\ )\ \ A\ (\ a\ /\ ab\ )\ /\ aa$

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c
2. A ← a A a / aa
3. A ← (a / ab) A (a / ab) / aa

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

The language is $\{a^i c b^i \mid i \geq 0\}$.

2. A ← a A a / aa

3. A ← (a / ab) A (a / ab) / aa

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

2. A ← a A a / aa

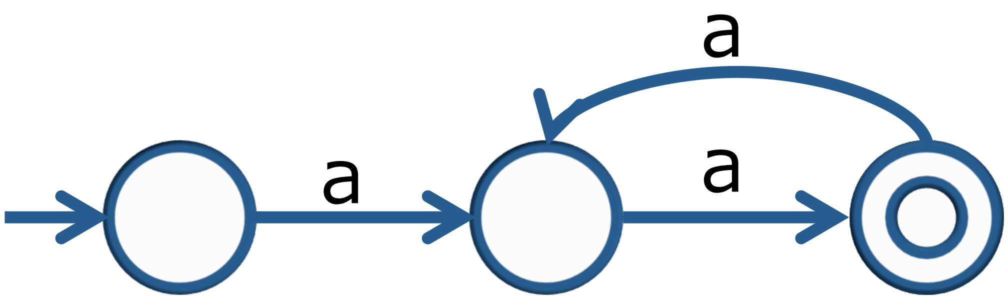
3. A ← (a / ab) A (a / ab) / aa

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.



2. A ← a A a / aa

Answer : Yes.
The language is $\{a^{2i} \mid i \geq 1\}$.

3. A ← (a / ab) A (a / ab) / aa

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

2. A ← a A a / aa

Answer : Yes.

3. A ← (a / ab) A (a / ab) / aa

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

2. A ← a A a / aa

Answer : Yes.

Due to the priority,
(a / ab) is the same as *a*.

3. A ← (a / ab) A (a / ab) / aa

Answer : Yes.
The language is $\{a^{2i} \mid i \geq 1\}$.
This is the same as question 2.

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

Can we check the regularity for an arbitrary PEGs?

2. A ← a A a / aa

Answer : Yes.

3. A ← (a / ab) A (a / ab) / aa

Answer : Yes.

Quiz:

- Are these PEGs convertible to DFAs?

1. A ← a A b / c

Answer : No.

Can we check the regularity for an arbitrary PEGs?

2. A ← a A a / aa

Answer : Yes.

3. A ← (a / ab) A

Answer : Yes.

Undecidable problem...

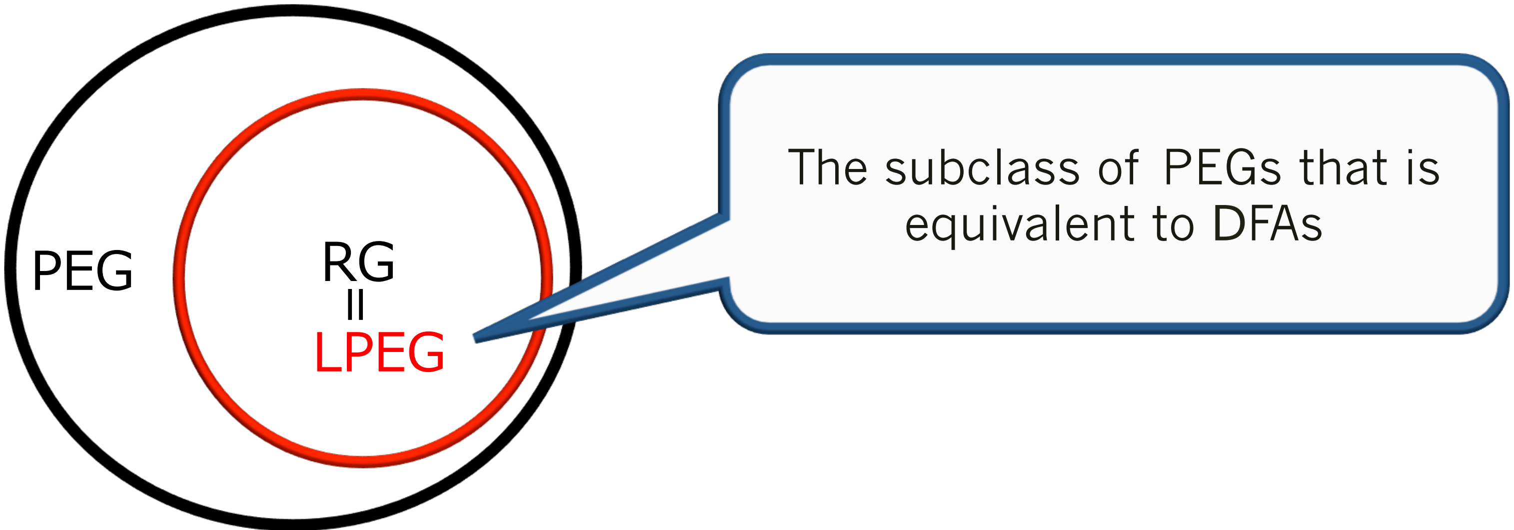
Contribution

We define the syntactic subclass of PEGs.

- We call it **Linear PEG (LPEG)**.

Merits

- Many techniques of REs are available
  - DFA transformation



Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

Parsing Expression Grammar (PEG)

- PEG G is a 4-tuple (N_G, Σ, e_S, P_G)
  - N_G : A finite set of nonterminals
  - Σ : A finite set of terminals
  - $e_S \in P_G$: A start expression
  - $P_G \in N_G \rightarrow e$: A finite set of rules

Parsing Expression Grammar (PEG)

- $P_G \in N_G \to e$: A rule
  - e ::= ε Empty

e ::=	ε	Empty
|	a	Character
|	.	Any character
|	e e	Sequence
|	e / e	**Prioritized choice**
|	e*	**Zero or more repetition**
|	!e	**Not-predicate**
|	&e	**And-predicate** (= !!e)
|	A	Nonterminal

Parsing Expression Grammar (PEG)

- $P_G \in N_G \rightarrow e$: A rule
  - e ::= ε　Empty

e ::=	ε	Empty
|	a	Character
|	.	Any character
|	e e	Sequence
|	e / e	**Prioritized choice**
|	e*	**Zero or more repetition**
|	!e	**Not-predicate**
|	&e	**And-predicate** (= !!e)
|	A	Nonterminal

ordered choice

Parsing Expression Grammar (PEG)

- $P_G \in N_G \rightarrow e$: A rule
  - e ::= ε Empty

e ::=	ε	Empty
|	a	Character
|	.	Any character
|	e e	Sequence
|	e / e	**Prioritized choice**
|	e*	**Zero or more repetition**
|	!e	**Not-predicate**
|	&e	**And-predicate** (= !!e)
|	A	Nonterminal

greedy repetition

Parsing Expression Grammar (PEG)

- $P_G \in N_G \to e$: A rule
  - e ::= ε　Empty

|	a	Character
|	.	Any character
|	e e	Sequence
|	e / e	**Prioritized choice**
|	e*	**Zero or more repetition**
|	!e	**Not-predicate**
|	&e	**And-predicate** (= !!e)
|	A	Nonterminal

lookahead

Languages

- The language $L(G)$ of a PEG $G = (N_G, \Sigma, e_S, P_G)$ is the set of strings $x \in \Sigma^*$ for which the start expression e_S matches x.

Example

Let $G = (\{\}, \{a, b\}, a, \{\})$.

$L(G) =$

$\{w \mid w \in \Sigma^*, \text{the prefix of the string } w \text{ is a}\}$.

Languages

- The language $L(G)$ of a PEG $G = (N_G, \Sigma, e_S, P_G)$ is the set of strings $x \in \Sigma^*$ for which the start expression e_S matches x.

Example

Let $G = (\{\}, \{a,$

$L(G) =$

$\{w \mid w \in \Sigma^*, \text{the prefix of the string } w \text{ is a}\}$.

Do not need to match entire string.

Languages

- The language $L(G)$ of a PEG $G = (N_G, \Sigma, e_S, P_G)$ is the set of strings $x \in \Sigma^*$ for which the start expression e_S matches x.

Example

Let $G = (\{\}, \{a, b\}, a, \{\})$.
$L(G) =$
$\{w \mid w \in \Sigma^*, \text{the prefix of the string } w \text{ is a}\}.$

w = a, aa, ab, aaa, aab, aba, abb,⋯

Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

Linear Parsing Expression Grammar (LPEG)

- LPEG G is a 4-tuple (N_G, Σ, e_S, P_G)
  - N_G : A finite set of nonterminals
  - Σ : A finite set of terminals
  - $e_S \in P_G$: A start expression
  - $P_G \in N_G \rightarrow e$: A finite set of rules

Linear Parsing Expression Grammar (LPEG)

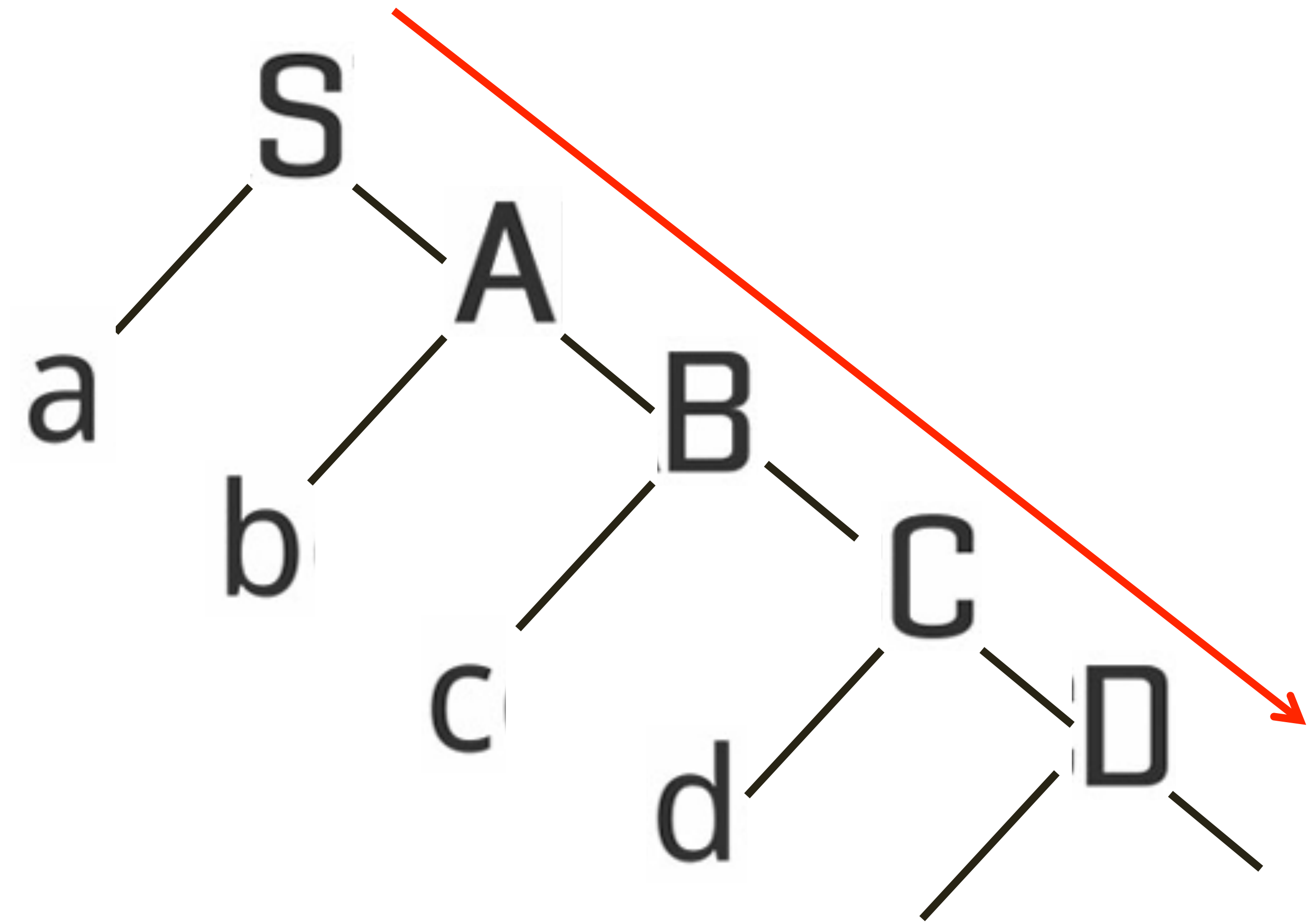
- $P_G \in N_G \to e$: A rule
  - e ::= p
    - | **p A**
    - | p e
    - | e / e
    - | &e e
    - | !e e

    A parsing expression that excludes some patterns of nonterminals (**linear parsing expression**)
  - p ::= ε
    - | a
    - | .
    - | p p
    - | p / p
    - | p*
    - | &p
    - | !p

    A parsing expression that excludes nonterminals (**n-free parsing expression**)

Linear Parsing Expression Grammar (LPEG)

PEGs whose syntax is limited to right-linear.



Linear Parsing Expression Grammar (LPEG)

Example
PEG $G = (\{A, B\}, \{a, b, c\}, A, P_G)$ is an LPEG, where P_G consists of the following rules:

$$
\begin{aligned}
A &\leftarrow \underline{aA}/\underline{bB}/c \\
B &\leftarrow \underline{aB}/\underline{bA}/c
\end{aligned}
$$

Nonterminals are not followed by expressions

Linear Parsing Expression Grammar (LPEG)

Example

PEG $G = (\{A, B\}, \{a, b, c\}, A, P_G)$ is an not LPEG, where P_G consists of the following rules:

Bad

$$A \leftarrow aAa / bB*$$
$$B \leftarrow aB / b$$

$B*$

⇓

$C \leftarrow BC / \epsilon$

Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

From DFAs to LPEGs

Theorem

LPEGs are a class that is equivalent to DFAs.

Steps of the proof

1. We show that for any DFA D there exists an LPEG G such that L(D) = L(G).
$\Rightarrow$ **From DFAs to LPEGs**

2. We show that for any LPEG G there exists a DFA D such that L(G) = L(D).
$\Rightarrow$ **From LPEGs to DFAs**

Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

From DFAs to LPEGs

Theorem

For any DFA D there exists an LPEG G such that $L(D) = L(G)$.

Sketch of proof

- Medeiros et al. showed the transformation from RE to PEG.
- We show that a PEG transformed from a RE is a right form of LPEG by mathematical induction.

Outline

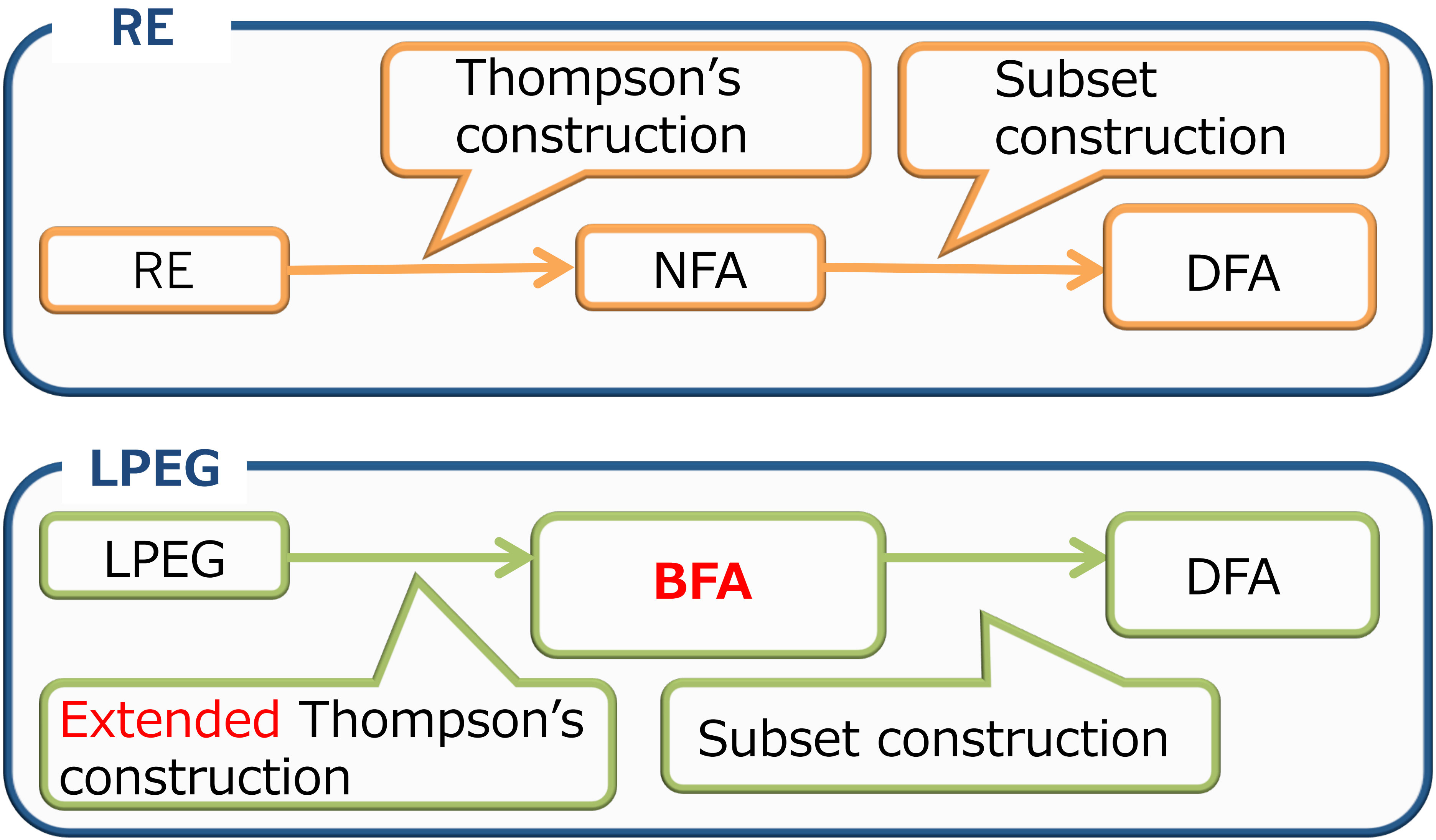
- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

From LPEGs to DFAs

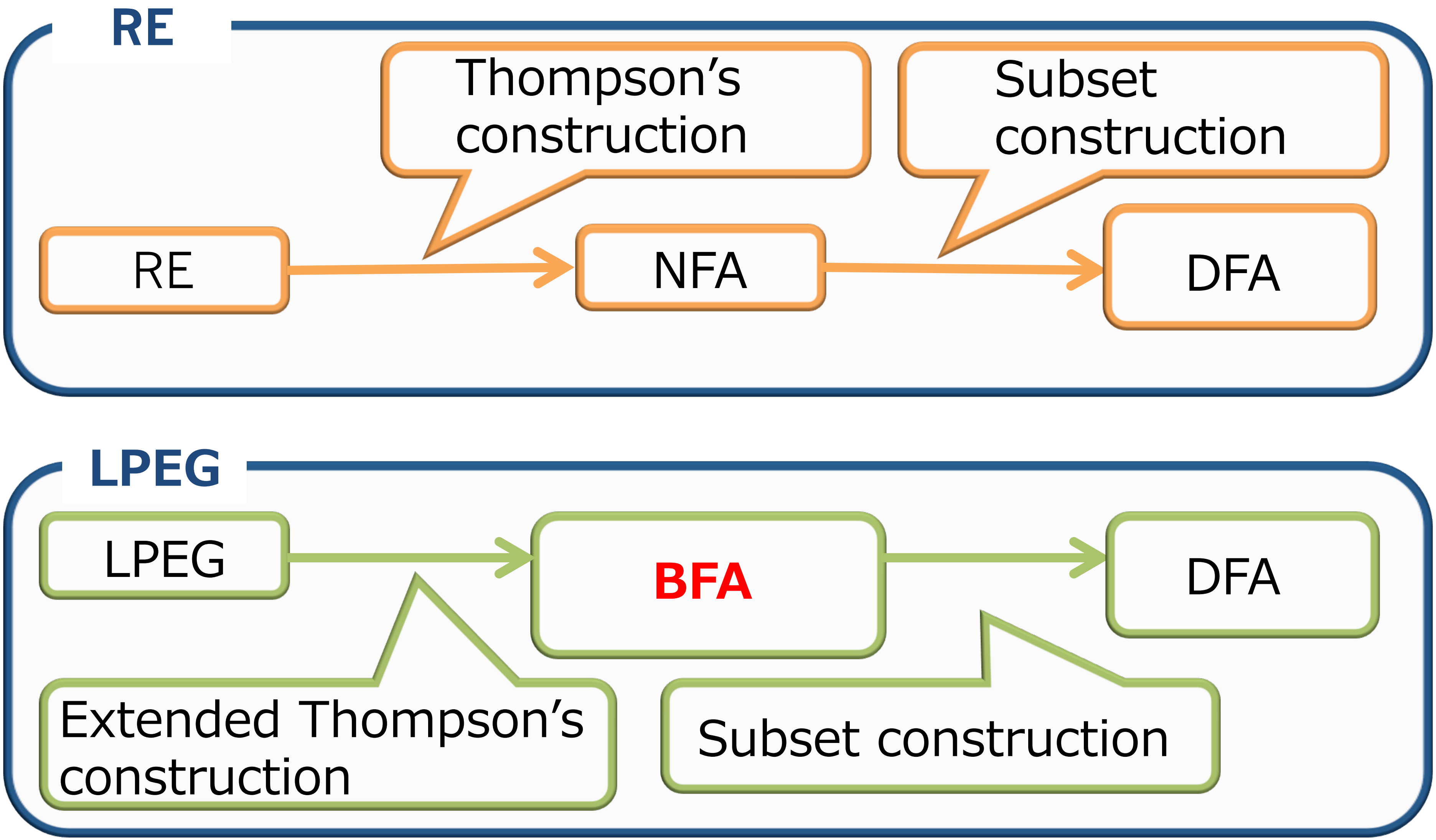
Theorem

For any LPEG G there exists a DFA D such that $L(G) = L(D)$.

A transformation from an LPEG to a DFA

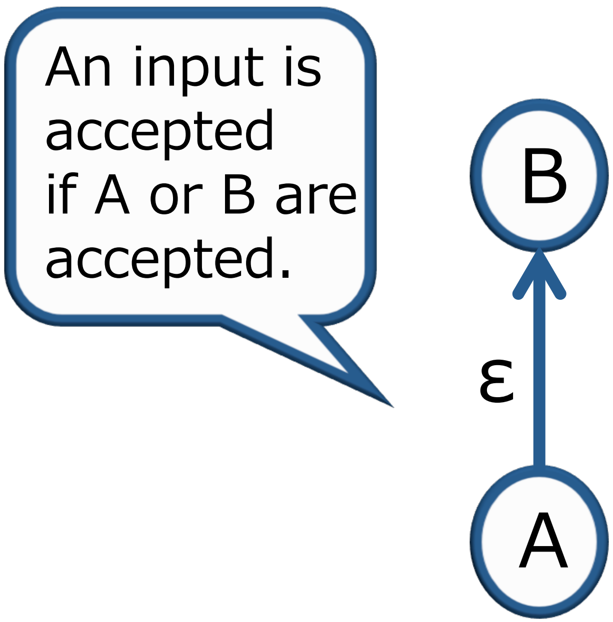


A transformation from an LPEG to a DFA



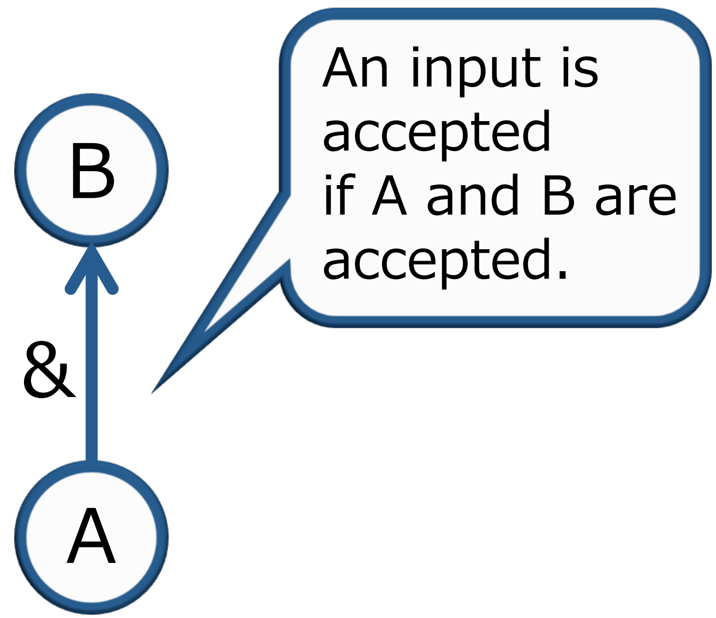
Why BFA?

- NFAs have…



"OR" transition

- NFAs do not have…

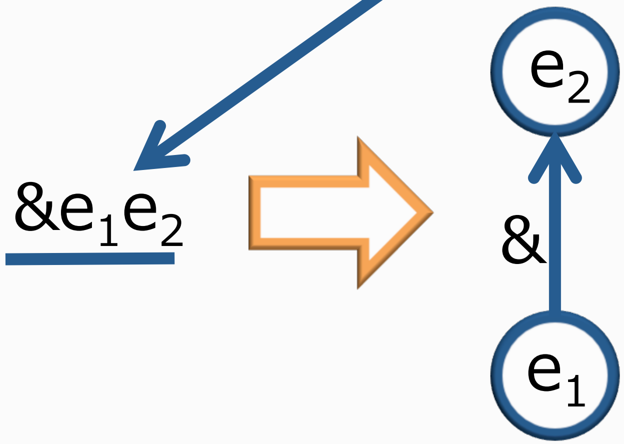


"AND" transition

Why BFA?

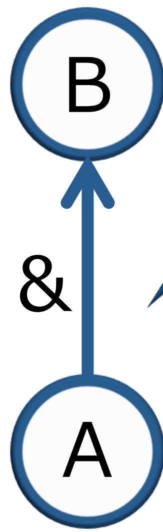
- NFAs have···

We need this transition to represent lookaheads.



"OR" transition

- NFAs do not have···



An input is accepted if A and B are accepted.

"AND" transition

Why BFA?

- In order to convert LPEGs to DFAs,
  we need automata that meets the following conditions:

- The automata
  1. Have the “AND” transition and “NAND” transition.
  2. Are convertible to DFAs.

Why BFA?

- In order to convert LPEGs to DFAs, we need automata that meets the following conditions:

- The automata
  1. Have the “AND” transition and “NAND” transition.
  2. Are convertible to DFAs.

⇒ **Boolean finite automata (BFAs)**

Boolean Finite Automata (BFAs)

- A BFA is a 5-tuple $B = (Q, \Sigma, \delta, f^0, F)$
  - Q is a finite non-empty set of states.
  - Σ is a finite set of terminals.
  - δ is a transition function that maps a state and a terminal into a boolean function
  - f^0 is an initial boolean function.
  - F is a finite set of accepting states.

Boolean Finite Automata (BFAs)

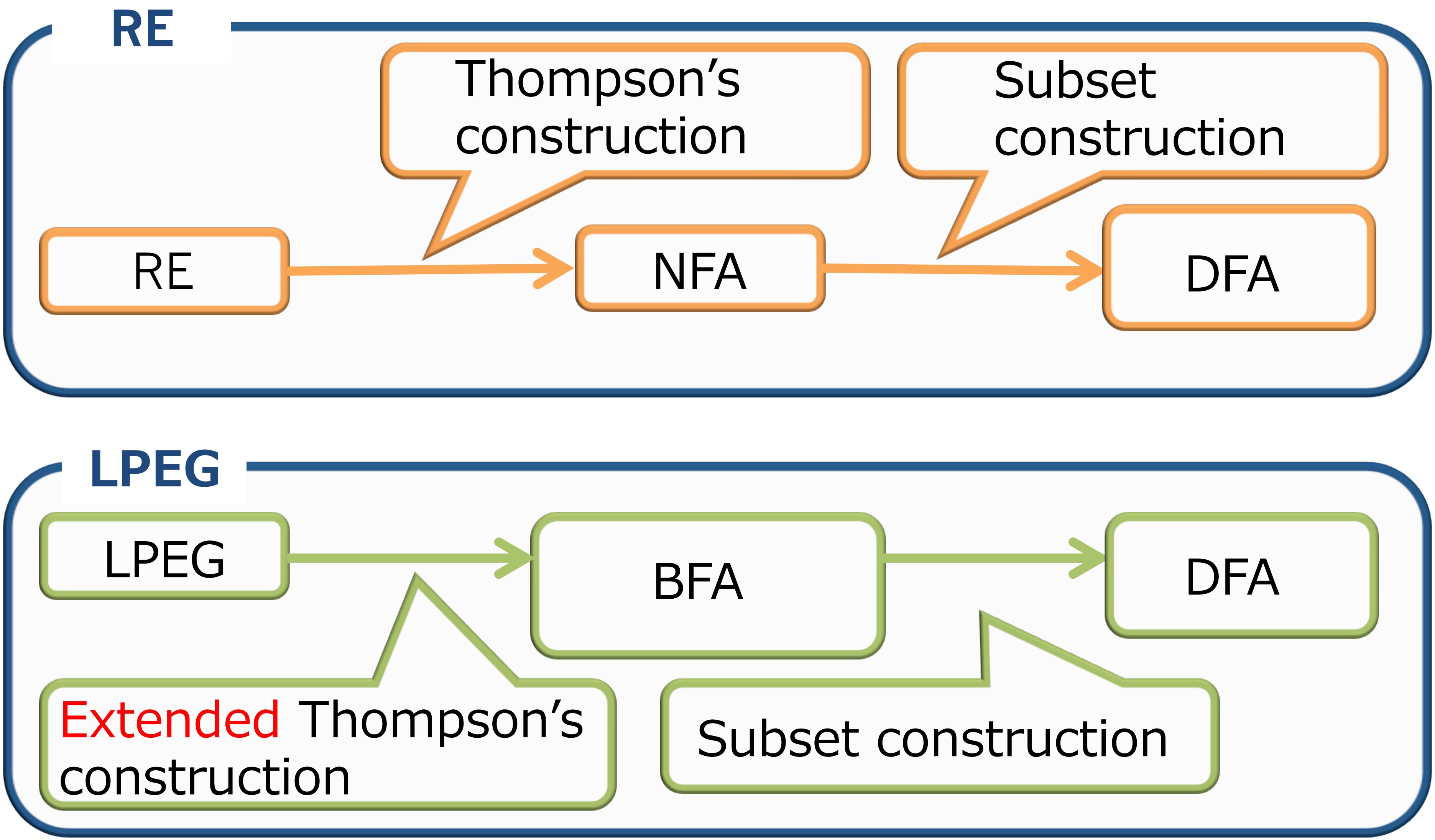
- A BFA is a generalization of NFA.
  - We can use general boolean functions on BFAs.
    - AND, NOT, OR ⋯
    - Not regex.
- A BFA is convertible to a DFA.

Theorem 2 (in [1])

For every boolean automaton B with n states there exists an equivalent deterministic automaton A_B with at most 2^{s^n} states, such that $L(A_B) = L(B)$.

[1] J.A. Brzozowski and E. Leiss: On equations for regular languages, finite automata, and sequential networks. Theoretical Computer Science. 10(1), 19-35(1980)

A transformation from an LPEG to a DFA



Extended Thompson's construction

- We can formalize the extended Thompson's construction as a function T_B.

- The function $T_B : e \to B$
  - Takes a linear parsing expression.
  - Returns a BFA that the language is equivalent to the linear parsing expression.

Function T_B

The foundation follows Morihata's work (Morihata,2012) for RE with lookaheads.

We extend his work with handling recursion.

$$
\begin{array}{rcl}
T(G) & = & (Q, \Sigma, \delta', f^0, F \cup P) \\
& where & (Q, \Sigma, \delta, f^0, F, P) = T_B(e_s) \\
& and & \delta' = \{((s, .), s) \mid s \in P\} \cup \delta \\
T_B(\epsilon) & = & (\{s\}, \Sigma, \{\}, s, \{s\}, \{\}) \\
T_B(a) & = & (\{s, t\}, \Sigma, \{((s, a), t)\}, s, \{t\}, \{\}) \\
T_B(!e) & = & (Q \cup \{s\}, \Sigma, \delta, s \wedge \overline{f^0}, \{s\}, F \cup P) \\
& where & (Q, \Sigma, \delta, f^0, F, P) = T_B(copy(e)) \\
T_B(e_1 e_2) & = & (Q_1 \cup Q_2, \Sigma, \delta, \phi(f_1^0, f_2^0, F_1), F_2, P_1 \cup P_2) \\
& where & (Q_1, \Sigma, \delta_1, f_1^0, F_1, P_1) = T_B(e_1), \\
& & (Q_2, \Sigma, \delta_2, f_2^0, F_2, P_2) = T_B(e_2) \\
& and & \delta = \{((s, a), \phi(t, f_2^0, F_1)) \mid ((s, a), t) \in \delta_1\} \cup \delta_2 \\
T_B(e_1 / e_2) & = & T_B(e_1 \mid !e_1 e_2) \\
T_B(e_1 \mid e_2) & = & (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, f_1^0 \vee f_2^0, F_1 \cup F_2, P_1 \cup P_2) \\
& where & (Q_1, \Sigma, \delta_1, f_1^0, F_1, P_1) = T_B(e_1) \\
& and & (Q_2, \Sigma, \delta_2, f_2^0, F_2, P_2) = T_B(e_2) \\
T_B(e^*) & = & T_B(e^\star !e) \\
T_B(e^\star) & = & (Q \cup \{s\}, \Sigma, \delta', s \vee f^0, F \cup \{s\}, P) \\
& where & (Q, \Sigma, \delta, f^0, F, P) = T_B(e) \\
& and & \delta' = \{((s, a), \phi(t, f^0, F)) \mid ((s, a), t) \in \delta\} \\
T_B(A) & = & \begin{cases} T_B(P_G(A)) \\ (\texttt{first application}) \\ \\ (\{\}, \Sigma, \{\}, f_{tmp_A}, \{\}, \{\}) \\ (\texttt{otherwise}) \end{cases}
\end{array}
$$

Function T_B

Morihata's works for RE with lookahead

Our extension

$$
\begin{array}{rcl}
T(G) & = & (Q, \Sigma, \delta', f^0, F \cup P) \\
 & \mathit{where} & (Q, \Sigma, \delta, f^0, F, P) = T_B(e_s) \\
 & \mathit{and} & \delta' = \{((s, .), s) \mid s \in P\} \cup \delta \\
T_B(\epsilon) & = & (\{s\}, \Sigma, \{\}, s, \{s\}, \{\}) \\
T_B(a) & = & (\{s, t\}, \Sigma, \{((s, a), t)\}, s, \{t\}, \{\}) \\
T_B(!e) & = & (Q \cup \{s\}, \Sigma, \delta, s \wedge \overline{f^0}, \{s\}, F \cup P) \\
 & \mathit{where} & (Q, \Sigma, \delta, f^0, F, P) = T_B(\mathit{copy}(e)) \\
T_B(e_1 e_2) & = & (Q_1 \cup Q_2, \Sigma, \delta, \phi(f_1^0, f_2^0, F_1), F_2, P_1 \cup P_2) \\
 & \mathit{where} & (Q_1, \Sigma, \delta_1, f_1^0, F_1, P_1) = T_B(e_1), \\
 & & (Q_2, \Sigma, \delta_2, f_2^0, F_2, P_2) = T_B(e_2) \\
 & \mathit{and} & \delta = \{((s, a), \phi(t, f_2^0, F_1)) \mid ((s, a), t) \in \delta_1\} \cup \delta_2 \\
T_B(e_1 / e_2) & = & T_B(e_1 \mid !e_1 e_2) \\
T_B(e_1 \mid e_2) & = & (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, f_1^0 \vee f_2^0, F_1 \cup F_2, P_1 \cup P_2) \\
 & \mathit{where} & (Q_1, \Sigma, \delta_1, f_1^0, F_1, P_1) = T_B(e_1) \\
 & \mathit{and} & (Q_2, \Sigma, \delta_2, f_2^0, F_2, P_2) = T_B(e_2) \\
T_B(e^*) & = & T_B(e^\star !e) \\
T_B(e^\star) & = & (Q \cup \{s\}, \Sigma, \delta', s \vee f^0, F \cup \{s\}, P) \\
 & \mathit{where} & (Q, \Sigma, \delta, f^0, F, P) = T_B(e) \\
 & \mathit{and} & \delta' = \{((s, a), \phi(t, f^0, F)) \mid ((s, a), t) \in \delta\} \\
T_B(A) & = & \begin{cases} T_B(P_G(A)) \\ \texttt{(first application)} \\ \\ (\{\}, \Sigma, \{\}, f_{tmp_A}, \{\}, \{\}) \\ \texttt{(otherwise)} \end{cases}
\end{array}
$$

From LPEGs to DFAs

Theorem

Let $G = (N_G, \Sigma, e_S, P_G)$ and $B = T_B(e_S.*)$.
Then, $L(G) = L(B)$.

Sketch of Proof

The proof is by induction on the structure of a linear parsing expression e. We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

Sketch of Proof

Case : e = !e

- We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

$$\begin{aligned} T_B(!e) &= (Q \cup \{s\}, \Sigma, \overline{\delta}, s \wedge \overline{f^0}, \{s\}, F \cup P) \\ \textit{where} &\quad (Q, \Sigma, \delta, f^0, F, P) = T_B(e) \end{aligned}$$

- We confirm that $T_B(e)$ is equivalent to e for any input $w \in \Sigma^*$.

Sketch of Proof

Case : e = !e

- We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

$$T_B(!e) \quad = \quad (Q \cup \{s\}, \Sigma, \overline{\delta}, s \wedge \overline{f^0}, \{s\}, F \cup P)$$
$$where \quad (Q, \Sigma, \delta, f^0, F, P) = T_B(e)$$

Let $B = T_B(e)$ and $B' = T_B(!e)$.
When e succeeds on w, then B also succeeds on w.
In this case,

- $!e$ fails on w
- B' rejects w since $s \wedge \overline{f^0} = s \wedge \overline{true} = false$

Sketch of Proof

Case : e = !e

- We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

$$T_B(!e) \quad = \quad (Q \cup \{s\}, \Sigma, \overline{\delta}, s \wedge \overline{f^0}, \{s\}, F \cup P)$$
$$where \quad (Q, \Sigma, \delta, f^0, F, P) = T_B(e)$$

Let $B = T_B(e)$ and $B' = T_B(!e)$.
When e succeeds on w, then B also succeeds on w.
In this case,

- $!e$ fails on w
- B' rejects w since $s \wedge \overline{f^0} = s \wedge \overline{true} = false$

Sketch of Proof

Case : e = !e

- We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

$$T_B(!e) \quad = \quad (Q \cup \{s\}, \Sigma, \overline{\delta}, s \wedge \overline{f^0}, \{s\}, F \cup P)$$
$$where \quad (Q, \Sigma, \delta, f^0, F, P) = T_B(e)$$

Let $B = T_B(e)$ and $B' = T_B(!e)$.
When e succeeds on w, then B also succeeds on w.
In this case,

- $!e$ fails on w
- B' rejects w since $s \wedge \overline{f^0} = s \wedge \overline{true} = false$

Sketch of Proof

Case : e = !e

- We assume that $T_B(e)$ is a BFA such that the language is equivalent to the language of e.

$$T_B(!e) \quad = \quad (Q \cup \{s\}, \Sigma, \delta, s \wedge \overline{f^0}, \{s\}, F \cup P)$$
$$where \quad (Q, \Sigma, \delta, f^0, F, P) = T_B(e)$$

Let $B = T_B(e)$ and $B' = T_B(!e)$.
When e fails on w, then B also fails on w.
In this case,

- $!e$ succeeds on w and consumes ε
- B' accepts ε since $s \wedge \overline{f^0} = true \wedge \overline{false} = true$

From LPEGs to DFAs

In the same way, we can confirm that the function T_B returns a BFA that is equivalent to the LPEG.

Hence, we say that for any LPEG G there exists a DFA D such that $L(D) = L(G)$.

Regularity of LPEGs

Consequently,

1. For any DFA D there exists an LPEG G such that $L(D) = L(G)$.

2. For any LPEG G there exists a DFA D such that $L(G) = L(D)$.

⇒ LPEG is a class that is equivalent to DFAs.

Outline

- Parsing Expression Grammar (PEG)
- Linear Parsing Expression Grammar (LPEG)
- Regularity of LPEGs
  - From DFAs to LPEGs
  - From LPEGs to DFAs
- Conclusion

Conclusion

We formalized LPEG that is an equivalent subclass to DFA.

- PEGs whose syntax is right-linear.

Open Problem

- L(PEG) $\supset$ L(CFG) problem
  - If so, we can parse any CFG in linear time.

LPEG to DFA Converter & Visualizer

http://regex-and-pe-to-dfa.com/