Parallel Parsing

The Implementation of a Parallel LL Parser Generator

William Henrich Due

30rd June 2023

KØBENHAVNS UNIVERSITET

Introduction

- Parallel LL parser generator.
- $\mathrm{LLP}(q, k) \subseteq \mathrm{LL}(k)$.
- Pareas uses $\mathrm{LLP}(1, 1)$.
- Two mistakes in the paper that describes LLP.
- Tested the parser generator thoroughly.

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$(abc, T, ())$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$(abc, T, ()) \vdash (abc, aTc, 2)$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$(abc, T, ()) \vdash (abc, aTc, 2) \vdash (bc, Tc, 2)$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$(abc, T, ()) \vdash (abc, aTc, 2) \vdash (bc, Tc, 2) \vdash (bc, Rc, (2, 1))$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$\begin{aligned}(abc, T, ()) &\vdash (abc, aTc, 2) \vdash (bc, Tc, 2) \vdash (bc, Rc, (2,1)) \\ &\vdash (bc, bRc, (2,1,4))\end{aligned}$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$(abc, T, ()) \vdash (abc, aTc, 2) \vdash (bc, Tc, 2) \vdash (bc, Rc, (2, 1))$$
$$\vdash (bc, bRc, (2, 1, 4)) \vdash (c, Rc, (2, 1, 4))$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$\begin{aligned}(abc, T, ()) &\vdash (abc, aTc, 2) \vdash (bc, Tc, 2) \vdash (bc, Rc, (2,1)) \\ &\vdash (bc, bRc, (2,1,4)) \vdash (c, Rc, (2,1,4)) \\ &\vdash (c, c, (2,1,4,3))\end{aligned}$$

LL parsing

Here is a LL(1) grammar.

$$1)\ T \to R \qquad 2)\ T \to aTc \qquad 3)\ R \to \varepsilon \qquad 4)\ R \to bR$$

$$\begin{aligned}(abc, T, ()) &\vdash (abc, aTc, 2) \vdash (bc, Tc, 2) \vdash (bc, Rc, (2,1)) \\ &\vdash (bc, bRc, (2,1,4)) \vdash (c, Rc, (2,1,4)) \\ &\vdash (c, c, (2,1,4,3)) \vdash (\varepsilon, \varepsilon, (2,1,4,3))\end{aligned}$$

Accepted! the string "*abc*" can be parsed.

LLP Parsing

Augment the grammar, this grammar is LL(1) and LLP(1, 1).

$$0)\ \ T' \to \vdash T \dashv$$

Now we parse the string "$\vdash abc \dashv$" instead and a LLP table is needed.

1. Initial pushdown store.
2. Final pushdown store.
3. Left parse.

	$\vdash$	a	b	c	$\dashv$
ε	$(T', T\dashv, 0)$				
$\vdash$		$(T, Tc, 2)$	$(T, R, (1,4))$		$(T\dashv, \varepsilon, (1,3))$
a		$(T, Tc, 2)$	$(T, R, (1,4))$	$(Tc, \varepsilon, (1,3))$	
b			$(R, R, 4)$	$(Rc, \varepsilon, 3)$	$(R\dashv, \varepsilon, 3)$
c				$(c, \varepsilon, ())$	$(\dashv, \varepsilon, ())$

LLP Parsing

Augment the grammar, this grammar is LL(1) and LLP(1, 1).

$$0)\ \ T' \rightarrow \vdash T \dashv$$

Now we parse the string "$\vdash abc \dashv$" instead and a LLP table is needed.

1. Initial pushdown store.
2. Final pushdown store.
3. Left parse.

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T \dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1, 4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0, 2, 1, 4, 3))$$

$$(T', \dashv, (0, 2, 1, 4, 3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0, 2)) \qquad (Tc, \varepsilon, (1, 4, 3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1, 4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0, 2, 1, 4, 3))$$

$$(T', \dashv, (0, 2, 1, 4, 3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0, 2)) \qquad (Tc, \varepsilon, (1, 4, 3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1, 4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0,2,1,4,3))$$

$$(T', \dashv, (0,2,1,4,3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0,2)) \qquad (Tc, \varepsilon, (1,4,3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1,4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0,2,1,4,3))$$

$$(T', \dashv, (0,2,1,4,3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0,2)) \qquad (Tc, \varepsilon, (1,4,3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1,4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0, 2, 1, 4, 3))$$

$$(T', \dashv, (0, 2, 1, 4, 3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0, 2)) \qquad (Tc, \varepsilon, (1, 4, 3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1, 4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Parallel Reduce

Use the associative **glue** operation.

$$(T', \varepsilon, (0, 2, 1, 4, 3))$$

$$(T', \dashv, (0, 2, 1, 4, 3)) \qquad (\dashv, \varepsilon, ())$$

$$(T', Tc\dashv, (0, 2)) \qquad (Tc, \varepsilon, (1, 4, 3))$$

$$(T', T\dashv, 0) \quad (T, Tc, 2) \quad (T, R, (1, 4)) \quad (Rc, \varepsilon, 3)$$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T \dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1, 4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T\dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1, 4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$
$(T', T\dashv)$	(T, Tc)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T\dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1, 4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$
$(T', T\dashv)$	(T, Tc)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(T', \dashv T)$	(T, cT)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T \dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1, 4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$
$(T', T \dashv)$	(T, Tc)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(T', \dashv T)$	(T, cT)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(\varepsilon, [^{\dashv}[^{T})$	$(]^{T}, [^{c}[^{T})$	$(]^{T}, [^{R})$	$(]^{R}]^{c}, \varepsilon)$	$(]^{\dashv}, \varepsilon)$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T\dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1,4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$
$(T', T\dashv)$	(T, Tc)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(T', \dashv T)$	(T, cT)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(\varepsilon, [^{\dashv}[^{T})$	$(]^{T}, [^{c}[^{T})$	$(]^{T}, [^{R})$	$(]^{R}]^{c}, \varepsilon)$	$(]^{\dashv}, \varepsilon)$

Now perform bracket matching and assert their types match up.

$$[^{\dashv}[^{T}]^{T}[^{c}[^{T}]^{T}[^{R}]^{R}]^{c}]^{\dashv}$$

LLP Parsing using Bracket Matching

$(\varepsilon, \vdash)$	$(\vdash, a)$	(a, b)	(b, c)	$(c, \dashv)$
$(T', T\dashv, 0)$	$(T, Tc, 2)$	$(T, R, (1,4))$	$(Rc, \varepsilon, 3)$	$(\dashv, \varepsilon, ())$
$(T', T\dashv)$	(T, Tc)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(T', \dashv T)$	(T, cT)	(T, R)	(Rc, ε)	$(\dashv, \varepsilon)$
$(\varepsilon, [^{\dashv}[^{T})$	$(]^{T}, [^{c}[^{T})$	$(]^{T}, [^{R})$	$(]^{R}]^{c}, \varepsilon)$	$(]^{\dashv}, \varepsilon)$

Now perform bracket matching and assert their types match up.

$$[^{\dashv}[^{T}]^{T}[^{c}[^{T}]^{T}[^{R}]^{R}]^{c}]^{\dashv}$$

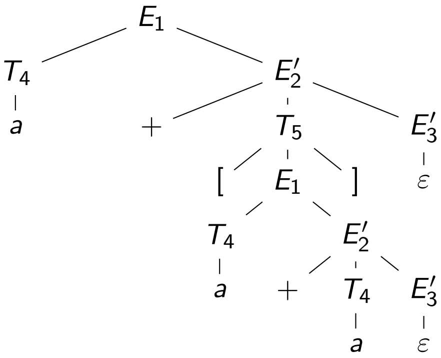
Construct the production sequence.

$$0, 2, 1, 4, 3$$

Syntax Tree

$$1)\; E \to TE' \quad 2)\; E' \to +TE' \quad 3)\; E' \to \varepsilon \quad 4)\; T \to a$$

$$(a + [a + a], E, ()) \vdash^* (\varepsilon, \varepsilon, (1, 4, 2, 5, 1, 4, 2, 4, 3, 3))$$



LLP Table

To perform LLP parsing a LLP table is needed.

1. Find the initial pushdown.
2. Parse the first symbol of the lookahead string to get the final pushdown store.
3. Construct LLP configuration.

The Problem

Consider the following augmented LL(2) grammar.

0) $S' \to \vdash S \dashv$ 1) $A \to \varepsilon$ 2) $S \to aAa$ 3) $A \to a$

The Problem

Consider the following augmented LL(2) grammar.

$$0)\ S' \to \vdash S \dashv \qquad 1)\ A \to \varepsilon \qquad 2)\ S \to aAa \qquad 3)\ A \to a$$

The PSLS (Prefix of a Suffix of a Leftmost Sentential) table. The grammar is LLP(2,2) since all the sets are singletons.

	$\dashv$	$a \dashv$	aa
$\vdash$			$\{S\}$
$\vdash a$		$\{A\}$	$\{A\}$
aa	$\{\dashv\}$	$\{a\}$	

The Problem

Consider the following augmented LL(2) grammar.

$$0)\ S' \rightarrow \vdash S \dashv \qquad 1)\ A \rightarrow \varepsilon \qquad 2)\ S \rightarrow aAa \qquad 3)\ A \rightarrow a$$

The PSLS (Prefix of a Suffix of a Leftmost Sentential) table. The grammar is LLP(2,2) since all the sets are singletons.

	$\dashv$	$a \dashv$	aa
$\vdash$			$\{S\}$
$\vdash a$		$\{A\}$	$\{A\}$
aa	$\{\dashv\}$	$\{a\}$	

A small example of a PSLS value.

$$S \Rightarrow^*_{lm} \vdash S \dashv \Rightarrow \vdash aAa \dashv \Rightarrow^* \vdash aa \dashv$$

This corresponds to the entry $\mathrm{PSLS}(\vdash a, a \dashv) = \{A\}$.

The Problem

Construct the LL(2) table.

	$\vdash a$	aa	$a \dashv$
S'	$S' \rightarrow \vdash S \dashv$		
S		$S \rightarrow aAa$	
A		$A \rightarrow a$	$A \rightarrow \varepsilon$

The Problem

Construct the LL(2) table.

	$\vdash a$	aa	$a \dashv$
S'	$S' \to \vdash S \dashv$		
S		$S \to aAa$	
A		$A \to a$	$A \to \varepsilon$

Try LL parsing the initial pushdown store $\text{PSLS}(\vdash a, a \dashv) = \{A\}$.

$$(a \dashv, A, ()) \vdash (a \dashv, \varepsilon, 1)$$

Due to this the final pushdown store cannot be determined since the first symbol can not be parsed.

The Problem

The problem is due to the PSLS definition.

$$\begin{aligned}\mathrm{PSLS}(x, y) = \{&\alpha : \exists S \Rightarrow^*_{lm} wuA\beta \Rightarrow wxB\gamma \Rightarrow^* wxy\delta,\\ &w, u \in T^*, A, B \in N, \alpha, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x,\\ &\alpha \text{ is the shortest prefix of } B\gamma \text{ such that } \mathrm{FIRST}_1(y) \subseteq \mathrm{FIRST}_1(\alpha)\}\\ \cup \{&a : \exists S \Rightarrow^* wuA\beta \Rightarrow wxa\gamma \Rightarrow^* wxy\delta,\\ &a = \mathrm{FIRST}_1(y), w, u \in T^*, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x\}\end{aligned}$$

The Problem

The problem is due to the PSLS definition.

$$
\begin{aligned}
\mathrm{PSLS}(x, y) = \{\alpha : \exists S \Rightarrow^*_{lm} wuA\beta \Rightarrow wxB\gamma \Rightarrow^* wxy\delta, \\
w, u \in T^*, A, B \in N, \alpha, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x, \\
\alpha \text{ is the shortest prefix of } B\gamma \text{ such that } \mathrm{FIRST}_1(y) \subseteq \mathrm{FIRST}_1(\alpha)\} \\
\cup \{a : \exists S \Rightarrow^* wuA\beta \Rightarrow wxa\gamma \Rightarrow^* wxy\delta, \\
a = \mathrm{FIRST}_1(y), w, u \in T^*, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x\}
\end{aligned}
$$

$$
\mathrm{FIRST}_1(y) \subseteq \mathrm{FIRST}_1(\alpha)
$$

$$
\not\Longrightarrow
$$

$$
(y, \mathrm{PSLS}(x, y), ()) \vdash^* (b, \omega, \pi) \text{ where } ab = y, a \in T \text{ and } b \in T^*
$$

The Problem

$$
\begin{aligned}
\mathrm{PSLS}_k(x,y) = \{\alpha : \exists S \Rightarrow^*_{lm} wuA\beta \Rightarrow wxB\gamma \Rightarrow^* wxy\delta,\\
w,u \in T^*, A, B \in N, \alpha,\beta,\gamma,\delta \in (N \cup T)^*, u \neq x,\\
\alpha \text{ is the shortest prefix of } B\gamma \text{ such that } y \in \mathrm{FIRST}_k(\alpha)\}\\
\cup \{y : \exists S \Rightarrow^* wuA\beta \Rightarrow wxa\gamma \Rightarrow^* wxy\delta,\\
a = \mathrm{FIRST}_1(y), w,u \in T^*, \beta,\gamma,\delta \in (N \cup T)^*, u \neq x\}
\end{aligned}
$$

The Problem

$$
\begin{aligned}
\mathrm{PSLS}_k(x, y) = \{\alpha : \exists S \Rightarrow_{lm}^{*} wuA\beta \Rightarrow wxB\gamma \Rightarrow^{*} wxy\delta,\\
w, u \in T^{*}, A, B \in N, \alpha, \beta, \gamma, \delta \in (N \cup T)^{*}, u \neq x,\\
\alpha \text{ is the shortest prefix of } B\gamma \text{ such that } y \in \mathrm{FIRST}_k(\alpha)\}\\
\cup \{y : \exists S \Rightarrow^{*} wuA\beta \Rightarrow wxa\gamma \Rightarrow^{*} wxy\delta,\\
a = \mathrm{FIRST}_1(y), w, u \in T^{*}, \beta, \gamma, \delta \in (N \cup T)^{*}, u \neq x\}
\end{aligned}
$$

- The time complexity is not impacted.
- The problem of deriving too many symbols.

"Since the length of both α_i and ω_i is limited by some constant z for the grammar, the time complexity is $O(z) = O(1)$." (Vagner, 2007)

The Problem

Construct the new $PSLS_2$ table.

	$\dashv$	$a \dashv$	aa
$\vdash$			$\{S\}$
$\vdash a$		$\{Aa \dashv\}$	$\{Aa\}$
aa	$\{\dashv\}$	$\{a \dashv\}$	

The Problem

Construct the new PSLS_2 table.

	$\dashv$	$a \dashv$	aa
$\vdash$			$\{S\}$
$\vdash a$		$\{Aa \dashv\}$	$\{Aa\}$
aa	$\{\dashv\}$	$\{a \dashv\}$	

Try LL parsing the initial pushdown store $\text{PSLS}_2(\vdash a, a \dashv) = \{A\}$.

$$(a \dashv, Aa \dashv, ()) \vdash (a \dashv, a \dashv, 1) \vdash (\dashv, \dashv, 1)$$

Now the final pushdown store can be found. Resulting in the LLP configuration $(Aa \dashv, \dashv, 1)$.

The Problem

The problem is due to the PSLS definition.

$$
\begin{aligned}
\mathrm{PSLS}(x, y) = \{\alpha : \exists S \Rightarrow_{lm}^{*} wuA\beta \Rightarrow wxB\gamma \Rightarrow^{*} wxy\delta,\\
w, u, y' \in T^{*}, a \in T, A, B \in N, \alpha, \beta, \gamma, \delta \in (N \cup T)^{*}, u \neq x,\\
y = ay', \alpha \text{ is the shortest prefix of } B\gamma\\
\text{such that } (y, \alpha, ()) \vdash^{*} (y', \omega, \pi)\}\\
\cup \{a : \exists S \Rightarrow^{*} wuA\beta \Rightarrow wxa\gamma \Rightarrow^{*} wxy\delta,\\
a = \mathrm{FIRST}_1(y), w, u \in T^{*}, \beta, \gamma, \delta \in (N \cup T)^{*}, u \neq x\}
\end{aligned}
$$

The Problem

The problem is due to the PSLS definition.

$$\begin{aligned}\mathrm{PSLS}(x, y) = \{\alpha : \exists S \Rightarrow^*_{lm} wuA\beta \Rightarrow wxB\gamma \Rightarrow^* wxy\delta,\\ w, u, y' \in T^*, a \in T, A, B \in N, \alpha, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x,\\ y = ay', \alpha \text{ is the shortest prefix of } B\gamma\\ \text{such that } (y, \alpha, ()) \vdash^* (y', \omega, \pi)\}\\ \cup \{a : \exists S \Rightarrow^* wuA\beta \Rightarrow wxa\gamma \Rightarrow^* wxy\delta,\\ a = \mathrm{FIRST}_1(y), w, u \in T^*, \beta, \gamma, \delta \in (N \cup T)^*, u \neq x\}\end{aligned}$$

1. Not tested.
2. It was created by Vladislav Vagner.

The Problem

Using Vladislav Vagners newer PSLS definition.

	$\dashv$	$a \dashv$	aa
$\vdash$			$\{S\}$
$\vdash a$		$\{Aa\}$	$\{A\}$
aa	$\{\dashv\}$	$\{a\}$	

Do LL parsing.

$$(a \dashv, Aa, ()) \vdash (a \dashv, a, 1) \vdash (\dashv, \varepsilon, 1)$$

My Mistakes

- The dictionaries are not surjective functions.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.
- The stack and input string does not need to equal each other in the definition of $\vdash$.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.
- The stack and input string does not need to equal each other in the definition of $\vdash$.
- "$\mathcal{M}_0(A) = \emptyset$ for all $A \in N' \setminus \{S\}$ and $\mathcal{M}_0(S) = \{\square^k\}$" is the correct initial dictionary when constructing FOLLOW_k.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.
- The stack and input string does not need to equal each other in the definition of $\vdash$.
- "$\mathcal{M}_0(A) = \emptyset$ for all $A \in N' \setminus \{S\}$ and $\mathcal{M}_0(S) = \{\square^k\}$" is the correct initial dictionary when constructing FOLLOW_k.
- N' should be used in step 3. of the FOLLOW_k algorithm.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.
- The stack and input string does not need to equal each other in the definition of $\vdash$.
- "$\mathcal{M}_0(A) = \emptyset$ for all $A \in N' \backslash \{S\}$ and $\mathcal{M}_0(S) = \{\square^k\}$" is the correct initial dictionary when constructing FOLLOW_k.
- N' should be used in step 3. of the FOLLOW_k algorithm.
- Algorithm 2.3 should use $v_j \in \mathrm{FIRST}_k(\gamma)$ not $y \in \mathrm{FIRST}_k(\gamma)$.

My Mistakes

- The dictionaries are not surjective functions.
- The property tested for is $\forall s \in T^* : \Pi(s) = \varphi(s)$.
- "This changed definition would still work for LLP parsing." is conjecture.
- The stack and input string does not need to equal each other in the definition of $\vdash$.
- "$\mathcal{M}_0(A) = \emptyset$ for all $A \in N' \backslash \{S\}$ and $\mathcal{M}_0(S) = \{\square^k\}$" is the correct initial dictionary when constructing FOLLOW_k.
- N' should be used in step 3. of the FOLLOW_k algorithm.
- Algorithm 2.3 should use $v_j \in \text{FIRST}_k(\gamma)$ not $y \in \text{FIRST}_k(\gamma)$.
- When showing the infinite loop, FIRST_k shoud be FIRST_1.

Usefulness

- Parsing large quantities of a formal language.
    - JSON.

Usefulness

- Parsing large quantities of a formal language.
    - JSON.
    - Large code bases in reproducible environments.

Usefulness

- Parsing large quantities of a formal language.
  - JSON.
  - Large code bases in reproducible environments.
- The LLP grammar class is limited: problems with JSON, Lua, Regular Language.

Usefulness

- Parsing large quantities of a formal language.
    - JSON.
    - Large code bases in reproducible environments.
- The LLP grammar class is limited: problems with JSON, Lua, Regular Language.
- I had trouble finding useful grammars that are LL(k) grammars where $k > 1$. But extending the lookback is somewhat different to the lookahead.

Usefulness

- Parsing large quantities of a formal language.
  - JSON.
  - Large code bases in reproducible environments.
- The LLP grammar class is limited: problems with JSON, Lua, Regular Language.
- I had trouble finding useful grammars that are LL(k) grammars where $k > 1$. But extending the lookback is somewhat different to the lookahead.
- Parsing more complex grammars, an example is allowing "$-$" to mean both the binary and unary operation for numbers, while ignore parenthesis. When testing this such a grammar was found to be LLP(2, 1).

Usefulness

- Parsing large quantities of a formal language.
    - JSON.
    - Large code bases in reproducible environments.
- The LLP grammar class is limited: problems with JSON, Lua, Regular Language.
- I had trouble finding useful grammars that are LL(k) grammars where $k > 1$. But extending the lookback is somewhat different to the lookahead.
- Parsing more complex grammars, an example is allowing "$-$" to mean both the binary and unary operation for numbers, while ignore parenthesis. When testing this such a grammar was found to be LLP(2, 1).
- There exists a larger grammar class LLP* which can parse all regular languages and more of LL.

A Little Example

A LISP Parser