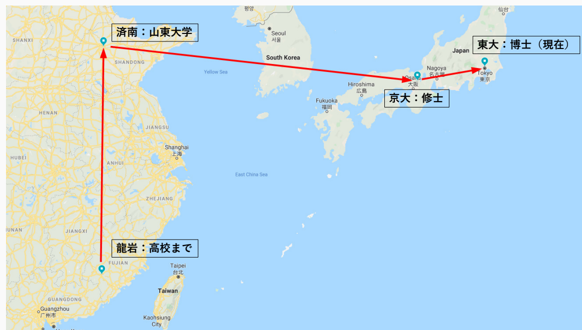
研 究 紹 介

李 楽（リ ラク）
博士 1 年
メール: lile@eidos.ic.i.u-tokyo.ac.jp
2019 年 11 月 27 日

東京大学情報理工学系研究科 電子情報学専攻 田浦研究室

自己紹介

- 李　楽 (リ　ラク)　博士１年
- メール: lile@eidos.ic.i.u-tokyo.ac.jp
- 東京大学情報理工学系研究科　電子情報学専攻　田浦研究室



- 好きなもの
  - こたつ、サッカー、コーヒー
  - ...(たくさんあるよと)

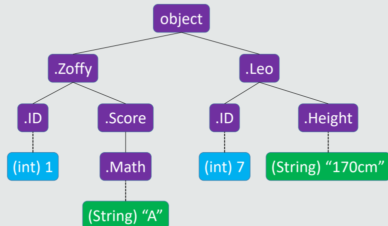
研究内容 (背景)

- ⋆ JSON や XML などは流行ってる
- ⋆ ただの文字列なので、コンピュータは「理解」できない

JSON の例

```
{"Zoffy": {
     "ID": 1,
     "Score": {"Math": "A" }
 },
 "Leo": {
     "ID": 7,
     "Height": "170cm"
 }
}
```

データ構造の例



このうえで有用なデータを読み取らせる。例えばクエリ、

```
クエリ： select object.Zoffy.ID
出力　： 1
```

研究内容 (背景)

JSON を処理したいなら、その文法に従って構文解析器 (Parser) を作れば (既存のものを使えば) いいじゃない？

文法：JSON

```
JSON     := null | true | false
          | NUMBER | STRING
          | OBJECT | ARRAY
OBJECT   := { }
          | { MEMBERS }
ARRAY    := [ ]
          | [ ELEMENTS ]
MEMBERS  := STRING : JSON
          | MEMBERS , STRING : JSON
ELEMENTS := JSON
          | ELEMENTS , JSON
```

ファイル：data.json

```
{"Zoffy":{"ID":1,"Score":{"Math":"A"}},
"Leo":{"ID":7,"Height":"170cm"}}
```

プログラム：processJson.py

```
import json

with open(data.json, 'r') as f:
    data = f.read()
json = json.loads(data)
zoffyID = int(json.get("Zoffy").get("ID"))
```

流行ってる JSON や XML ならいいものの

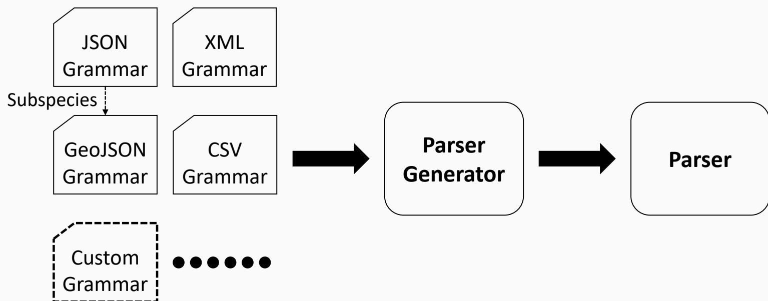
フォーマット/文法	構文解析器
数えきれない存在してる	汎用性が低い
バージョンアップする、亜種がある	書き直しが面倒
特定ケースにカスタマイズする	1 から作る必要がある

それぞれに対して、構文解析器を作ろう → 重複作業が多い

研究内容 (背景)

解析器ジェネレータ (Parser Generator)

- 与えられた文法に従って、構文解析器を自動的に生成する



プログラムを書くより、文法を書いたほうが効率的

研究内容 (課題)

原状：解析器ジェネレータは新たな研究課題ではない

- コンパイラの生成に欠かせない一環 (Compiler-compiler)
- 既にたくさん存在する (ANTLR, Bison ...)

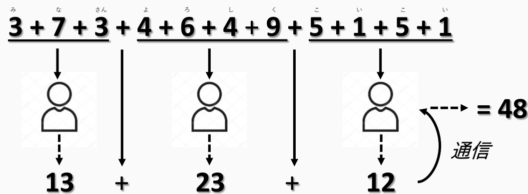
現状：この時代においては、何が求められてるのか？

傾向	解析器
データ数の爆発 (Facebook: アップロード 1GB+/s)	更に高性能
スレッドあたりの性能が頭打ち (コア数や並列技術で性能を埋める)	並列化の実装、デバッグが面倒

$\rightarrow$ 課題：解析器の最適化と並列化もジェネレータに任せたい

並列構文解析器のジェネレータ
→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 並列化



逐次：１０回の足し算
並列：３＋２＝５
(計算量最大の人の計算量＋結果まとめるための計算量)
→ 通信遅延を無視したら、２倍の性能向上
→ N 分割しても、N 倍にはならない

研究紹介

並列構文解析器のジェネレータ

→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 構文解析器の並列化

缶詰職人が缶を缶詰にするように、缶を缶詰にすることができますか？
Oh, **can you can a can** as the canner can can a can?

文法

```
疑問文(MAIN):= 助動詞 主語 述語
主語        := 人称代名詞
             | 名詞
述語        := 動詞 人称代名詞
             | 動詞 名詞
助動詞      := can
人称代名詞  := you
動詞        := can
名詞        := a can
```

研究紹介

並列構文解析器のジェネレータ

→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 構文解析器の並列化

缶詰職人が缶を缶詰にするように、缶を缶詰にすることができますか？

Oh, **can you can | a can** as the canner can can a can?

can you can **a can** **(助動詞can　主語you　述語can a can)**

文法

```
疑問文(MAIN):= 助動詞 主語 述語
主語        := 人称代名詞
             | 名詞
述語        := 動詞 人称代名詞
             | 動詞 名詞
助動詞      := can
人称代名詞  := you
動詞        := can
名詞        := a can
```

研究紹介

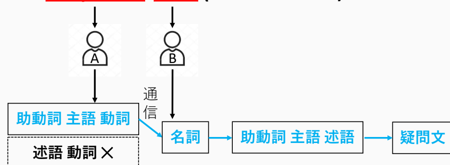
並列構文解析器のジェネレータ

→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 構文解析器の並列化

缶詰職人が缶を缶詰にするように、缶を缶詰にすることができますか？

Oh, **can you can | a can** as the canner can can a can?



文法

```
疑問文(MAIN):= 助動詞 主語 述語
主語       := 人称代名詞
            | 名詞
述語       := 動詞 人称代名詞
            | 動詞 名詞
助動詞     := can
人称代名詞 := you
動詞       := can
名詞       := a can
```

→A の計算量が比較的に多いので、もうちょっと平均的に分割できないかな？

研究紹介

並列構文解析器のジェネレータ

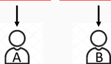
→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 構文解析器の並列化

缶詰職人が缶を缶詰にするように、缶を缶詰にすることができますか？

Oh, **can you | can a can** as the canner can can a can?

can you **can a can** **(助動詞can 主語you 述語can a can)**



文法

```
疑問文(MAIN):= 助動詞 主語 述語
主語        := 人称代名詞
             | 名詞
述語        := 動詞 人称代名詞
             | 動詞 名詞
助動詞      := can
人称代名詞  := you
動詞        := can
名詞        := a can
```

研究紹介

並列構文解析器のジェネレータ

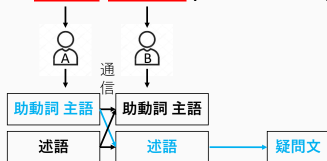
→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ 構文解析器の並列化

缶詰職人が缶を缶詰にするように、缶を缶詰にすることができますか？

Oh, **can you | can a can** as the canner can can a can?

can you **can a can** **(助動詞can　主語you　述語can a can)**



文法

```
疑問文(MAIN):= 助動詞 主語 述語
主語       := 人称代名詞
           | 名詞
述語       := 動詞 人称代名詞
           | 動詞 名詞
助動詞      := can
人称代名詞   := you
動詞       := can
名詞       := a can
```

どのルールから解析するのか？(全部の組合せを試す)

通信終了まで解析できるか確保できない (通信遅延が高い)

→ 作業負荷の均衡がなかなか難しい

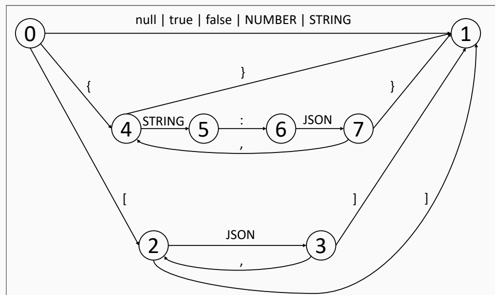
並列構文解析器のジェネレータ
　→ 並列化 → 構文解析器の並列化 → オートマトンの並列化

⋆ オートマトンの並列化 (問題を数理化しょう)

文法：JSON

```
JSON     := null
          | true
          | false
          | NUMBER
          | STRING
          | OBJECT
          | ARRAY
OBJECT   := { }
          | { MEMBERS }
ARRAY    := [ ]
          | [ ELEMENTS ]
MEMBERS  := STRING : JSON
          | MEMBERS , STRING : JSON
ELEMENTS := JSON
          | ELEMENTS , JSON
```

JSON のオートマトン



どのルールから解析するのか？(全部の組合せを試す)
　→ どの状態から始まるのか？

研究紹介

高速化 $\neq$ 並列化
$\rightarrow$ 並列化は到底高速化の一つの手段だけ

高速化のほかの手段

- 計算量の少ないデータ構造
- コンパイラ (ループ展開...)、ハードウェア (局所性...) の特性を考慮したアルゴリズム
- 数値計算ライブラリ BLAS などの利用 (巨人の肩の上に立つ)
  ......