Abstract Syntax Networks for Code Generation and Semantic Parsing

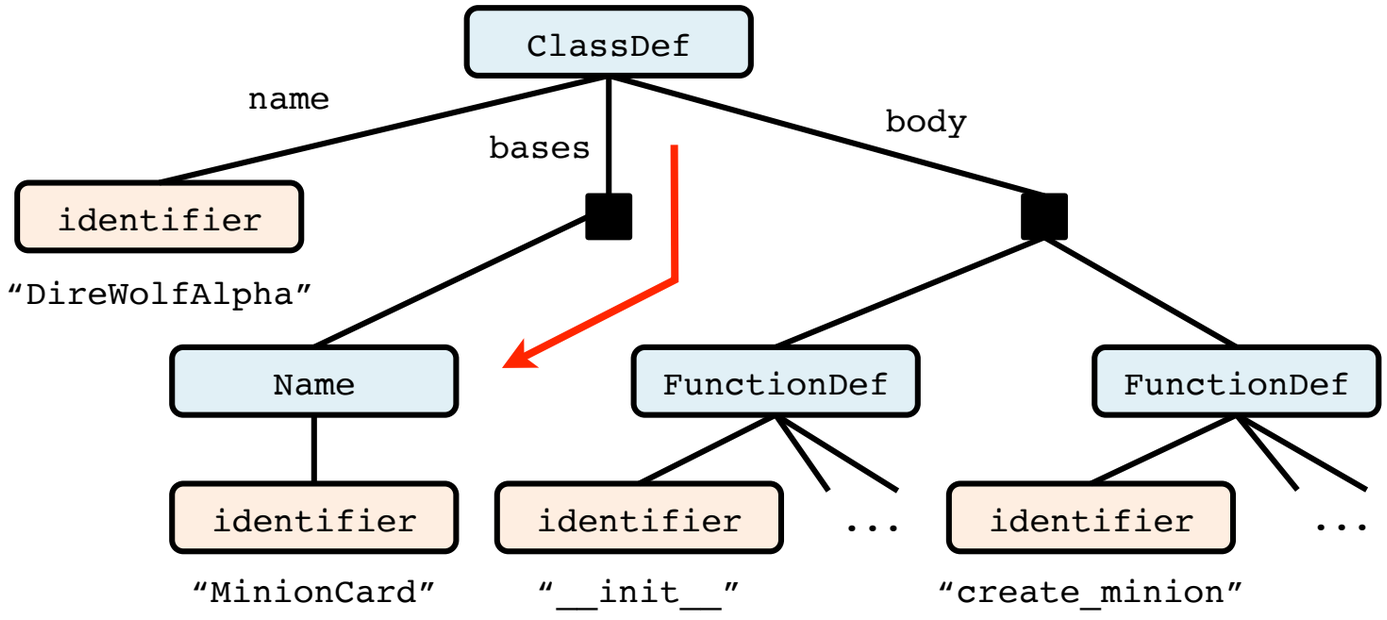
Maxim Rabinovich* **Mitchell Stern*** **Dan Klein**
Computer Science Division
University of California, Berkeley
{rabinovich,mitchell,klein}@cs.berkeley.edu

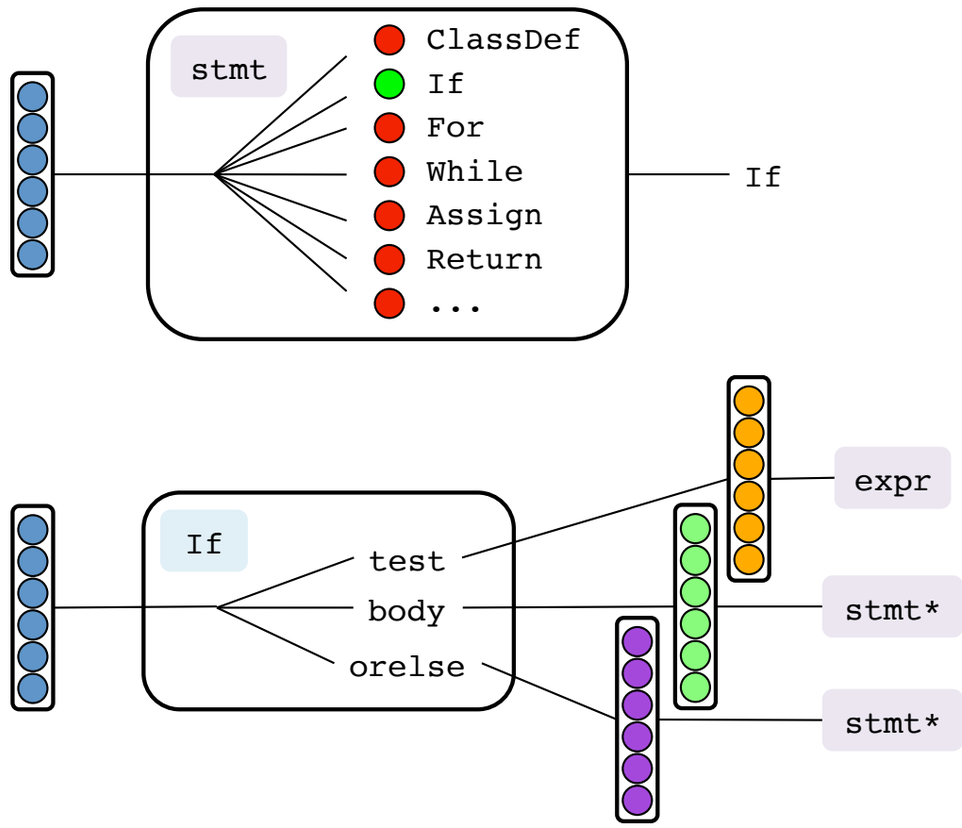
読む人: 能地 宏 (NAIST)

SNLP 2017

図は論文より引用

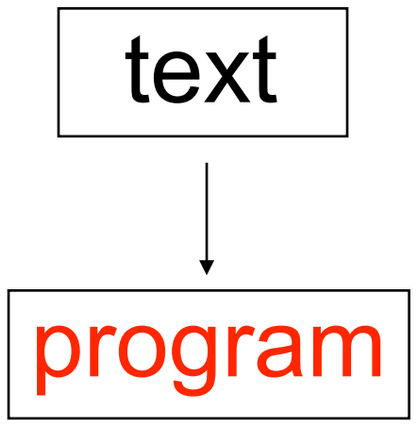
Abstract syntax networks





- Abstract syntax tree (左) を生成する neural networks
- Top-down (vertical) LSTM +
  ノードに応じて LSTM の計算法が変化 (モジュール; 左)
- 文法の仕様 (ASDL) によってモジュールが定まる

タスク: code generation



sort my list in descending order

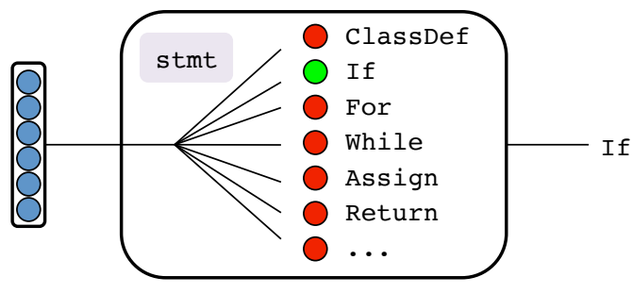
```
sorted(mylist, reverse=True)
```

- 自然言語 (や他の入力) から目的の言語の対応するコードを得る
- 教師あり学習: (text, program) のペアから学習
- 翻訳に近い (が、出力の構造に対する制約が厳しい)
  - `if cond1: body1; else: body2`
  - Well-formedness, well-typedness, executability
  - **Abstract syntax network**: well-formedness を保証するために言語の文法仕様を組み込むことのできるモデル

Outline

- Hearthstone
  - 去年リリースされた code generation のベンチマークのためのデータセット (Ling et al., 2016)

- Abstract syntax networks
  - 記述した文法に従って決定されるモデル



- 結果と分析、今後の課題、議論


```
class ManaWyrm(MinionCard):
  def __init__(self):
    super().__init__(
      'Mana Wyrm', 1,
      CHARACTER_CLASS.MAGE,
      CARD_RARITY.COMMON)
  def create_minion(self, player):
    return Minion(
      1, 3, effects=[
        Effect(
          SpellCast(),
          ActionTag(
            Give(ChangeAttack(1)),
                 SelfSelector()))
      ])
```

Hearthstone データセット (Ling+'16)

- カードのメタデータとその python 実装の組からなるデータ
  - カードの属性値は得られていることを仮定 (画像処理しない)


```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```

```
class DireWolfAlpha(MinionCa
  def __init__(self):
    super().__init__(
      "Dire Wolf Alpha", 2, C
      CARD_RARITY.COMMON, mi
  def create_minion(self, pla
    return Minion(2, 2, auras
      Aura(ChangeAttack(1), M
    ])
```

input (query) ⟶ output (program)

データの目的？

- (恐らく) 特定のアプリケーションを意図して作られたものではない
- 入力データからある程度複雑なプログラム (ここでは1つのpython クラス) を生成する手法のためのベンチマーク
- なぜカードゲームと python クラスか？
  - カードゲームのオープンソース実装が存在するため、カードに対応するコードの収集が容易であるため

Hearthstone

https://us.battle.net/hearthstone/ja/game-guide/

Heathbreaker

https://github.com/danielyule/hearthbreaker/

readme.md

Hearthbreaker

A Hearthstone Simulator

Hearthbreaker is an open source Hearthstone simulator for the purposes of machine lea
Hearthstone: Heroes of WarCraft. It implements every card in the game. Every attempts
Hearthstone precisely, including edge cases and bugs. The results of playing simulated
cards which work well together and cards which do not.
Hearthbreaker is not designed to allow player to play Hearthstone against each other, n
human opponents within Hearthstone itself. It is designed to be used as a library for ana

1カード = 1クラス

https://github.com/danielyule/hearthbreaker/blob/5dad317744882cb4c4fbbce53edc3d7f4576552e/hearthbreaker/cards/minions/neutral.py#L138

```
137
138 class DireWolfAlpha(MinionCard):
139     def __init__(self):
140         super().__init__("Dire Wolf Alpha", 2, CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
141
142     def create_minion(self, player):
143         return Minion(2, 2, auras=[Aura(ChangeAttack(1), MinionSelector(Adjacent()))])
144
```

Simulator 上での実装

データ集め


```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```

```
class DireWolfAlpha(MinionCa
  def __init__(self):
    super().__init__(
      "Dire Wolf Alpha", 2,
      CARD_RARITY.COMMON, mi
  def create_minion(self, pl
    return Minion(2, 2, aura
      Aura(ChangeAttack(1),
])
```

人手で属性値を抽出

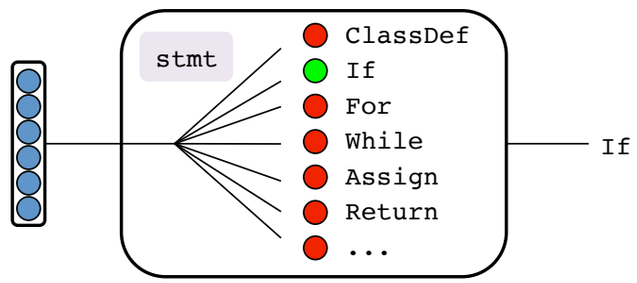
input (query) ⟶ output (program)

訓練データ: 533ペア; テストデータ: 66ペア

Outline

- Hearthstone
  - 去年リリースされた code generation のベンチマークのためのデータセット (Ling+'2016)

- Abstract syntax networks
  - 記述した文法に従って決定されるモデル



- 結果と分析、今後の課題


```
class ManaWyrm(MinionCard):
  def __init__(self):
    super().__init__(
      'Mana Wyrm', 1,
      CHARACTER_CLASS.MAGE,
      CARD_RARITY.COMMON)
  def create_minion(self, player):
    return Minion(
      1, 3, effects=[
        Effect(
          SpellCast(),
          ActionTag(
            Give(ChangeAttack(1)),
                 SelfSelector()))
      ])
```

近年の seq2seq generation

- 文だけでなく線形化した構造を出力するモデルが流行

- Vinvalys+'15 NIPS
  - 句構造 parsing ができてしまう

John has a dog . → (S (NP NNP)$_{\mathrm{NP}}$ (VP VBZ (NP DT NN)$_{\mathrm{NP}}$)$_{\mathrm{VP}}$.)$_{\mathrm{S}}$

- Jia & Liang'16 ACL
  - logical form を出力
  - data augmentation

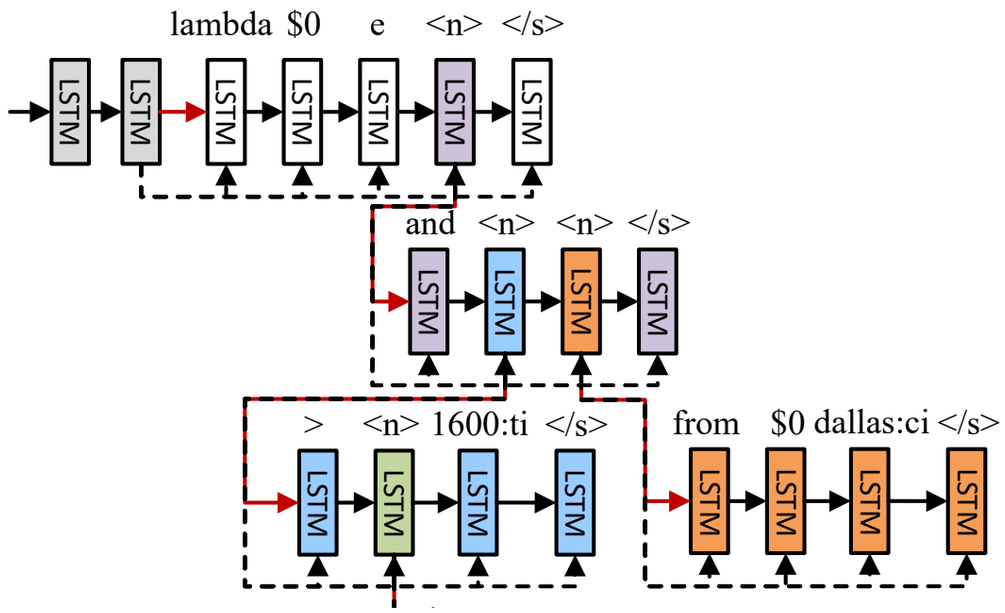
x: "*what is the population of iowa ?*"
y: `_answer ( NV , ( _population ( NV , V1 ) , _const ( V0 , _stateid ( iowa ) ) ) )`

- Konstas+'17 ACL
  - AMR parsing もできるよ

```
(h / hold-04
  :ARG0 (p2 / person
    :ARG0-of (h2 / have-org-role-91
      :ARG1 (c2 / country
```

過去の研究と問題点

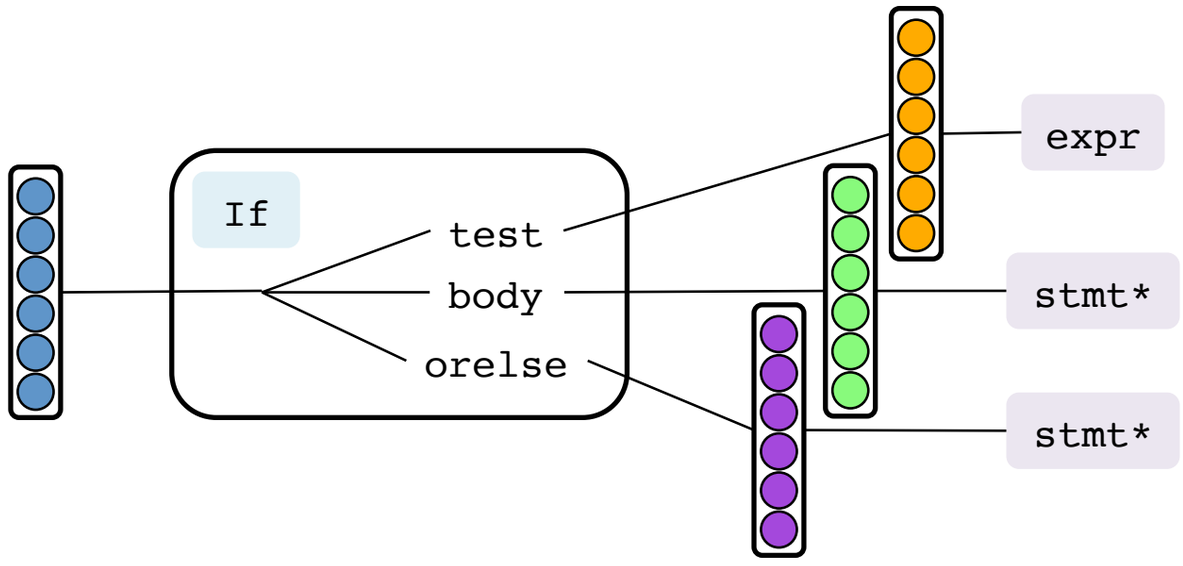
- コード生成の既存研究
  - Ling+'16: Latent Predictor Networks
    - 基本的に seq2seq + attention
    - 問題点: 正しい python 文法のコードを吐くことは保証されない
  - Dong & Lapata'16: top-down LSTM
    - Top-down なので木構造の出力は保証できるが、木構造が python の文法に則っていることは保証されない



本研究の主な主張

- 一つのクラスのコードのような複雑な構造を出力するには単純な seq2seq ではダメ (特に well-formedness)

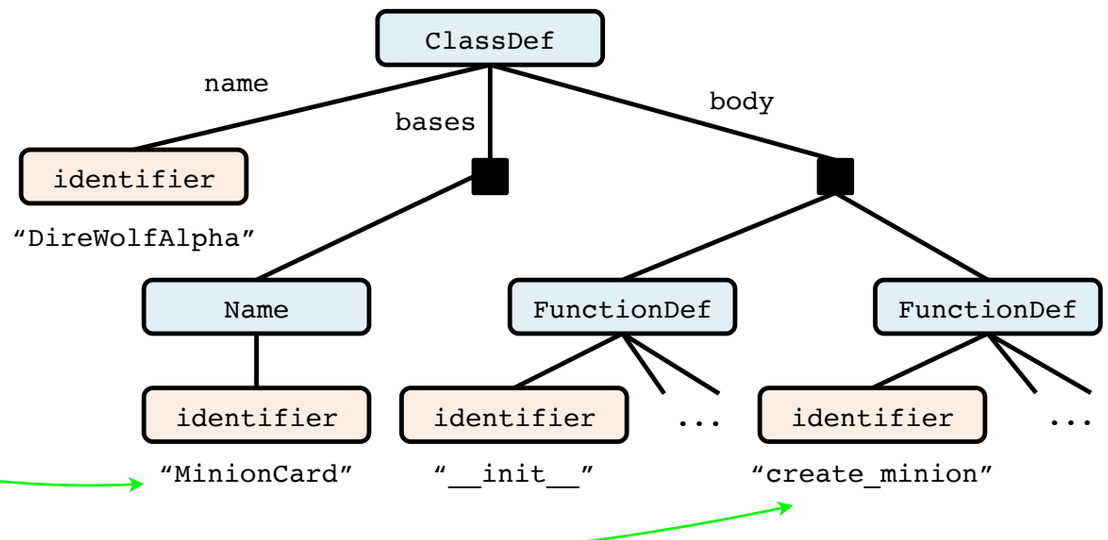
- プログラミング言語の文法規則に則って丁寧にモデル化を行うことが重要



- モデルは複数モジュールからなり、文法を指定すれば定まる
  - 文法を記述すれば他のデータにも適用可能 (e.g., Prolog)

Abstract Syntax Networks

```
class DireWolfAlpha(MinionCard):
  def __init__(self):
    super().__init__(
      "Dire Wolf Alpha", 2, CHARACTER_CLASS.ALL,
      CARD_RARITY.COMMON, minion_type=MINION_TYPE.BEAST)
  def create_minion(self, player):
    return Minion(2, 2, auras=[
      Aura(ChangeAttack(1), MinionSelector(Adjacent()))
    ])
```

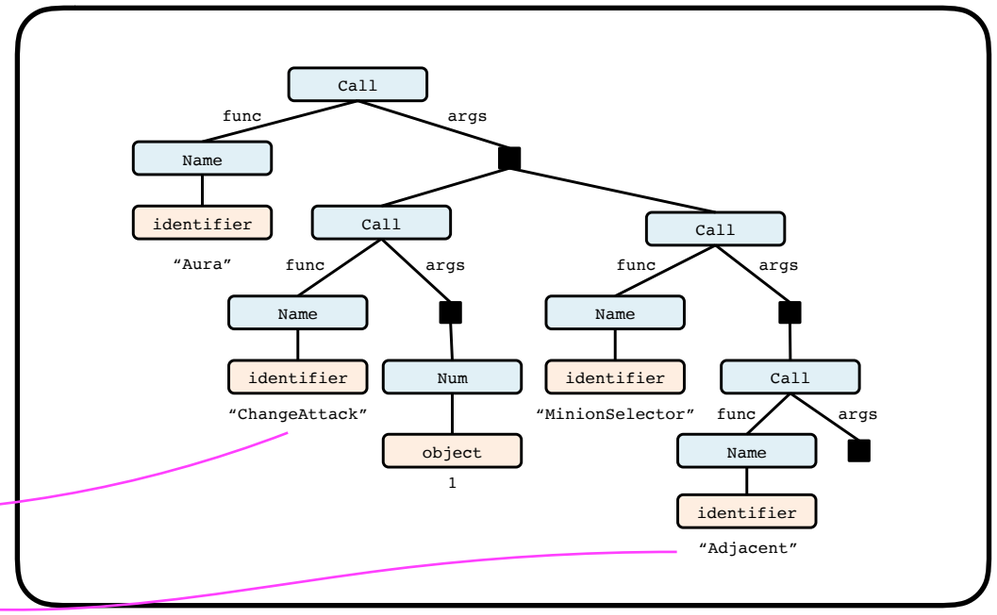


- コード (左) は抽象構文木 (右) に変換できる
  - decoder は抽象構文木を生成し、それをコードに逆変換
  - python の文法規則を外部から与える (abstract syntax description language)

```
stmt
 = FunctionDef(
     identifier name, arg* args, stmt* body)
 | ClassDef(
     identifier name, expr* bases, stmt* body)
 | Return(expr? value)
 | ...
```

Encoder-decoder (with attention)


```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```



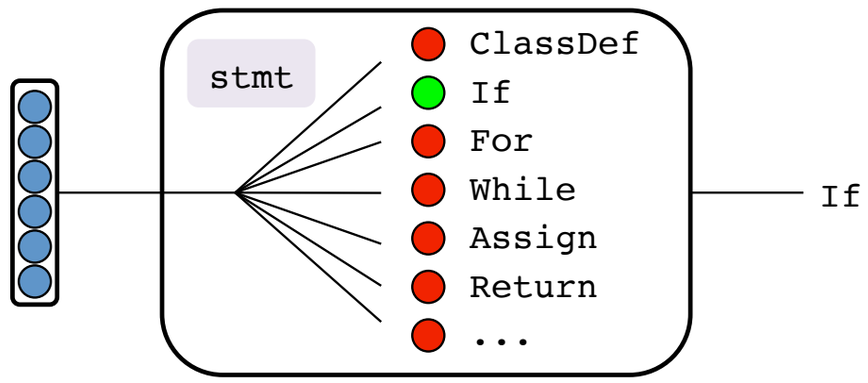
input ⟶ output

- input は属性値毎に別の bi-LSTM で encode
- decoder への初期値: 全て結合して線形変換
- 入力を attend しながら、 top-down な LSTM で生成
- supervised attention (ヒューリスティックに alignment)

文法によってモジュールが決まる

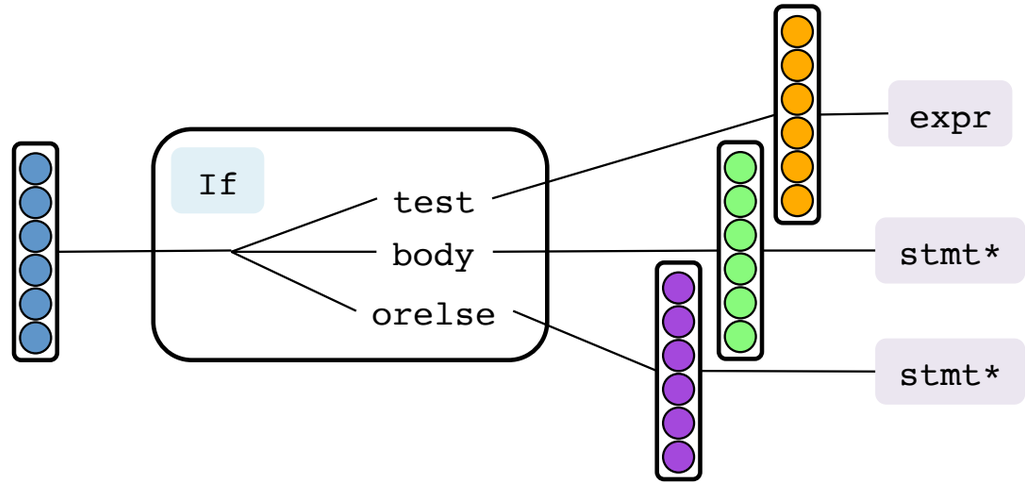
```
primitive types: identifier, object, ...

stmt
 = FunctionDef(
     identifier name, arg* args, stmt* body)
 | ClassDef(
     identifier name, expr* bases, stmt* body)
 | Return(expr? value)
 | ...
```



モジュール1: どの命令を選ぶか

```
expr
 = BinOp(expr left, operator op, expr right)
 | Call(expr func, expr* args)
 | Str(string s)
 | Name(identifier id, expr_context ctx)
 | ...
```



モジュール2: 内部の生成

- ネットワークは python 用にハードコードするのではなく文法を通して自動で生成 (他の文法にも適用可能)

Semantic parsing も可能

what microsoft jobs do not require a bscs?

answer(company(J,'microsoft'),job(J),not((req_deg(J,'bscs'))))

logical form

Jobs dataset

```
expr
  = Apply(pred predicate, arg* arguments)
  | Not(expr argument)
  | Or(expr left, expr right)
  | And(expr* arguments)

arg
  = Literal(lit literal)
  | Variable(var variable)
```

Figure 9: The Prolog-style grammar we use for the JOBS task.

文法を書いて、 logical form ⇔ 抽象構文木の変換を定めれば良い

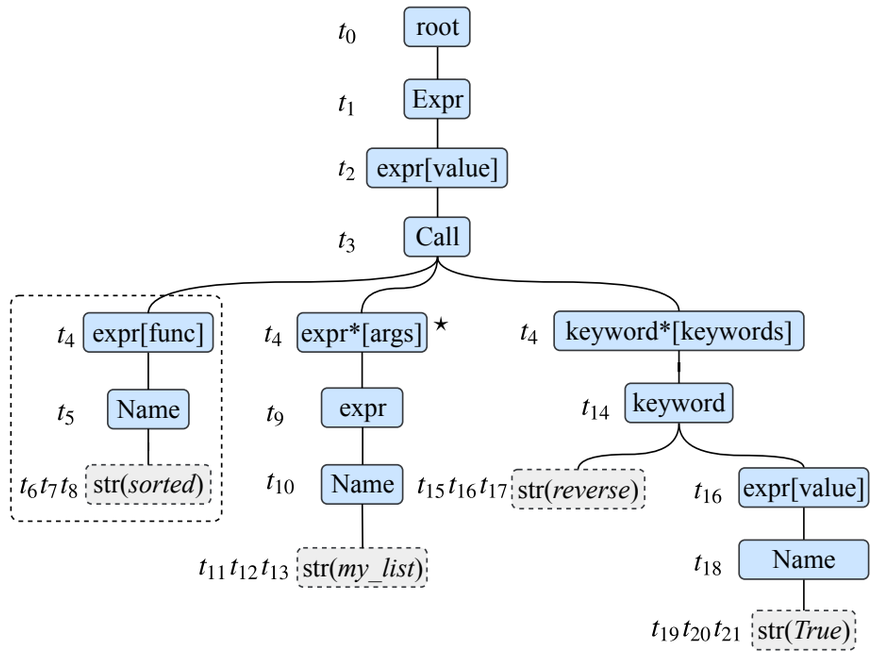
(ほぼ) 同じ研究 ACL'17

A Syntactic Neural Model for General-Purpose Code Generation

Pengcheng Yin
Language Technologies Institute
Carnegie Mellon University
pcyin@cs.cmu.edu

Graham Neubig
Language Technologies Institute
Carnegie Mellon University
gneubig@cs.cmu.edu

- モデルのスコア計算が異なる
- 提案法は LSTM の内部状態がモジュール固有の feed-forward で変化するが、彼らは一つの (top-down) LSTM のみ

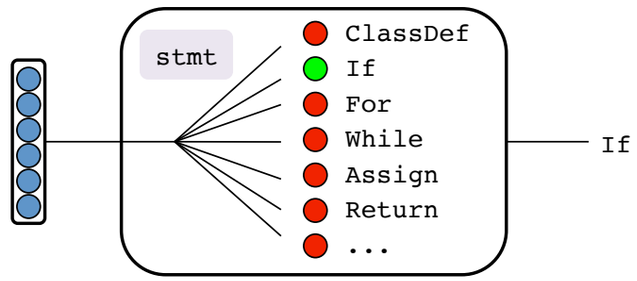


abstract syntax tree

Outline

- Hearthstone
  - 去年リリースされた code generation のベンチマークのためのデータセット (Ling+'2016)

- Abstract syntax networks
  - 記述した文法に従って決定されるモデル



- 結果と分析、今後の課題


```
class ManaWyrm(MinionCard):
  def __init__(self):
    super().__init__(
      'Mana Wyrm', 1,
      CHARACTER_CLASS.MAGE,
      CARD_RARITY.COMMON)
  def create_minion(self, player):
    return Minion(
      1, 3, effects=[
        Effect(
          SpellCast(),
          ActionTag(
            Give(ChangeAttack(1)),
                 SelfSelector()))
      ])
```

結果

System	Accuracy	BLEU	F1
NEAREST	3.0	65.0	65.7
LPN	6.1	67.1	–
ASN	**18.2**	**77.6**	**72.4**
+ SUPATT	**22.7**	**79.2**	**75.6**

Our system	16.2	**75.8**	Yin & Neubig'17

- Accuracy: コードの完全一致
- モジュールを使った丁寧なモデル化と supervised attention が重要

割と複雑なコードも生成可能


```
class ManaWyrm(MinionCard):
  def __init__(self):
    super().__init__(
      'Mana Wyrm', 1,
      CHARACTER_CLASS.MAGE,
      CARD_RARITY.COMMON)
  def create_minion(self, player):
    return Minion(
      1, 3, effects=[
        Effect(
          SpellCast(),
          ActionTag(
            Give(ChangeAttack(1)),
                 SelfSelector())))
      ])
```

より複雑な例


```
class MultiShot(SpellCard):
  def __init__(self):
    super().__init__(
      'Multi-Shot', 4,
      CHARACTER_CLASS.HUNTER,
      CARD_RARITY.FREE)
  def use(self, player, game):
    super().use(player, game)
    targets = copy.copy(
                game.other_player.minions)
    for i in range(0, 2):
      target = game.random_choice(targets)
      targets.remove(target)
      target.damage(
        player.effective_spell_damage(3),
        self)
def can_use(self, player, game):
  return (
    super().can_use(player, game) and
    (len(game.other_player.minions) >= 2))
```

```
class MultiShot(SpellCard):
  def __init__(self):
    super().__init__(
      'Multi-Shot', 4,
      CHARACTER_CLASS.HUNTER,
      CARD_RARITY.FREE)
  def use(self, player, game):
    super().use(player, game)
    minions = copy.copy(
                game.other_player.minions)
    for i in range(0, 3):
      minion = game.random_choice(minions)
      minions.remove(minion)
  def can_use(self, player, game):
    return (
      super().can_use(player, game) and
      len(game.other_player.minions) >= 3)
```

gold　　pred

命令中の数字の扱いが逆になっている (neural っぽい？)

評価にも問題 (e.g., 変数名の不一致)

おわりに

- Encoder-decoder で decode 対象が複雑な構造 (木, グラフ) を持っている時に対象に対する知識を組み込む自然な方法
  - 色々な拡張はできそう (AMR など)
  - Encoder-decoder AMR は alignment の問題を回避できる

- 完全な code-generation には程遠い
  - well-typedness, executability
  - Semantic coherence (離れた場所での変数名, etc)