現代プログラマのための

Prolog入門

Sep. 12 2024 - Masaki Hara @ Tech Lunch

今回話すこと

- **Prolog**は論理型プログラミング言語
- 古くは人工知能として持て囃された
- その実態は「単一化」と「バックトラック」を持つ特殊な手続き型言語
- 現代では型推論器など記号処理タスクの実行エンジンとして大いに役に立っている

10分でPrologのことを理解するのはさすがに無理なので、大まかな特徴や応用をざっくり掴んでもらえたらなと思います。

Prolog入門

起動と終了

```
$ brew install swi-prolog

$ swipl

?- halt.
```

またはCtrl+Dで終了

```
# test1.pl
parent(taro, hanako).
parent(jiro, hanako).
sibling(X, Y) :-
  parent(X, Z),
  parent(Y, Z).
```

```
?- [test1].
```

ファイル名を指定して
ロード

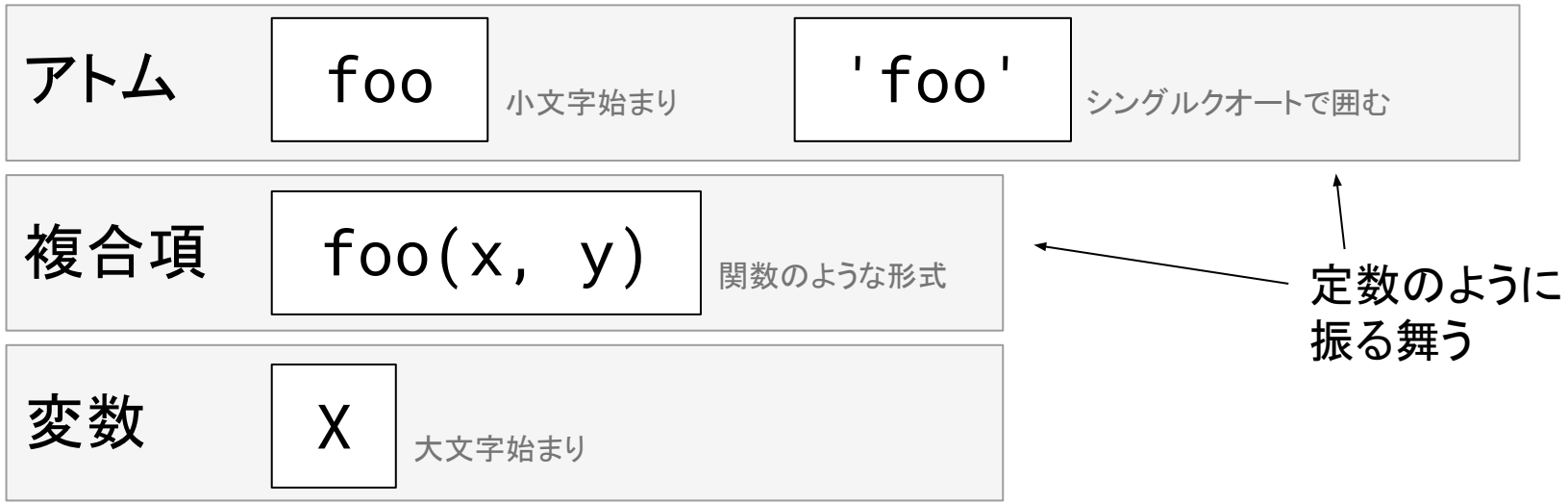
```
?- sibling(taro, X).
X = taro ; SPACE
X = jiro.
```

Enterで中断
Spaceで続行

Prologの基本的な使い方

- ファイルに定義を書き、コマンドラインで命令を行う
- **Prolog**処理系は、「条件を満たす変数を全て見つける」処理を行う
  - 前ページの例では X が変数になっている
  - sibling(taro, X) という条件は、X = taro または X = jiro を入れれば成立する

項 (term): Prologの主役となる構文



項 (term): Prologの主役となる構文

複合項の糖衣構文

`[1, 2]` = `.(1, .(2, []))`

`1 + 2` = `'+'(1, 2)`

`[X | Y]` = `.(X, Y)`

リテラル

`123`

空リスト

`[]`

項の一番外側は「命令 (述語, 条件)」として振る舞う

```
halt.
```

項の一番外側は halt というアトム。
haltは命令として振る舞う。

```
append([1, 2], [3], X).
```

項の一番外側は append という複合項。
appendは命令として振る舞う。

最上位以外、どこに変数を書いてもいい

```
?- append(X, Y, [1, 2]).
X = [],
Y = [1, 2] ;
X = [1],
Y = [2] ;
X = [1, 2],
Y = [] ;
false.
```

Append自体もPrologコードで書ける

```
% 空リストを足しても変わらない
append([], X, X).
% 第一引数が1要素増えると、結果リストも1要素増える
append([X|Y], Z, [X|W]) :- append(Y, Z, W).
```

Prologの実行モデル

Prologの理想

- 条件を宣言的に書く
- 解を全て探してくれる

宣言的なメンタルモデル

Prologの現実

- 命令列を並べる
  - 命令順が重要
- 単純なバックトラックによる探索

命令的なメンタルモデル

命令順によって結果が異なる (命令的な側面)

```
?- Y = [], append(X, Y, Y).
Y = X, X = [] ;
false.
```

```
?- append(X, Y, Y), Y = [].
ERROR: Stack limit (1.0Gb)
exceeded
```

Prologの実際

- （宣言的な視点）Prologは論理式ソルバーである。
- （命令的な視点）Prologは「単一化」と「バックトラック」を備えた命令的プログラミング言語である。
  - 単一化 = 自由変数を含む等式を解く
  - バックトラック = 分岐の片方を試し、探索が終わったら状態を巻き戻して、分岐のもう一方も試す

- 項 (term) は関数の積み重ね
- この関数は一般の関数ではなく、互いに重なり合わない性質を持つ
  - 単射かつ相互排他
  - コンストラクタとも呼ばれる
- Prologの項としての 1 + 3 は 2 + 2 とは等しくない
  - 関数名(+)が同じで、引数も同じときだけ同じとみなす

単一化

- 単一化 (unification) = 項に関する連立方程式を解く問題
- ある種の唯一解 (最汎単一化子; MGU) を求めることができる
  - 解がない(矛盾)ケースもある
- たとえば f(X, a) = f(b, Y) なら X = b, Y = a

- **Prologでは1つの述語を実行するのに複数の実行経路がある**
- 普通の分岐と異なり、どちらに進めばいいのか開始時点ではわからないことがある
  - たとえば append/3 は append([], X, X). と append([X|Y], Z, [X|W]) :- append(Y, Z, W). の2つの分岐があるが、第一引数が自由変数だとどちらに進めばいいかわからない

- **Prolog**では1つの述語を実行するのに複数の実行経路がある
- 普通の分岐と異なり、どちらに進めばいいのか開始時点ではわからないことがある
- 両方の分岐を試すためにバックトラックを行う

バックトラックの手順

- 分岐1に進む
- 分岐1が終わったら、分岐地点まで巻き戻す
- 次の分岐を同様に試す

- 組み込みのバックトラックを持つ言語では、「巻き戻される副作用」と「巻き戻されない副作用」を区別する必要がある
  - これはWeb開発におけるトランザクションの取り扱いとも似ている(トランザクション中に外部APIを叩くと巻き戻せない問題)
- 単一化は「巻き戻される副作用」として扱う
  - そのためにも単一化とバックトラックを抱き合わせて言語処理系で取り扱う必要がある

- バックトラックは非決定性計算を実現する方法のひとつ
  - Listモナドと呼ばれることもある
- 非決定性計算自体も副作用のひとつ

- 数学的には、非決定性計算とは1つの値のかわりに集合を返す関数ということになる
  - f: A → B のかわりに f: A → P(B) を考える (Pは冪集合)
  - 圏論的に言うと、冪集合モナドのKleisli射を考えている
- 「集合を返す関数」=「2項関係」
  - f: A → P(B) を考えるのではなくR ⊆ A × B を考える
- 2項関係では定義域と終域が対称の関係になる
  - Kl(P) = Rel

- 定義域と終域が対称の関係 → 逆計算の正当化
  - append(X, Y, [1, 2]). のような計算をする発想

Prologの応用

Prologの応用について以下を紹介

- 初期の人工知能
- **Datalog**
- 型推論

- PrologはLispとともに、初期の人工知能研究で使われた
  - 逆問題を解くなど、現在の人工知能にも通じる共通点もみられる
  - プログラムとデータの同型性という意味ではPrologとLispにも似た面がある
- 比較的最近の応用ではWatsonの自然言語処理に使われていたらしい
  - ここ数年で深層学習ベースの自然言語処理が強くなってしまったので今後同様の応用が出る見込みは少なそう
- 今でもProlog = 人工知能という印象は強いかも

Datalog

- **Prolog**から計算的な側面を排除したサブセット
  - 本スライドの parent / sibling の例がこれに当たる
- 一種のグラフデータベースのように使うことができる
- **SQL**に影響を与えたとも言われている

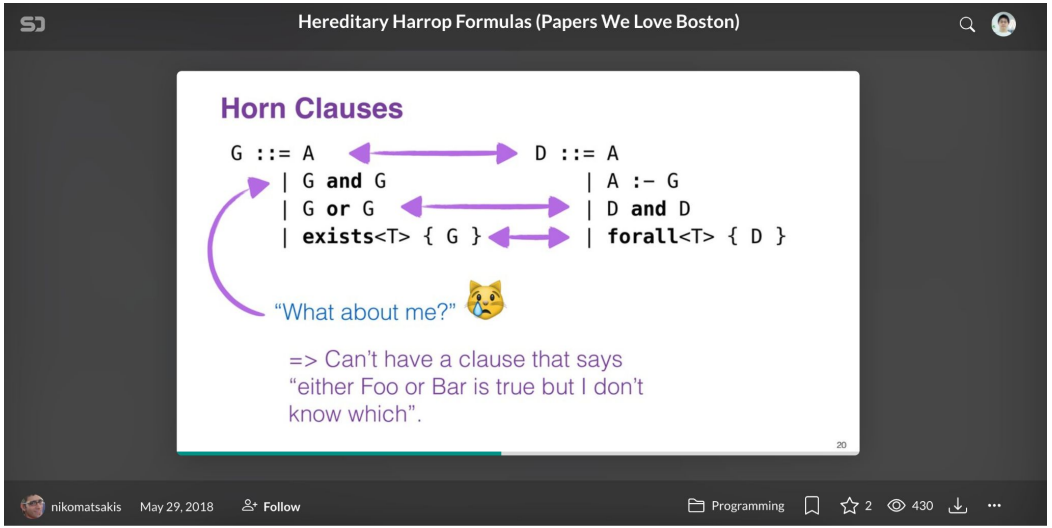
型推論

- **Hindley-Milner**型の型推論では、型変数を置くことで宣言順によらない型推論を可能にしている。
  これは**Prolog**の単一化と同じもの
- 型クラスを持つ**Haskell**や**Rust**の型推論では、複数の実装のなかから制約を充足できるものを探索する。
  これは**Prolog**のバックトラックと対応

型推論

- 型推論器をPrologに見立てることで、Rustの型システムを強化する試み

https://speakerdeck.com/nikomatsakis/hereditary-harrop-formulas-papers-we-love-boston



まとめ

今回話すこと

- **Prolog**は論理型プログラミング言語
- 古くは人工知能として持て囃された
- その実態は「単一化」と「バックトラック」を持つ特殊な手続き型言語
- 現代では型推論器など記号処理タスクの実行エンジンとして大いに役に立っている