核言語

Plato は核言語に型付ラムダ計算という計算体系を使用している。核言語の文法を以下に示す。

```
term :  x                                 variable
        \x. term                          abstraction
        term term                         application
        \X. term                          type abstraction
        term @type                        type application
        'let' x = term 'in' term          let expression
        'fix' term                        fix combinator
        term.x                            projection
        [x '=' term]                      record
        x [term]                          tag value
        'case' term 'of' [(n, term)]      case expression

type :  X                                 type variable
        T -> T                            type of functions
        'forall' X:K. T                   universal type
        \X:K. T                           operator abstraction
        T T                               operator application
        [x ':' type]                      type of record
        [(x, [type])]                     variant type
```

バージョン 0.1.0.0 時点では、この文法に識別子型を追加して、相互再帰を実現しているが、これは以降のバージョンで取り除かれる可能性がある。なぜなら、名前を参照する型システムでは大域的情報を保持しておく必要があり、型検査が煩雑になってしまうからである。相互再帰関数の実装を応用した、同型再帰による再帰関数の実装を検討している。

Command

核言語において、プログラム全体は Commands というデータ型に変換される。 Commands はインポートされたモジュールと、トップレベルで定義された型コンストラクタ、データコンストラクタの記録、そして、main 関数からなる。main 関数は、Let 式で表され、トップレベルの関数が束縛に、ソースプログラムに現れる main 関数の項が Let 式の本体に当てられる。このように翻訳することにより、相互再帰関数が扱いやすくなる。

変数は抽象された識別子の相対位置として自然数で表される。これを de Bruijn インデックスという。 トップレベルで定義された型コンストラクタやデータコンストラクタは文脈（Context）に大域的に保持され、プログラム中のどこでも参照することができる。

再帰関数

関数はトップレベルや Let 式の束縛として以下のように記述される。

```
iseven : Nat -> Bool;
iseven n = case n of {
    Zero -> True;
```

```
    Succ n' -> isodd n';
};

isodd : Nat -> Bool;
isodd n = case n of {
    Zero -> False;
    Succ n' -> iseven n';
};

main : Bool;
main = iseven (Succ (Succ Zero));
```

これは核言語において以下のように表現される。

```
let
    r = fix (\ieio: {iseven: Nat -> Bool, isodd: Nat -> Bool}.
        {iseven = \ case n of {
            Zero -> True;
            Succ n' -> isodd n';};
         isodd n = case n of {
            Zero -> False;
            Succ n' -> iseven n';};
        };
    )
 in r.iseven (Succ (Succ Zero));
```