プログラミング言語処理系論
2022　問題集

講義の何回目に出した問題かは、ローマ数字で表していますから
参考にしてください

- 課題1　Javaの言語仕様を入手し、同様の解析を行なってみよ。仕様はここ↓

https://docs.oracle.com/javase/specs/jls/se18/jls18.pdf

Javaは、Oracleが仕様を保守する、典型的なオブジェクト指向言語である。オブジェクトを核として、パッケージ等、プログラム単位がどのように定義されているか記述せよ。

(II)

- 課題1’　Ecmascriptの言語仕様を入手し、同様の解析を行なってみよ。

Ecmascriptは、Web Browserを実行環境と想定するスクリプト言語である。言語のコンセプトが何かを中心に、仕様書の構造を解析せよ

(II)
Rubyについて

- 課題1’’　Rubyの言語仕様を入手し、同様の解析を行なってみよ。JISが定められているが、入手しやすい仕様はITC-LMSにおいています

- JIS X3017:2013
- Rubyの最新は3.1.1(2022.2版) JISとの関係は不明（recall Javascript）。規格の保守は破棄されているらしい
- Rubyもコミュニティベースで保守されている典型的なオブジェクト指向言語である。ここでは、FortranやJavaと比較して、言語の定義がどの程度成熟しているか併せて観察せよ

(II)
課題1（1’, 1’’）についての注意

- 解析すべきプログラミング言語としての特徴のポイントはFortranとほぼ同じです
  - 並列性を含む実行制御における特徴。Javaならthreadとメモリモデル
  - データ実体の表現。特にオブジェクト指向ならばクラスを中心にした解析

  - Syntaxの定義には、本質的にBNFが使われていることを観察してください

(II)
もしくは

- https://docs.oracle.com/javase/specs/jvms/se18/jvms18.pdf（2022）
- （課題２）上のドキュメントを以下の観点から要約せよ［この課題は、6月にもう一度出すので今回は無視してもよい］
  - VMの仮想機械としての構造
  - Constant pool, method area等、実行に特徴的な領域の特定と、実行開始までの流れ(startup)

(IV)
Pythonではどうか

- コンセプトは？
- データ構造の設計
- オブジェクトの設計
- 制御構造の設計
- モジュールとインポート

（Pythonには、正式な言語の定義が与えられていない
（利用者マニュアルのみ））

（問題3） Pythonの適当なサブセットをさだめその言語定義を与え、仕様書にまとめよ。30ページ程度の簡略版で構わない。

（ヒント） 仕様書の章組みは、他のオブジェクト指向言語で仕様が与えられているもの（Java, Ruby等）を参考にするとよい。Annotationやdecorationを無視した簡略版で構わない（ここらへんの捨て方は少し経験が必要…）

(IV)
もっとおそろしい言語があってな

- Fortranのごく初期においては
  - 関数呼び出しにおいて、関数コールごとの実行環境（フレーム）は関数ごとに固定
  - グローバルな変数は存在せず、EQUIVALENCE文で関数コールごとに対応を指定
- （課題*：考古学） Fortranの関数コールにおけるフレームの作り方について調査せよ。Fortranは、「再帰」を理解できないプログラマを大量に養成したといわれる（半分デマ）が、実際Fortranでは特に指定しない限り再帰が書けない。その理由をフレームの作り方と関連付けて述べよ

(IV)

- Perl5の実行について
  （課題4） PerlのB::Conciseモジュールを利用して、以下についてレポートせよ
  (1) 適当なプログラムに対してのソースとパース木の対応（-src）の観察
  (2) 実行に際して必要となるデータ構造（スタック、フレーム、ヒープ）
  - (2)について無理する必要はありません

(V)

- symbol_update()
- ここまでがベースとなる処理

- 次からのsymtable_visit_*()
  - Source code traversalの典型
  - （問題 5） Pythonのsymtable_visit_*()を読み、その内部を以下の観点から説明せよ
    1. ASTの構造に従ったtraversalの仕方
    ２．各端点における処理の準同型に従った処理
    ３．全体としてなされる処理の記述

(VI)

- （課題6）
  手近にあるコンパイラ　on HW CPU をひとつ対象にし、calling conventionとフレームを解析せよ。この時に
  以下のことに注意せよ
  （１）コンパイラ・OS・CPUを明記すること
  （２）解析の手法を明らかにすること
  （３）calling conventionは、呼び出し側と呼び出される側の約束であるが、そのときに呼び出される側から呼び出し側へも約束が存在することに注意せよ

(VI)

- （課題6’）
  手近にあるコンパイラ　on a VMをひとつ対象にし、calling conventionとフレームを解析せよ。この時に以下のことに注意せよ
  （１）コンパイラ・VMを明記すること
  （２）解析の手法を明らかにすること
  （３）calling conventionは、呼び出し側と呼び出される側の約束であるが、そのときに呼び出される側から呼び出し側へも約束が存在することに注意せよ
- 注意：CPythonのソースコードを解析して、フレームがどのように構築されるかまで明らかにする必要はない（すればもっと良い）

(VII)
課題7

- 自分で言語を定義してみよ
  - 仕様書を書け（これが一番大事）
  - High Level Conceptは「新しい概念のない電卓である」でもかまわない
  - ただし、変数は扱えるようにすること
  - 関数コールをデザインすること
  - Lexical analysisに関係するコードは自分で書くこと
  - Parse したら、ASTは生成できるようにすること
  - どちらか
    - 評価系 (eval)を書いてください
    - 自分で適当にVMを設計して、それをターゲットにしたコンパイラを作ってもよい
  - スクラッチからやるのが面倒であるならば、教材その他を適当に拡張してかまいません。
    - 教材は、最低限のことはできる（最低限のことしかできない）ようにしています
    - 大きい（本格的な）言語は、読み込むのに一苦労ということはよくわかります

(VIII)

- （課題8）
  JAVA　VMの仕様を読み、以下の観点から整理して特徴を記せ。
  手っ取り早い方法は、2章を要約することだろう
- JVMの仕様はここ
  https://docs.oracle.com/javase/specs/jvms/se18/jvms18.pdf

（１）VMの命令があつかうデータ
（２）メモリ（スタックとヒープ）管理
（３）スレッド管理
（４）オブジェクト管理

(VIII)

- （課題8’）ceval.c を以下の観点から要約せよ。要約は、JVMの整理の仕方に倣ってみよ（以下のスライドで概要を示す）JVMの2章程度の内容でよい。
  - CPython VMの仮想機械としての構造
  - Pythonの実行をサポートするためのフレーム管理、メソッド呼び出し、ループ実行等の命令セットの特徴

(IX)
問題9

- Stack machineのVMを一つ設計せよ
  - 命令セットはODC.zip中のopcode.hでよい。
    - 例外処理しない。スレッドの管理しない。
    - ここまで簡単化すればvmgenは（古いツールだが）役に立つかもしれない
  - stack segmentとdata segmentを定義すること
  - 各命令を実行したときのstack/data segmentの状態変化を記述すること
    - OP_CALL とOP_RETURNについては詳述すること
    - この記述に従ってvmgenでVMを作成するとなおよい
  - コンパイラを作る必要はない

(IX)
MetaInterpreter（最適化の対象）

- ループ
  - Pythonで書いたループ
    - RPythonのinterpreterの中でも、dispatcher loopの中でのループになっている
    - ということで、Pythonで書いたループではなく、RPythonのループを対象にした
      - PyPy用に最適化ルーチンを書くのではなく、RPython用に最適化ルーチンを書くことで、概念を一般化＝MetaInterpreter
  - （問題10）以下の論文を要約せよ。3—7章だけで構わない
    C.F. Bolz “Meta-Tracing Just-in-Time Compilation for Rpython,” PhD thesis, Heinrich-Heine-Universitat Dusseldorf, 2012.

(IX)
Loop Unrolling Challenge

- （問題 11）以下のプログラムが、今使われている代表的なCPU上で、unrollingを使って速くなるかどうかを検証せよ
  - 行列積（double/int）
  - キャッシュ最適化の効果と分別できるように、十分小さなループで検証すること（サイズを次第に大きくして言って、キャッシュの効果が現れ始める直前でやめることはできるか？）
  - 小さいループでは、実行時間の計測が不安定になるが、それを安定させるためにはどうしたらよいか考えてみよ
  - CPUとコンパイラを明記すること
  - コンパイラによっては、勝手にunrollする最適化を行うものがあるので、アセンブリ出力、又はコンパイラのメッセージ出力を確認して、unrolling段数を正しく把握することが必要

(X)
課題12

- Pythonでは、ast_opt.cとcompile.cに最適化が書かれている
  - 講義では、主要な部分にしか触れていないということを前提に、2つのファイル中の最適化を網羅的に説明せよ
  - さらに最適化を進める可能性について論ぜよ。

(X)
課題13

- 以下の論文のうち、一つを選択し、その要約をせよ。

  Lopes, N. et.al.: Provably Correct Peephole Optimizations with Alive, PLDI2015

  Mullen E. et.al.: Verified Peephole Optimizations for CompCert, PLDI2016

- （一般論だが）論文の要約は実は難しい。この場合、最低限方法論の詳述をすること。数ページになるはずである

(XI)

- （問題14）以下の命令列に対してvalue numberingを行え。授業中で概説したアルゴリズムを完成させよ。できるならばプログラムを書いてみよ。

```
    g=x+y
h=u-v
i=x+y
x=u-v
u=g+h
v=i+x
w=u+v
```

```
add i0 i1 i2
sub i3 i7 i5
add i0 i1 i6
sub i3 i7 i0
add i2 i5 i3
add i6 i0 i7
add i3 i7 i8
```

- （問題14の続き）以下のコードを考える。
  def f(a,b) {return ((a+b)+3)*(3+(b+a))};
- RTLは以下のようになるだろう（名前a → i0, b → i1）

  add i0 i1 r2

  mov 3 r3

  add r2 r3 r4

  mov 3 r5

  add i1 i0 r6

  add r5 r6 r7

  mul r4 r7 r8

  return r8

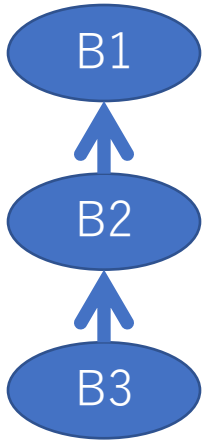
- この時、value numberingを行え。rで始まる変数はreturnした後使用しないという仮定を置くと、命令のいくつかを削除することができる。これをやってみよ。（dead code elimination）

問題14の注意

- アルゴリズムをプログラムとして実装する必要はない。（プログラムを書いて、それが正しく動けばなおよい）ただし
  - Commutativeな演算については考慮の対象とすること
  - Quantityに対して最終的に割り当てられる変数名を決定するアルゴリズムを含むこと
  - Dead code eliminationやredundant assignment eliminationを行うアルゴリズムを含むこと

(XI)
Domの性質

- Dominator Tree
  - 2つのノードSとTに関し、SがTをdominateし、かつTがSをdominateすることはあり得ない
  - Dominateという関係は木構造を作る（この木構造をdominator treeという）



(XI)
課題15

- 前ページの命題を証明せよ：
  - 2つのノードSとTに関し、SがTをdominateし、かつTがSをdominateすることはあり得ないことを証明せよ
  - Dominateという関係は木構造を作ることを証明せよ

(XI)
課題16：プログラム

- 以下の（疑似）プログラムを考える。

```
while () {
  if (S1) {S2};
  while (S3) {
     while () {
        switch (S4) {
        case 0: continue;
        case 1: S5; break;
        default: S6; break;
        }
        if (S7) break;
     }
     switch (S8) {
     case 0: continue;
     case 1: S9; goto Label;
     default: return S10;
     }
  }
Label:
}
```

（課題16続き）
次のデータ構造を考える

BB[1] = {succ = {2,3}, pred={9}}

BB[2] = {succ = {3}, pred={1}}

BB[3] = {succ = {4}, pred={1,2,4,8}}

BB[4] = {succ={3,5,6},pred={3,7}}

BB[5] = {succ={7}, pred={4}}

BB[6] = {succ={7}, pred={4}}

BB[7] = {succ={4,8},pred={5,6}}

BB[8] = {succ={3,9,10}, pred={7}}

BB[9] = {succ={1}, pred={8}

BB[10] = {succ={}, pred={8}}

(XI)
課題16

(1) 前述のデータ構造が、SnがBB[n]に対応するフローグラフになっていることを確認せよ。

(2) S1がentry pointであるとして、各ブロックのdominatorを計算するアルゴリズムを設計せよ（プログラムまでは書く必要がない）。さらに、それに従って、前ページのスライドにあげたフローグラフの各ブロックのdominatorの集合を列挙せよ。

(3) 上の結果を用いてdominator treeを計算するアルゴリズムを設計せよ（プログラムまでは書く必要がない）。さらに、それに従って前ページのスライドにあげたフローグラフのdominator treeを計算せよ。

(4) backedgeをすべて挙げ（4個）、それに対応する（自然）ループ構造を元の疑似プログラム上で示せ。

(XII)
課題17

- 以下のコードについてglobal value numberingを行え。以下の手続きを書けば、プログラムを書く必要はない。（プログラムを書けばなおよい）
  - 各quantityについてreaching definitionを明らかにすること
  - redundant code eliminationを行うこと。ただし、aからtは、このコードの後でも使用されうるとする
  - SSA変換をする必要はないが、しても構わない。

```
[a, b, c, d,i, m, s, t are given]
while (x < 100) {
 if (x == i) {
    i = c × b;
   m = i + 4;
   a = c;
 } else {
   d = c;
   i = d × b;
   s = a × b;
   t = s + 1;
 }
 x = a × b;
 y = x + 1;
}
print i;
```

課題17についてのNote

- 各変数のUseに対して、quantityを計算する
  - 当該変数のUseに対し、reaching definitionの集合Rを計算する
  - If (|R| == 1) { /* unique definition */
    そのdefのquantityをそのまま使う
    else
    新たなquantityをそのUseに割り当てる

(XII)
課題18

- GCCの最適化ルーチンではdf-core.c, df-problems.c, df-scan.cで一般的なソルバーを提供している。このルーチン群を解析し、それらを利用して実際にどのような問題が解かれているか列挙せよ。そのうち、reaching definitionを例にとり、gen, kill, out, inがどのように表現されているか記述せよ。

(XII)
課題19

(1) 説明されたLoop Invariant Hoistingの概要からアルゴリズムを導き出し、SSAを用いて表現せよ

(2) SSAを用いてDerived Induction Variableを見つけるアルゴリズムを考えよ。Strength Reductionを適用できるとなお良い。

(3) (2)に関し、配列のデータを順にアクセスするときの配列のインデックスの計算の方法が手近にあるコンパイラでどのように実装されているか調査せよ。最適化オプションと組み合わせること