プログラミング言語処理系論（２０１４）全体講評

2014/08/05
佐藤周行

1. 全体講評

今回のレポートは、全体としては今までにない特徴をもっていました。すなわち、問題１１，１２，１３（およびそのコンビネーション）を解いてきた人がほぼ全員ということです。採点は楽でした。ただ、「この講義では触れない」と言っていた１１を解いてきた学生さんのレポートの質が前と比較して向上してきたのに気が付きました。学部レベルの教材がまともになったんでしょうね。次回からはある程度ものを教えられないと解けないような問題にしましょう（デナイトコウギノイミガナイ）。

１１，１２，１３，１４は、コンパイラ最適化に関係する問題でした。１１については、自分でモデルを作って解析するのは当然のようにできていて、それを実験で検証しているレポートが 2 件ありました。コンピュータサイエンスのスタンダードな方法論がきちんと身についていると思います。実験での検証と同列なのが、１２における「アルゴリズムの補完」をプログラムを組むことで示したレポートが出ました。value numbering は、コンパイラの最適化ルーチンの一つですから、もちろんソースはどこかに存在するわけですが、自分で組めるようになっているのはとてもよい心がけだと思います。今後とも頑張ってください。

この講義の前半で力を入れたことは、VM まで含めて自分で言語処理系を設計、実装するための基礎的な考え方で、「ここまでわかっていれば、自作のスクリプト言語ができてくるかな」と思っていたのですが、本格的な言語を VM まで含めて実装したレポートがでてきました。言語は nano という名前ですでに公開されているようです。教えたりなかったところは「その言語は基本コンセプトとして何を表現するものなのか」についてでしたが、その点もテーマを絞って実現されていました。川勝君。見事だと思います。願わくは言語の実装のレポートが複数出てくるように。誘導も含めて次回以降の佐藤の課題にしましょう。

2. 表彰

佐藤の成績の付け方はとても甘いです。その代わり、優秀なレポートを提出した人を講評時に表彰することで差をつけています。最近は成績表を見る目も厳しいですからね。 そういうわけで特に優秀なレポートを出した人をここにあげて表彰したいと思います。今回はプログラムを書いた人を中心に3位までを出すことにします。漏れた形になった方は、MANABAにコメントを載せておきましたので参考にしてください。

1位　川勝孝也君　2位　川尾太郎君　3位　村岡恒輝君

履歴書に書いて威張れるかどうかはよくわかりませんが、適当にご自慢ください。

3. 個別講評

ここでは、解いてもらえなかった問題を含め、問題全体を概観することにしましょう。６－９，１１，１２，１３についてはちょっとだけ詳しく論じます。

問題１　Java か Ruby の言語仕様を要約してみろという問題でした。いい加減厚いですからね、やらなくても OK です。ただ、Java に限らず、言語仕様を最初から最後まで読んでおくとあとあとプログラミング言語に対する土地勘のようなものが働くようになります。その意味でも Ruby の言語仕様は適当に薄くてお勧めです。書き方もしっかりしてる。

問題２　XML のパーサを書いてみろ、という問題です。大昔（佐藤が学生の頃）は、パーサの作成は一大テーマだったのですが、現在はツールが整備されているので楽です。その風潮に逆らって「手で書いてみろ」というのは、やはり支持をえられないようです。

問題３　Bison（Yacc）の生成するオートマトンの解析をしてみろという問題ですが、これも万人向けの問題ではなかったようですね。

問題５　Perl5 の実行コード（パース木に毛の生えたもの）を見てみろという問題でした。言語の実行系を作ろうと思う人は、一度は refer しておくべき構造だと思います。

問題６　言語を設計・実装せよという問題でした。「言語仕様」を与えることと「実行環境」を作ることの両方が求められます。言語仕様の書き方の具体的なところは講義ではなかなか触れられませんでしたが、「基本コンセプト」を定めることと「文法」と「意味」を定めることが必要です。ある程度の規模になると「ライブラリ関数」を仕様の中で与えることも求められます。実行環境については、VM を決めて、それをターゲットにしたコードを吐き出すという形にするとコンパイラを作っている気分になります。そうでなければ問題５のような形式でしょうか。ここまでの講義では触れていなかったのですが、最適化ルーチンをどう構成するかは…まぁいいか。Advanced なトピックですね。とにかく、大昔にバリアと思われていた「トークンの切り分け」「パーサを書くこと」は劇的に楽になっています。この文書に触れた人は、一度トライしてみてください。

問題７　問題６に関連して、変数のバインディングの問題です。

問題８　手近のコンパイラをさわってみて、calling convention を解析してみよという問題です。これをやるにはコンパイラの取説を見るのがまずは第一歩、次にはアセンブリ言語の理解。

問題９　関数呼び出しのフレームを設計せよという問題です。問題７，８からの続きの問題です。問題６と合わせると、まともな言語の実装がここで一段落ということです。川勝君はここまでたどり着いたということですね。

問題１０　Java VM の仕様を解析せよという問題です。問題１と同じでいい加減厚いですからやらなくていいのですが、最近は Jython とか Scala とか、JVM で動く言語が出てきているというのは、JVM が VM らしくなったということですかね。同じことを MS CLI や Parrot でやってみてもよいですが、実用性を求めるなら JVM は一度深く掘ってみてもよいかもしれません。

問題１１　ループ再構成の問題とループアンローリングの問題です。「やっても正しく動く」ことの証明はまぁ、できるからよいとして（1990 年代に発行された成書を見れば十分です。その意味でもはや古典的）性能モデルをどう作るかですね。最近のプロセッサは、1990 年代のものとは違い、(ある程度) ふんだんにハードウェア資源を投入していますから、古い教科書では説明しきれない現象（性能劣化が起きるはずなのに、なかなか観測されない等）もあります。たとえば、現在ではキャッシュの way 数が昔より増えていますから、キャッシュコンフリクトによる性能劣化が表に出にくくなっていることもあるようです。そういう意味でも、(古典的な教科書をもとにしたモデル化でも）一度実測してモデルを検証するということは大切です。行列積では、単純なキャッシュのモデルで性能の差が説明できます。ループアンローリングについては、繰り返し内の命令数が増えることで、命令スケジューリングの機会が増えることが一義的ですが、Intel のチップが相手だと、定量的な解析はなかなか難しいですね。

問題１２　value numbering の問題です。プログラムを書きましょう。式の numbering では、可換な演算子への配慮が必要です。また、式番号の探索には一般にハッシュ表を使うとよいと一般的に言われていますが、global value numbering ならともかく、ローカルな value numbering では、効果がないかもしれません。でも、negative な予想は一回実験して検証しないうちは何もいえませんね。手で value numbering をやった人はきちんとできていたと思います。発想をプログラムにするというのは CS の基本的な方法論ですから、その意味で一歩先に進むことを期待します。

問題１３　natural loop の見つけ方です。みなさん、dominator tree を作って、バックエッジを見つけるという正統的な方法で解いていたのは喜ばしいことです。これもプログラムを書くとなおよいのですが、まぁよしとしましょう。

問題１４　データフロー方程式を「手で」解く問題です。問題１５とあわせて解くとよいと思います。一度手で解くと、ソルバーのプログラムを作るときに気を付けなければならないことが見えてきます。

問題１５　問題１４のソルバーを作るときに必須なのが停止性の証明です。さらにソルバーが最適解を出すという証明があると安心できます。ソルバーの実装の時はみなこの程度まで気にしているということで、もし論文を書くようなことがあったら気を付けるとよいと思います。

問題１６　Knoop の論文を要約しろという問題でした。1992 年だしなぁ、もういいや。

問題１７　SSA の応用例について、典型的なものを考えてみようということです。今や、SSA は GNU のものを含め、どこでも出てきますから（Intel Compiler が SSA based と言っていますが、ソースが非公開なのでよくわからないですね)、一度触れておくというのもよいのではないでしょうか。

以上