CS 263-Spring 2008

Design and Analysis of Programming Languages

- Instructor: Dana Scott
  <danas@eecs.berkeley.edu>
- TA: Sridhar Ramesh <sramesh@berkeley.edu>

Last edited 13 February 2008

Suggested Projects

PROJECT 0. Topics for discussion. (no credit)

Start collecting short, succint answers to these questions. We will find a way to post answers on a discussion group.

Think also of some *additional* general questions about programming languages. Send Prof. Scott your suggestions to `<danas@eecs.berkeley.edu>`.

(1) ***What good*** are programming language?

(2) Which are the ***good*** programming languages?

(3) How does one ***choose*** a programming language?

(4) How does one ***design*** a programming language?

(5) Do we need ***more*** programming languages?

(6) What is the role of a ***compiler***?

(7) Can programming be made ***independent*** of the machine platform?

(8) What are ***virtual machines***?

(9) What is the difference between ***distributed*** and ***parallel*** computing?

(10) How do we know that a program is ***correct***?

(11) How do we know that a compiler is ***correct***?

(12) What use are ***types*** in programming languages?

(13) Do compilers need to ***know about*** types?

(14) Why are versions of **FORTRAN** and **COBAL** still in use?

(15) Are some programming languages better than others for ***team work***?

(16) How valid is the "equation" **`Algorithms + DataStructures == Programs`** these days?

(17) What is **Literate Programming**?

(18) Do some languages promote ***good programming*** or ***beautiful programming*** better than others?

(19) What is the difference between ***abstract syntax*** and ***concrete syntax*** ?

(20) Can a language be both a ***functional language*** as well as an ***imperitive language*** ?

PROJECT 1. What good are combinators? (**** 4 stars)

■ Assumptions

• Suppose it were possible to implement well the computations with combinators on an abstract (= virtual) machine — and to do this on many platforms.

• Suppose it were also possible to build a generic compiler for your favorite language producing code for the virtual machine.

■ Claim

• Then you would have both flexibility and a clear-cut handle on ***correctness*** and ***verifiability***.

■ Project

• Defend or dispute this claim by searching on the web for: ***combinator abstract machine*** and ***categorical abstract machine***.

■ Notes

• There are many Google hits for both searches. And these hits will lead to other searches.

• Take into account that more current work concentrates on ***typed languages***.

PROJECT 2. Are there better pairing functions? (*** 3 stars)

■ Background

• In class, in order to give a quick introduction to Gödel numbers it was remarked that the pairs $\langle p, q \rangle$ of integers (including **0**) were in a one-one correspondence with the ***positive*** numbers of the form $2^p (2q + 1)$.

• The problem with using the exponential encoding of pairs to encode syntax is that the numbers grow too quickly: even small combinators have astronomically large Gödel numbers.

■ Project

• Find a better encoding of pairs to use with Gödel numbers so that the growth is reduced.

• Express the method in terms of combinators.

• Is there an easily stated bound on the Gödel number of a combinator in terms of the length of the combinator?

■ Hint

• Pairs of integers can be counted either by growing ***triangles*** or ***squares***. Either method should give reasonable results.

PROJECT 3. How fast is Ackermann? (*** 3 stars)

■ Background

Recursion equations for the function discovered by Ackermann can be given as:

$$\mathbf{ack}(0, n) = n + 1$$
$$\mathbf{ack}(m + 1, 0) = 1$$
$$\mathbf{ack}(m + 1, n + 1) = \mathbf{ack}(m, \mathbf{ack}(m + 1, n))$$

■ Project

• Explain briefly why is **ack** a ***total*** recursive function.

• Find in the literature (or on the WWW) a proof that **ack** is ***not*** primitive recursive.

• (Extra credit) Find a combinator for **ack** computing on the Church numerals.

PROJECT 4. How to simulate lists? (* 1 star)

■ Background

In class we were able to calculate with combinators and pairs of objects using the combinators **pair**, **left**, **right**.

■ Project

Find combinators to simulate ***finite lists***. Each list should have a ***length***. If possible set things up so if **L** represents the list $\langle A_1, A_2, \ldots, A_n \rangle$, then **L[cnum[i]]** reduces to A_i. Perhaps, **L[zero]** could reduce to the length

of the list, but other conventions are possible.

PROJECT 5. Closure under composition (* 1 star)

■ Background

• It is easy to see that the ***monotone operators*** on sets of integers are closed under composition.

■ Project

• Work out the proof that ***continuous operators*** (of any number of arguments) are closed under composition.

■ Hint

• Try a simple case such as $\Phi(\Psi(X, Y), \Theta(X, Y))$ first.

■ Extra credit (* 1 extra star)

• Can you use some results about the semantics of combinators in $\mathbb{P}$ to prove this?

PROJECT 6. Application as an operator (* 1 star)

■ Background

• In showing that $\mathbb{P}$ is a model of the rules of combinators, we had to define ***application*** and ***λ-abstraction*** using sets of integers.

■ Project

• Using the definitions, show explicitly that not only is $U[X]$ ***continuous***, but we have

$$\lambda X.\, \Phi(X, Y_0, Y_1, \ldots, Y_{n-1})[X] == \Phi(X, Y_0, Y_1, \ldots, Y_{n-1})$$

for every continuous operator Φ.

PROJECT 7. S as an operator (* 1 star)

■ Project

• Prove that $S = \lambda X.\, \lambda Y.\, \lambda Z.\, X[Z][Y[Z]] \in \mathbb{Rcc}$.

PROJECT 8. Partial recursive functions (** 2 stars)

■ Background

• If $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is a ***partial function*** mapping (some of the) integers to integers, then there is a obvious related ***continuous operator*** $\Phi:\mathbb{P}\to\mathbb{P}$ defined by:

$$\Phi(X) = \{f(x) \mid x \in X\}.$$

• Because we take $n = \{n\}$ for $n \in \mathbb{N}$, note that $\Phi(n) = f(n)$, if $f(n)$ is ***defined*** — ***otherwise*** the value is ϕ.

■ Project

• Prove that every ***partial recursive function*** $p : \mathbb{N} \rightharpoonup \mathbb{N}$ there is a set $P \in \mathbb{RE}$ such that $P[n] = p(n)$ for all $n \in \mathbb{N}$.

PROJECT 9. The $20 Prize Problem (**** 4 stars)

■ Project

• Find the ***fewest*** and ***neatest*** **recursive sets** $R_1, \ldots, R_n$ such that ***all*** **recursively enumerable sets** can be generated from them by the binary $U[X]$ operator.

■ Adjudication

• The class will vote on the best solution. Teamwork is permitted.

Time Frame

• Solutions due by class time on **Wednesday, 20 February, 2008**.

PROJECT 10. Other fixed points (* 1 star)

■ Project

• Suppose $\Phi(X)$ is a continuous operator, and $A \subseteq \mathbb{N}$ is such that $A \subseteq \Phi(A)$. Is there a ***fixed point P*** of Φ with $A \subseteq P$? And is there a ***least such***?

■ Extra credit (* 1 extra star)

• Using the continuous operator $A \bigcup X$, find a way of defining P also using some ***combinators***.

PROJECT 11. Sequences and fixed points (* 1 star)

■ Background

• We defined the set of sequence numbers of sequences with all terms belonging to a set X as $X^* = \{s \mid \S\, s \subseteq X\}$. This is a continuous operator.

■ Project

• Explain using least fixed points the meaning of this operator : $Y^\infty = Y \bigcup (Y^\infty)^*$.

PROJECT 12. Finding other models (*** 3 stars)

■ Project

• Show that there are ***many other*** models for the **`crules`** between $\mathbb{RE}$ and $\mathbb{P}$ also using the same $U[X]$ operation.

PROJECT 13. Some set-theoretical properties (** 2 stars)

■ Project

• ***Prove*** the following two theorems:

Theorem. For $U, V, X \in \mathbb{P}$ and for continuous operators Φ and Ψ we have:

$$U[X] \bigcup V[X] = (U \bigcup V)[X] \text{ and } \lambda X.(\Phi(X) \bigcup \Psi(X)) = \lambda X.\Phi(X) \bigcup \lambda X.\Psi(X) .$$

Theorem. For for continuous operators Φ and Ψ we have:

$$\lambda X.(\Phi(X) \bigcap \Psi(X)) = \lambda X.\Phi(X) \bigcap \lambda X.\Psi(X) .$$

• Give a ***counter-example*** with finite U and V to show that the first equation for unions ***does not*** hold for intersections.

■ Extra credit (* 1 star)

• Show (quickly) why both theorems can be generalized to operators of ***more variables***.

• Show that the theorem about unions can be generalized to ***infinite unions***.

• Why can we ***not*** generalize the intersection result to ***infinite intersections***?

PROJECT 14. Simutaneous equations (** 2 stars)

■ Project

• Given two ***computable*** and ***continuous*** operators $\Phi(X, Y)$ and $\Psi(X, Y)$, each of ***two arguments***, show that the ***least*** solutions to the pair of equations:

$$X = \Phi(X, Y) \text{ and } Y = \Psi(X, Y)$$

are indeed in $\mathbb{RE}$.

• Does the method generalize to ***more variables***?

■ Hint

• Take advantage of combinators **`pair`**, **`left`**, **`right`** as interpreted by our semantics as sets in $\mathbb{RE}$.

PROJECT 15. Formal-language theory (** 2 stars)

■ Background

• With our approach to coding sequences, the set $\mathbb{N}$ of non-negative integers can be viewed as well as the set $\mathbb{N}^*$of all ***finite sequences*** of integers.

• To relate our constructions to what is usually done in formal-language theory (hereafter called FLT), we should identify first an ***alphabet***. This is easy, as we just take the one-termed sequences as our (infinite) alphabet. In fact, define:

$$\mathbb{A} = \{\langle n \rangle \mid n \in \mathbb{N}\} = \langle \mathbb{N} \rangle = \{2n + 1 \mid n \in \mathbb{N}\}.$$

And, to have a mnemonic we define the set of ***strings*** over our alphabet:

$$\mathbb{S} = \mathbb{N}^* = \mathbb{N}.$$

• Now the above use of * is ***not*** what is usually meant in FLT. We need to say that for all sets $X \subseteq \mathbb{S}$ we can define a new operator:

$$X^\star = \{\langle\rangle\} \bigcup \left(X^\star \frown X\right).$$

This defines the least set of strings containing the given set X and closed under concatenation. We can now say $\mathbb{S} = \mathbb{A}^\star$, as is normal in FLT.

• To jibe further with standard FLT notation, we should also write $\epsilon = \langle\rangle$, and, for $\sigma, \tau \in \mathbb{S}$, we may write $\sigma\tau = \sigma \frown \tau$. Hence, for sets $X, Y \subseteq \mathbb{S}$, we use (just for this project) the shorthand notation $XY = X \frown Y$. We also use the shorthand $\sigma X = \{\sigma\} X$.

■ Project

• For given $\sigma, \tau \in \mathbb{S}$, prove that there is a ***unique solution*** to this fixed-point equation:

$$X = \{\sigma, \tau\} \bigcup \sigma X \bigcup X \tau.$$

• For fixed-point equations like this, explain on ***general principles*** we have developed why the least fixed points must be $\mathbb{RE}$. (Of course, automata theory can show why many such equations have ***recursive*** solutions.)

• Does this ***pair*** of equations have a ***unique pair*** of solutions:

$$X = \{\sigma\} \bigcup XY \text{ and } Y = \{\tau\} \bigcup YX \ ?$$

And ***what are*** these sets?