Compiler Construction

Exam

Mon, Jun 25, 2018, 13:30–16.30

- Please make sure that you write your name and student number on each sheet of paper that you hand in. On the first sheet of paper, indicate the total number of sheets handed in.
- This is a closed-book exam: it is not allowed to consult lecture notes, books, etc. Whenever reference material is required it is given as part of the exam.
- Always explain the reasoning behind your answers, even when not explicitly asked for.
- Each question includes the amount of points which can be obtained when done correctly, adding up to a total of 100.

---

1 Program analysis

Question 1. (5 points) There is a strong relationship between the ordering $\sqsubseteq$ and the join operator of a lattice, $\sqcup$. Define each in terms of the other. ∎

A modification of the Available Expression analysis (let us use AEV as the abbreviation for the analysis) from the slides is that we are only interested in expressions that are available in a particular variable. A non-trivial expression a is available in x at label ℓ if it has been evaluated and assigned to x on all paths leading to ℓ and if the values of x and the variables in the expression a have not changed since then.

Question 2. (14 points) Write down the complete specification of the analysis. That is give the equations for $AEV_{\mathrm{N}}(\ell)$ and $AEV_{\mathrm{X}}(\ell)$ for the blocks in the While language, and any auxiliary functions that may be necessary or useful (like $kill_{AEV}$ and gen_{AEV}) (if this is easier for you, you may use Haskell to phrase them). Do not forget to explain your equations.
Hint: it is important to carefully decide what is the best lattice to use.

2 Attribute Grammars

Given is the following definition of an AG datatype:

```
DATA E
  | Add
    b1 : B
    b2 : B
  | Leaf
    b : B
  | Split
    e1 : E
    e2 : E
DATA B
  | Bin
    b1 : B
    b2 : B
  | One
  | Zero
DATA R
  | Root
    e : E
```

Here the datatype B represents binary numbers, and E is a datatype that represents some kind of binary tree with Bs in them. R functions as the root for such a complete tree. In the questions below you are *not* allowed to use copy rules or USE statements.

Question 3. (9 points) Define attributes and the associated code that deliver the decimal representation of the binary numbers in the root of each value of type B. Note: this code only involves the B datatype.

Question 4. (7 points) Define attributes and the associated code that computes the largest of all such decimal values occuring in a value of type R, and then propagates this value down to all occurrences of a B in the value of type E (there is no need to propagate the value to occurrences of Bs inside a B).

3 Type inference

The rule for generalization in Hindley-Milner is as follows:

$$\frac{\Gamma \vdash_{\text{UL}} e : \sigma \qquad \alpha \text{ not free in } \Gamma}{\Gamma \vdash_{\text{UL}} e : \forall \alpha . \sigma} \text{ [gen]}$$

Question 5. (6 points) Give a (part of) a derivation tree that shows that without the side condition on α, the rule would be able to derive an invalid type. ∎

Question 6. (5 points) The rule and its counterpart the instantation rule are both not syntax directed. How is this issue addressed in an implementation of Hindley-Milner? ∎

4 Annotated Type Systems

For function application and definition inference rules are given below from the specification of a usage analysis.

$$\frac{\widehat{\Gamma} \sim_{\mathsf{UA}} \widehat{\Gamma}_1 \bowtie \widehat{\Gamma}_2 \quad \widehat{\Gamma}_1 \vdash_{\mathsf{UA}} t_1 :^{\varphi_1} \widehat{\tau}_2{}^{\varphi_2} \to \widehat{\tau}^{\varphi} \quad \widehat{\Gamma}_2 \vdash_{\mathsf{UA}} t_2 :^{\varphi_2} \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\mathsf{UA}} t_1\ t_2 :^{\varphi} \widehat{\tau}}$$

$$\frac{\widehat{\Gamma}[x \mapsto^{\varphi_1} \widehat{\tau}_1] \vdash_{\mathsf{UA}} t_1 :^{\varphi_2} \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\mathsf{UA}} \lambda x.\, t_1 :^{\varphi} \widehat{\tau}_1{}^{\varphi_1} \to \widehat{\tau}_2{}^{\varphi_2}}$$

Question 7. (4 points) What is the lattice of annotations that this analysis works on, and what do the values in the lattice represent? ∎

Question 8. (8 points) Explain the rule for applications in some detail, and don't forget to consider the role of the leftmost predicate. ∎

Question 9. (6 points) The above rule for lambda-abstraction does not work in certain cases. Explain under which circumstances the rule is unsafe and why. ∎

Question 10. (4 points) The problem can be fixed by introducing a so-called containment relationship. Explain what the purpose of this relationship is and how it helps address the problem (you may but do not have to define the relationship). ∎

5 Miscellaneous

Question 11. (10 points) Give a concise explanation of how Two-Space Copying does garbage collection. Also, give one advantage and one disadvantage of this algorithm as compared to the other two algorithms discussed in the course. ∎

Question 12. (11 points) Given is the following desugared Haskell program:

```
let
  i x = x
  c x y = x
in
  c (i 2) (i 3)
```

(a) Give the graph representation for the body of the let. Be careful in making sharing explicit!

(b) Reduce the expression to weak head normal form by reducing it with normal-order reduction (as was demonstrated graphically in the slides).

(c) What is the major difference between your reduction and an applicative order reduction? ∎

Question 13. (11 points) Specify a (precise) Galois Connection between

$$L = \mathbf{Lab}_* \to \mathcal{P}(\mathbf{Var}_* \to \mathbf{N})$$

and

$$M = \mathbf{Lab}_* \to \mathbf{Var}_* \to \mathcal{P}(\{\text{odd}, \text{even}\}) \ ,$$

where $\mathbf{N}$ is the set of natural numbers (including 0), and the others are all as they were in the slides. If you use general constructions like the total function space combinator, then specify this clearly.

You need not prove that your construction is in fact a Galois Connection, but you do have to give a suitable example value in L and give the corresponding abstract value by showing how the value is abstracted by your abstraction function.