Compiler Construction

Exam

Wed, Jun 28, 2017, 9:00–12.00

- 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.
- Plan the making of the exam. Some questions involve coding; it is easy get stuck in a detail and spend too much time on it.
- Each question includes the amount of points which can be obtained when done correctly, adding up to a total of 100.
- Note: there are 14 separate questions divided over 5 sections.

---

1 General questions to warm up

Question 1. (6 points) The compiler is often considered to consist of a front-end and back-end. What kind of aspects of compilation are handled by the former, and what kind by the second? Name two compilation phases for each. ∎

Question 2. (6 points) Defend and/or attack the statement "Static analyses always have to approximate". ∎

2 Attribute Grammars

Consider the terms of the (untyped) lambda-calculus,

$$t \in \mathbf{Tm} \quad \text{terms},$$

defined by

$$t ::= x \mid \lambda x.\, t_1 \mid t_1\ t_2.$$

and its use of variable symbols,

$$x \in \mathbf{Var} \quad \text{variables}.$$

Using the UUAG system, we define the following grammar for representing terms:

```
{type Var = String}
data Tm
  | Var  x :: {Var}
  | Lam x :: {Var} t1 :: Tm
  | App  t1 :: Tm t2 :: Tm.
```

Question 3. (6 points) Define a Haskell function $fv :: \mathsf{Tm} \to [\mathsf{Var}]$ that collects the free variables of a lambda-term. (The resulting list may contain duplicates.) ∎

An alternative notation for lambda-terms, using so-called *De Bruijn indices*, was invented by the Dutch mathematician Nicolaas de Bruijn and allows for a nameless representation of variables. Its syntax, with terms ranged over by $\widehat{t}$,

$$\widehat{t} \in \widehat{\mathbf{Tm}} \qquad \text{nameless terms,}$$

is given by

$$\widehat{t} ::= n \mid \lambda\, \widehat{t}_1 \mid \widehat{t}_1\, \widehat{t}_2.$$

Each natural number in a nameless term denotes the occurrence of a variable and indicates how many binders there are between that occurrence and the binder for its corresponding variable. For example, the following lambda-terms,

$$\begin{aligned}
&\lambda x.\, x\\
&\lambda x.\, \lambda y.\, x\\
&\lambda x.\, \lambda y.\, \lambda z.\, x\ z\ (y\ z)\\
&\lambda f.\, \lambda g.\, \lambda x.\, f\ (\lambda y.\, g\ y)\ x,
\end{aligned}$$

are namelessly represented by, respectively,

$$\begin{aligned}
&\lambda\ 0\\
&\lambda\ \lambda\ 1\\
&\lambda\ \lambda\ \lambda\ 2\ 0\ (1\ 0)\\
&\lambda\ \lambda\ \lambda\ 2\ (\lambda\ 2\ 0)\ 0.
\end{aligned}$$

Let us use the following grammar to represent nameless terms:

```
{type Nat = Int}
data Tm^
  | Var^  n :: {Nat}
  | Lam^ t1 :: Tm^
  | App^ t1 :: Tm^ t2 :: Tm^.
```

Question 4. (10 points) Give attribute definitions for translating from the conventional notation of lambda-terms into a representation using De Bruijn indices:

```
attr Tm
  syn deBruijn :: Tm^.
```

Introduce and define any auxilliary attributes you deem necessary. ∎

3 Program analysis

Question 5. (6 points) Given a partial order $\sqsubseteq$ on (L, L), where we may write $x \sqsubseteq y$ instead of $(x, y) \in \sqsubseteq$. Give the formal condition(s) for a partial order to also be a lattice.

What distinguishes a lattice from a complete lattice (you may answer this formally, or by appealing to intuition)? ∎

Question 6. (12 points) As explained in the book, Very Busy Expressions analysis aims to discover for each program point which expressions are guaranteed to be used again before any of the variables in the expressions are redefined.

Given are these equations for the Very Busy Expressions analysis:

$$VB_{\mathrm{X}}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \mathtt{final}(S_\star) \\ \bigcap\{VB_{\mathrm{N}}(\ell') \mid (\ell', \ell) \in \mathtt{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$VB_{\mathrm{N}}(\ell) = (VB_{\mathrm{X}}(\ell) - kill_{VB}(B^\ell)) \cup gen_{VB}(B^\ell)$$

Now, answer the following questions:

- define the analysis lattice,
- give the kill and gen sets for the case of assignments,
- explain how you can tell from the equations whether the analysis is backward or forward
- and whether it is may or must.
- Are we looking for solutions with the smallest sets, or the largest? ∎

Question 7. (6 points) Does the formula in the previous part also work for programs that start and end with a While loop? Explain. ∎

4 Type systems

Consider a small, implicitly typed functional programming language,

$$\begin{array}{rcll} t & \in & \mathbf{Tm} & \text{terms,} \\ x & \in & \mathbf{Var} & \text{variables,} \\ n & \in & \mathsf{Nat} & \text{naturals} \end{array}$$

defined by

$$\begin{array}{rcl} t & ::= & n \mid \mathtt{false} \mid \mathtt{true} \mid x \mid \lambda x.\, t_1 \mid t_1\ t_2 \mid \mathbf{let}\ x = t_1\ \mathbf{in}\ t_2\ \mathbf{ni}, \\ & \mid & \mathbf{if}\ t_1\ \mathbf{then}\ t_2\ \mathbf{else}\ t_3\ \mathbf{fi} \end{array}$$

and assume we subject it to the Hindley-Milner (HM) typing discipline. Its type language is then built from type variables.

$$\alpha \in \mathbf{TyVar} = \{\alpha, \beta, \gamma, \cdots\} \qquad \text{type variables.}$$

and stratified into types and type schemes, or monomorphic and polymorphic types respectively.

$$\begin{array}{lcll} \tau & \in & \mathbf{Ty} & \text{types} \\ \sigma & \in & \mathbf{TyScheme} & \text{type schemes,} \end{array}$$

given by

$$\begin{array}{lcl} \tau, \nu & ::= & \mathsf{Nat} \mid \mathsf{Bool} \mid \alpha \mid \tau_1 \to \tau_2 \\ \sigma & ::= & \tau \mid \forall \alpha . \sigma_1 . \end{array}$$

In the slides a unification algorithm was provided with the following type,

$$\mathcal{U} : \mathbf{Ty} \times \mathbf{Ty} \rightharpoonup \mathbf{TySubst},$$

and that is used by algorithm W.

Question 8. (8 points) Give the substitution that results for each of the following calls to $\mathcal{U}$ (and *error* if the substitution fails):

(i) $\mathcal{U}(\alpha \to \beta \ , \ \beta \to \alpha)$,

(ii) $\mathcal{U}(\alpha \to \mathsf{Bool} \to \alpha \ , \ \gamma \to \gamma \to \mathsf{Nat})$,

(iii) $\mathcal{U}(\alpha \to \mathsf{Bool} \to \beta \ , \ \gamma \to \alpha)$,

(iv) $\mathcal{U}((\mathsf{Nat} \to \alpha) \to \beta \to \mathsf{Bool} \ , \ \beta \to \alpha \to \gamma)$. ∎

Assume an environment Γ in the type rules for the HM system with

$$\begin{array}{l} [\mathit{id} \mapsto \forall a.\, a \to a, \\ \ \mathit{ii} \mapsto \mathsf{Nat} \to \mathsf{Nat}, \\ \ \mathit{const} \mapsto \forall a. \forall b.\, a \to b \to a, \\ ] \subset \Gamma \end{array}$$

Question 9. (10 points) For the HM system, give the derivation tree for:

$$\mathbf{let}\ f = \lambda i \to \lambda x \to i\ (i\ x)\ \mathbf{in}\ \mathit{const}\ (f\ \mathit{ii}\ 4)\ (f\ \mathit{id}\ \mathit{True})$$

Clearly indicate which type rules have been applied where, with names like VAR, LET, APP, ABS, NAT, TRUE and FALSE. Maybe you can come up with clever abbreviations and such to make the tree more consise. That is fine, as long as can understand it. ∎

The limit of what HM can infer is illustrated by the following example, which the HM typing discipline cannot type:

$$\begin{array}{ll} \mathbf{let}\ f = \lambda i \to \mathit{const}\ (i\ 3)\ (i\ \mathtt{true})\ \mathbf{in} & \\ \mathbf{let}\ x = f\ \mathit{id} & \mathbf{in}\ x \end{array}$$

Question 10. (6 points) Why is this the case? Where in the HM type system does the typing of the example go wrong? ∎

5 Annotated Type Systems

Question 11. (6 points) Annotations for side-effect analysis are defined as follows:

$$\varphi ::= \{!\pi\} \mid \{\pi :=\} \mid \{\mathbf{new}\pi\} \mid \varphi_1 \cup \varphi_2 \mid \emptyset$$

Explain what these annotations represent. ■

Question 12. (6 points) Annotations are often considered *equal modulo* UCAI. What does that mean? ■

Question 13. (6 points) Explain (in detail) the following type and effect rule:

$$\frac{\widehat{\Gamma} \vdash_{\mathrm{SE}} e_1 : \widehat{\tau}_2 \xrightarrow{\varphi_0} \widehat{\tau}_0 \;\&\; \varphi_1 \qquad \widehat{\Gamma} \vdash_{\mathrm{SE}} e_2 : \widehat{\tau}_2 \;\&\; \varphi_2}{\widehat{\Gamma} \vdash_{\mathrm{SE}} e_1\ e_2 : \widehat{\tau}_0 \;\&\; \varphi_0 \cup \varphi_1 \cup \varphi_2}\ [\mathrm{app}]$$

■

Question 14. (6 points) Give a rule for subeffecting for this analysis. ■