Algorithms

Exercise 2: Strategies for algorithm design

1 Hutton's Razor strikes back

1. *Type inference* describes the process of assigning types to previously untyped terms of a formal language. A type may be seen as an approximation of the value of a term. We use types to try and catch errors in a program before it is run. For example, recall the definition of Hutton's razor:

   $$e = e + e \mid n \in \mathbb{N}$$

   We extend this language with boolean values and conditionals:

   $$e = e + e \mid n \in \mathbb{N} \mid b \in \mathbb{B} \mid \textbf{if } e \textbf{ then } e \textbf{ else } e$$

   Now let us define the evaluation function for this language:

   $$\begin{array}{lcl}
[\![\cdot]\!] & : & e \to (\mathbb{N} + \mathbb{B})_\perp \\
[\![n]\!] & = & \textbf{inl } n \\
[\![b]\!] & = & \textbf{inr } b \\
[\![e_0 + e_1]\!] & = & \begin{cases} \textbf{inl } (x + y) & \text{if } [\![e_0]\!] = \textbf{inl } x \text{ and } [\![e_1]\!] = \textbf{inl } y \\ \perp & \text{otherwise} \end{cases} \\
[\![\textbf{if } e_0 \textbf{ then } e_1 \textbf{ else } e_2]\!] & = & \begin{cases} [\![e_1]\!] & \text{if } [\![e_0]\!] = \textbf{inr } \mathit{true} \\ [\![e_2]\!] & \text{if } [\![e_0]\!] = \textbf{inr } \mathit{false} \\ \perp & \text{otherwise} \end{cases}
\end{array}$$

   Note that the codomain of $[\![\cdot]\!]$ is $(\mathbb{N} + \mathbb{B})_\perp$. $\mathbb{N} + \mathbb{B}$ represents a *tagged union*. Given two sets, A and B, the tagged union of the two is defined as:

   $$A + B = \{ \textbf{inl } a \mid a \in A \} \cup \{ \textbf{inr } b \mid b \in B \}$$

   In other words, a tagged union is simply a union where the elements are tagged with either **inl** or **inr** to indicate which set they originate from.

   The subscript $\perp$ indicates that the set is "lifted". This simply means that we include $\perp$ (read as *bottom* or *undefined*) in the set. In other words:

   $$(\mathbb{N} + \mathbb{B})_\perp = \{ \perp, \textbf{inl } 0, \textbf{inl } 1, \ldots, \textbf{inr } \mathit{true}, \textbf{inr } \mathit{false} \}$$

   To summarise, the co-domain of the evaluation function is the set of $\perp$, all natural numbers tagged with **inl**, and all boolean values tagged with **inr**. We use $\perp$ in the definition of the evaluation function to indicate failure: *i.e.* things we cannot evaluate.

   However, according to the grammar for e, a perfectly valid expression in this language would be the following:

   $$\textbf{if } 4 \textbf{ then } \mathit{false} \textbf{ else } 8$$

This expression does intuitively not make much sense. If we apply the evaluation function to it, it will evaluate to $\bot$. To avoid such cases, we will add a type system to our language. We first need to define suitable types. A good start would be one type for each kind of value in our language:

$$\tau = \textbf{nat} \mid \textbf{bool}$$

We know that a well-formed expression will either evaluate to a natural number or to a boolean value. Since types are approximations of the values of expressions, our definition of τ seems adequate.

We now wish to define a binary relation $\vdash e : \tau$ which relates expressions e to types τ. For example, we want to say that all numbers in our language are of type **nat**: $\vdash n : \textbf{nat}$. Enumerating all of these relations would be impossible, however, so we declare rules according to which types are assigned different forms of expressions instead. We call this the *deductive system*. Each rule in this system is presented in the following way:

$$\frac{\textit{condition1} \qquad \textit{condition2} \qquad \dots}{\textit{conclusion}}\ \text{RULE-NAME}$$

These rules are like implications in logic: if all the conditions are true (shown above the bar), then we can draw the conclusion which is shown below the bar. The full deductive system for our language is now shown below:

$$\frac{}{\vdash n : \textbf{nat}}\ \text{T-VAL} \qquad\qquad \frac{}{\vdash b : \textbf{bool}}\ \text{T-BOOL}$$

$$\frac{\vdash e_0 : \textbf{nat} \qquad \vdash e_1 : \textbf{nat}}{\vdash e_0 + e_1 : \textbf{nat}}\ \text{T-ADD} \qquad \frac{\vdash e_0 : \textbf{bool} \qquad \vdash e_1 : \sigma \qquad \vdash e_2 : \sigma}{\vdash \textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2 : \sigma}\ \text{T-COND}$$

To show how this works, let us prove that the type of **if** *true* **then** $(23 + 42)$ **else** 4 is **nat**. If we think of the expression as a tree, we have a conditional expression at the top. This means that we need to use the T-COND rule first:

$$\frac{\vdash \textit{true} : \textbf{bool} \qquad \vdash (23 + 42) : \textbf{nat} \qquad \vdash 4 : \textbf{nat}}{\vdash \textbf{if}\ \textit{true}\ \textbf{then}\ (23 + 42)\ \textbf{else}\ 4 : \textbf{nat}}\ \text{T-COND}$$

Here we have replaced the meta variables e_0, e_1, and e_2 with the corresponding expressions. Next we need to show that all of the three conditions hold. We work our way from left to right. $\vdash \textit{true} : \textbf{bool}$ holds according to the T-BOOL rule:

$$\frac{\dfrac{}{\vdash \textit{true} : \textbf{bool}}\ \text{T-BOOL} \qquad \vdash (23 + 42) : \textbf{nat} \qquad \vdash 4 : \textbf{nat}}{\vdash \textbf{if}\ \textit{true}\ \textbf{then}\ (23 + 42)\ \textbf{else}\ 4 : \textbf{nat}}\ \text{T-COND}$$

We use T-ADD to show that $\vdash (23 + 42) : \textbf{nat}$:

$$\frac{\dfrac{}{\vdash \textit{true} : \textbf{bool}}\ \text{T-BOOL} \quad \dfrac{\vdash 23 : \textbf{nat} \qquad \vdash 42 : \textbf{nat}}{\vdash (23 + 42) : \textbf{nat}}\ \text{T-ADD} \qquad \vdash 4 : \textbf{nat}}{\vdash \textbf{if}\ \textit{true}\ \textbf{then}\ (23 + 42)\ \textbf{else}\ 4 : \textbf{nat}}\ \text{T-COND}$$

T-ADD has two conditions, both of which we can prove using T-VAL:

$$\frac{\dfrac{}{\vdash \textit{true} : \textbf{bool}}\ \text{T-BOOL} \quad \dfrac{\dfrac{}{\vdash 23 : \textbf{nat}}\ \text{T-VAL} \quad \dfrac{}{\vdash 42 : \textbf{nat}}\ \text{T-VAL}}{\vdash (23 + 42) : \textbf{nat}}\ \text{T-ADD} \qquad \vdash 4 : \textbf{nat}}{\vdash \textbf{if}\ \textit{true}\ \textbf{then}\ (23 + 42)\ \textbf{else}\ 4 : \textbf{nat}}\ \text{T-COND}$$

Lastly, we use T-ADD one more time to prove that $\vdash 4 : \mathbf{nat}$:

$$\dfrac{\dfrac{}{\vdash \mathit{true} : \mathbf{bool}}\ \text{T-BOOL} \quad \dfrac{\dfrac{}{\vdash 23 : \mathbf{nat}}\ \text{T-VAL} \quad \dfrac{}{\vdash 42 : \mathbf{nat}}\ \text{T-VAL}}{\vdash (23+42) : \mathbf{nat}}\ \text{T-ADD} \quad \dfrac{}{\vdash 4 : \mathbf{nat}}\ \text{T-VAL}}{\vdash \textbf{if}\ \mathit{true}\ \textbf{then}\ (23+42)\ \textbf{else}\ 4 : \mathbf{nat}}\ \text{T-COND}$$

The type inference problem can now be formalised as: given an arbitrary expression e, can we assign a type τ to it such that the typing relation holds:

$$\vdash e : \tau$$

(a) In a language of your choice, implement suitable data structures to represent expressions and types in our language. You should also implement the evaluation function.

(b) Turn the deduction system described above into an algorithm and implement it in a language of your choice.

(c) What is the time complexity of your type inference algorithm?

(d) *Type checking* is a similar problem to type inference. It asks: given an expression e and a type τ, is τ a suitable type for e? Show how type checking can be implemented by reducing it to type inference.

(e) Suppose that we want to lift the restriction on conditionals and allow each branch to return a potentially different type. Discuss how this could be implemented in the type system. Describe all changes you would make to the type syntax and the typing rules.