Homework 01

98-317 Hype for Types

Due: September 3, 2019 23:59 ET

1 Introduction

1.1 Preliminaries

Welcome to the first homework of 98-317 Hype for Types.

We have a course website, `https://hypefortypes.github.io`. This will be the place we distribute homeworks and slides. It also contains our course policy, which we ask you to read and uphold in its entirety.

We have a Piazza, `https://piazza.com/class/jzrgppibs613tb`. This will be the place for you to ask questions about the course, the homeworks, the lectures, etc. You can also help answer other people's questions. The instructors will be active on Piazza, and will use it to announce things about the course.

We will probably have an Autolab soon. This will probably be the place where homeworks are handed in. If and when we get an Autolab, it will be posted on the course website, and we will make an announcement on Piazza. If the strategy for handing in homeworks turns out to not include using Autolab, we will make an announcement to this effect on Piazza.

1.2 Getting this homework

1. Navigate to the course website
2. Download the appropriate tar file
3. Move the tar file to an empty directory
4. Extract the contents of the tar file with `tar -xf handout.tar`

1.3 Handing in this homework

1. Type `make handin.tar`
2. Either upload `handin.tar` to Autolab or send an email

- to azdavis@andrew.cmu.edu,
- with subject "98-317 HW01",
- with attachment `handin.tar`

1.4 Layout of this homework

After getting the homework, the structure of the homework directory should be the following.

- `Makefile` defines targets for `make`. Type `make` for information. As an example, type `make handin.tar` to create a tar file ready for handing in.
- `handin` contains the files which will be handed in.
- `pdf` contains this writeup.
- `support` contains support code, which is freely usable in your implementation. The `.sig` files serve as documentation for the support code.
- `tests/Sandbox.sml` contains any code you want. When a repl is created with `make repl`, code defined in the sandbox will be loaded into the repl, along with all of the code in `support` and `handin`.
- `tests/Tests.sml` contains tests. Run the tests with `make test`.

2 `synthtype`

In this homework, you will be translating inference rules that we give you into code that implements these inference rules. Specifically, you will be writing cases of the `synthtype` function, which takes in an expression in the E# language, and outputs a type for this expression, if it has one. Consult section 3 for more information about the E# language. Section 3.3 is most relevant for implementing `synthtype`.

2.1 `Str`, `Plus`, `Len`

In `handin/Statics.sml`, in `synthtype`, define the cases for `Str`, `Plus`, and `Len`. Each one of these cases is very similar to another respective case which we have already defined. For instance, the `Plus` case is very similar to the `Concat` case, which has already been defined.

2.2 `Let`

In `handin/Statics.sml`, in `synthtype`, define the case for `Let`.

Unlike the previous cases, in this case, the context must be modified. In the inference rules, Γ is the context which contains information about the variables in scope and their types. In the code, we represent this context with a variable named `ctx`.

The operations permitted on a context are detailed in `support/CTX.sig`. Notably, given

- an existing context `ctx`,
- a variable `x`, and
- a type `t`,

a new context that contains all of the information from the old `ctx`, plus the information that the variable `x` has type `t`, can be created with the code

```
Ctx.add x t ctx
```

To reiterate:

```
val ctx: Ctx.t = ... (* a context *)
val x: string = ... (* a variable *)
val t: Typ.t = ... (* a type *)
val newCtx: Ctx.t = Ctx.add x t ctx
```

Note that, in this code example, `ctx` is not "updated" to hold the information that `x` has type `t`. Instead, a new context named `newCtx` is created. This new context holds all the information that was in `ctx`, plus the new information.

Because this case is harder than the previous cases, it is optional. However, we still encourage you to give it a try, and remember you can ask the instructors for help if you are stuck.

3 The E# Language

"E#" is pronounced "Eee Sharp".

3.1 Types

τ ::=	int	integers
	$\mid \mathsf{str}$	strings

3.2 Expressions

e ::=	$x \mid y \mid z \mid \ldots$	variables
	$\mid 1 \mid 2 \mid 3 \mid \ldots$	integer literals
	$\mid$ "" $\mid$ "foo" $\mid$ "bar" $\mid \ldots$	string literals
	$\mid e_1 + e_2$	integer addition
	$\mid e_1 \wedge e_2$	string concatenation
	$\mid \mathsf{len}(e_1)$	string length
	$\mid \mathsf{toStr}(e_1)$	conversion to string
	$\mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$	variable binding

3.3 Statics

We've given names to each of the inference rules for your convenience.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \textsc{Var} \qquad \frac{n \in \mathbb{Z}}{\Gamma \vdash n : \mathsf{int}}\ \textsc{Int} \qquad \frac{}{\Gamma \vdash \text{"..."} : \mathsf{str}}\ \textsc{Str}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \textsc{Plus} \qquad \frac{\Gamma \vdash e_1 : \mathsf{str} \qquad \Gamma \vdash e_2 : \mathsf{str}}{\Gamma \vdash e_1 \wedge e_2 : \mathsf{str}}\ \textsc{Concat}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{str}}{\Gamma \vdash \mathsf{len}(e_1) : \mathsf{int}}\ \textsc{Len} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int}}{\Gamma \vdash \mathsf{toStr}(e_1) : \mathsf{str}}\ \textsc{ToStr} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}\ \textsc{Let}$$

3.4 Dynamics

The dynamics shown here have been implemented for you in `support/Dynamics.sml`. It is presented here only to complete the formal definition of E#, and involves concepts and notation not discussed in class.

The judgement being defined here are

- $e\ \mathsf{val}$, which holds when the expression e is a value and cannot be evaluated further.
- $e \mapsto e'$, which holds when e can be evaluated in 1 step to the new expression e'.

Together with the statics, we can prove certain safety properties about the E# language:

Progress If $\Gamma \vdash e : \tau$, then $e\ \mathsf{val}$ or $e \mapsto e'$.

Preservation If $\Gamma \vdash e : \tau$ and $e \mapsto e'$, then $\Gamma \vdash e' : \tau$.

We use the notation $[e/x]e'$ to mean "the expression e is substituted in for wherever the variable x appears in the expression e'". We often pronounce $[e/x]e'$ as "e for x in e'".

The dynamics rules might be a bit confusing, given that we didn't talk at all about dynamics in the lecture. The especially confusing dynamics rules are the ones where the arguments to the function-like constructs ($+$, $\wedge$, len, and toStr) are values. The idea is, when the arguments are values, we want to take a step to "the result." To emphasize, the notation in those rules isn't the greatest, so it might be confusing. If you're interested in what's going on here and want to know more, ask us!

$$\frac{n \in \mathbb{Z}}{n \text{ val}} \textsc{Int-Val} \qquad \frac{}{\text{“...”} \text{ val}} \textsc{Str-Val} \qquad \frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \textsc{Plus-LHS}$$

$$\frac{e_1 \text{ val} \qquad e_2 \mapsto e_2'}{e_1 + e_2 \mapsto e_1 + e_2'} \textsc{Plus-RHS} \qquad \frac{e_1 \text{ val} \qquad e_2 \text{ val} \qquad e_1 = n_1 \qquad e_2 = n_2}{e_1 + e_2 \mapsto (n_1 + n_2)} \textsc{Plus-Res}$$

$$\frac{e_1 \mapsto e_1'}{e_1 \wedge e_2 \mapsto e_1' \wedge e_2} \textsc{Concat-LHS} \qquad \frac{e_1 \text{ val} \qquad e_2 \mapsto e_2'}{e_1 \wedge e_2 \mapsto e_1 \wedge e_2'} \textsc{Concat-RHS}$$

$$\frac{e_1 \text{ val} \qquad e_2 \text{ val} \qquad e_1 = \text{“...”} \qquad e_2 = \text{“...”}}{e_1 \wedge e_2 \mapsto \text{“.....”}} \textsc{Concat-Res} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{len}(e_1) \mapsto \mathsf{len}(e_1')} \textsc{Len-Arg}$$

$$\frac{e_1 \text{ val} \qquad e_1 = \text{“...”} \qquad |\text{“...”}| = n}{\mathsf{len}(e_1) \mapsto n} \textsc{Len-Res} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{toStr}(e_1) \mapsto \mathsf{toStr}(e_1')} \textsc{ToStr-Arg}$$

$$\frac{e_1 \text{ val} \qquad e_1 = n}{\mathsf{toStr}(e_1) \mapsto \text{“n”}} \textsc{ToStr-Res} \qquad \frac{e_1 \mapsto e_1'}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mapsto \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2} \textsc{Let-Arg}$$

$$\frac{e_1 \text{ val}}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mapsto [e_1/x]e_2} \textsc{Let-Res}$$