# 15-150 Assigment 2

Jonathan Li
jlli
Section S
May 21, 2016

---

**Task 2.1**

---

This code does not typecheck. For the function `bopit`, the return type is type `real`, while its input `c` is also of type `real`. However, according to the function declaration for `bopit`, `bopit` always evaluates to the value given by the function `squareit` on `(c + 1.0)`. However, the function `squareit` has a return type of `int`. This is because all three function declarations lie in the same scope, and the most recent function declaration of the function named `squareit` refers to one of type `real` $->$ `int`.

---

**Task 2.2**

---

(a) In line (10), `i` is replaced with the real 5.0, as `i` was just declared as that value in the scope of the innermost `let/in/end` expression.

(b) In line (13), `p` is substituted with the most recent binding of the variable `p`, which in the scope of the innermost `let/in/end` expression refers to the input `p : int` to the function `generate`. Since we are evaluating expressions with respect to the function call in line (21), `p` refers to the variable `r` in the highest scope of the code, which is declared in line (1) as the value `4 : real` So in line (13), `p` is substituted with `4 : real`.

(c) In line (15), `a` refers to the most recent binding of the variable, which takes place in line (13):

```
val a : real = a - real p
```

Now, the `a` in line (13)'s value declaration refers to the value binding of `a` in line (11) to the value `temp * (real r)`. The last value bound to the variable `temp` took place in the highest scope of the expression in line (4) to `p - 1.0 => 2.0`. The value `r` in line (11) refers to the input `r` of the function `generate`, which in the context of line (21) points to the value `trunc i`, where the value `i` is defined in line (2) to be `2.0 : real`. Thus, `r` in line (11) evaluates to 2.0, and the value of `a` in line (11) evaluates to `(2.0 * 2.0) => 4.0`. In line (13), if we take the value of `p` to be the input supplied to the function `generate` in line (21), which points to the value `r` in the highest scope of the code, defined in line (1) as `4 : int`, then the value of a is:

```
a => (4.0) - (real 4)
a => 0.0
```

Thus, in line (15), `a` is substituted with the value `0.0`.

(d)

$$\begin{aligned}
\texttt{generate(r,trunc i,temp)} &\cong \texttt{2 + (trunc(w + (real t) - a))}\\
&\cong \texttt{2 + (trunc((5.0 * 2.0) + (real 30) - 0.0))}\\
&\cong \texttt{2 + (trunc(10.0 + 30.0 - 0.0))}\\
&\cong \texttt{2 + 40}\\
&\cong \texttt{42}
\end{aligned}$$

---

**Task 2.3**

---

$$\begin{aligned}
\texttt{val r : int} = &\ \texttt{(let val a : real = real (double 3) in 5 + (trunc a) end)}\\
\mapsto &\ \texttt{5 + trunc(real(double 3))}\\
\mapsto &\ \texttt{5 + (trunc(real(2 * 3)))}\\
\mapsto &\ \texttt{5 + (trunc(real(6)))}\\
\mapsto &\ \texttt{5 + (trunc(6.0))}\\
\mapsto &\ \texttt{5 + 6}\\
\mapsto &\ \texttt{1}
\end{aligned}$$

## Task 3.1

$$
\begin{aligned}
\texttt{fact 3} &\cong \texttt{3 * fact (3 - 1)} && [\text{Definition of } \texttt{fact}\text{, equivalence (2)}]\\
&\cong \texttt{3 * (fact 2)} && [\text{math}]\\
&\cong \texttt{3 * (2 * (fact 1))} && [\text{Referential transparency, equivalence (b)}]\\
&\cong \texttt{3 * (2 * (1 * (fact 0)))} && [\text{Referential transparency, equivalence (a)}]\\
&\cong \texttt{3 * (2 * (1* 1))} && [\text{Referential transparency, equivalence (1)}]\\
&\cong \texttt{3 * (2 * 1)} && [\text{math}]\\
&\cong \texttt{3 * 2} && [\text{math}]\\
&\cong \texttt{6} && [\text{math}]
\end{aligned}
$$

By Extensional Equivalence, `fact 3` $\cong$ `6`.

## Task 3.2

Consider `fact ~5`:

$$
\begin{aligned}
\texttt{fact} \sim\!5 &\cong (\sim\!5) * (\texttt{fact} \sim\!6)\\
&\cong (\sim\!5) * (\sim\!6 * (\texttt{fact} \sim\!7))\\
&\cong (\sim\!5) * (\sim\!6 * (\sim\!7 * (\texttt{fact} \sim\!8)))\\
&\cong (\sim\!5) * (\sim\!6 * (\sim\!7 * (\sim\!8 * ...)))
\end{aligned}
$$

Clearly, `fact ~5` loops infinitely. What about `f ~5`?

$$\texttt{f} \sim\!5 \mapsto \texttt{f} \sim\!5 \mapsto \texttt{f} \sim\!5 \mapsto \ldots$$

It appears that `f ~5` loops infinitely as well. According to the definition of equivalence, all expressions that loop infinitely are equivalent to each other. Therefore,

$$\texttt{fact\~{}5} \cong \texttt{f\~{}5}$$

### Task 4.1

**Theorem 1:** $\forall n \in \mathbb{N}$, `sumEven n` $\cong$ `n * (n+1)`.
*Proof:* By induction on $n$.
**Base case:** $n = 0$.
To Show: `sumEven 0` $\cong$ `0 * (0 + 1)`
Proof

| | | | |
|---|---|---|---|
| `sumEven 0` | $\cong$ | $0$ | [Definition of `sumEven`] |
| | $\cong$ | $0 * (1)$ | [math] |
| | $\cong$ | $0 * (0 + 1)$ | [math] |

By extensional equivalence, `sumEven 0` $\cong$ `0 * (0 + 1)`.
**Inductive Step:** $n = k$
Inductive Hypothesis: `sumEven k` $\cong$ `k * (k + 1)`
To Show: `sumEven (k + 1)` $\cong$ `(k + 1)*((k + 1) + 1)`
Proof:

| | | | |
|---|---|---|---|
| `sumEven(k + 1)` | $\cong$ | `2 * (k + 1) + (sumEven k)` | [Definition of `sumEven`] |
| | $\cong$ | `2 * (k + 1) + (k * (k + 1))` | [Inductive Hypothesis, Referential Transparency] |
| | $\cong$ | `2 * k + 2 * 1 + (k * (k + 1))` | [Distributivity of *] |
| | $\cong$ | `(k * (k+1)) + 2 * k + 2 * 1` | [Commutativity of *] |
| | $\cong$ | `k * k + (k * 1 + 2 * k) + 2 * 1` | [Associativity of +] |
| | $\cong$ | `k * k + 3 * k + 2 * 1` | [math] |
| | $\cong$ | `k * k + 3 * k + 2` | [math] |
| | $\cong$ | `(k + 1) * (k + 2)` | [fact (A), Referential Transparency] |
| | $\cong$ | `(k + 1) * ((k + 1) + 1)` | [math] |

By extensional equivalence, `sumEven (k + 1)` $\cong$ `(k + 1) * ((k + 1) + 1)`. Since the base case and the inductive step hold, Theorem 1 must be true.

### Task 4.2

**Theorem 2:** $\forall n \in \mathbb{N}, n \geq 1$, `exp2 n` $\cong$ `g n`.
Theorem 2 is false. This can be shown by the counterexample when $n = 1$. If we let $n = 1$, then:

```
exp2 n ↦ case 1 of 0 => 1 | _=> 2 * exp2(n-1)
       ↦ 2 * exp2(1 - 1)
       ↦ 2 * exp(0)
       ↦ 2 * (case 0 of 0 => 1 | _=> 2 * exp2(n-1))
       ↦ 2 * 1
       ↦ 2
```

So `exp2 1` $\cong$ `2`. When we evaluate `g n` for $n = 1$, we see that:

```
g 1 ↦ case 1 of 1 => 1|2 => 2 | _=> 2 * exp2(n-1)
    ↦ 1
```

So `g 1` $\cong$ `1`. `1` $\ncong$ `2`, so Theorem 2 must be false.