CSCE 330 Fall 2000
EXAMPLE OF DENOTATIONAL SEMANTICS FOR A PROGRAM
WITH A WHILE LOOP
Wednesday 00/9/13


Consider the following program $P$:

```
read(n); fact := 1; i := 1;
while i <= n do
    fact := fact * i;
i := i + 1;
od;
write(fact);
```

If the input stream is empty, then

$$dsem_{PROG}(P, empty) =
\text{out}(dsem_{SL}(\text{read}(n); \text{fact} := 1; i := 1; \text{while} i <= n \text{ do } \text{fact} := \text{fact} * i; i := i + 1; \text{od}; \text{write}(@fact)), \text{init}(empty)) =
\text{out}(dsem_{SL}(\text{fact} := 1; i := 1; \text{while} i <= n \text{ do } \text{fact} := \text{fact} * i; i := i + 1; \text{od}; \text{write}(@fact),
\text{dsem}_{RD}(\text{read}(n), \text{init}(empty)))) =
\text{out}(dsem_{SL}(\text{fact} := 1; i := 1; \text{while} i <= n \text{ do } \text{fact} := \text{fact} * i; i := i + 1; \text{od}; \text{write}(@fact, error)) =
\text{out}(error) = error.$$

Now, assume that the input stream is not empty and it consists of the integer $z$. Then,

$$dsem_{PROG}(P, < z >) =
\text{out}(dsem_{SL}(\text{read}(n); \text{fact} := 1; i := 1; \text{while} i <= n \text{ do } \text{fact} := \text{fact} * i; i := i + 1; \text{od}; \text{write}(@fact)), \text{init}(< z >)) =
\text{out}(dsem_{SL}(\text{fact} := 1; i := 1; \text{while} i <= n \text{ do } \text{fact} := \text{fact} * i; i := i + 1; \text{od}; \text{write}(@fact),
\text{dsem}_{RD}(\text{read}(n), \text{init}(< z >)))) =
\text{out}(dsem_{SL}(\text{fact} := 1; i := 1; \text{while} i <= n \text{ do}$$
\[ fact := fact \cdot i; i := i + 1; od; write(fact), < mem1, empty, empty >, \]
where \( mem1 = \{ < n, z >, < fact, undef >, < i, undef > \} \).

The two assignments change the state in such a way that:
\[ dsem_{PROG}(P, < z >) = \]
\[ out(dsem_{SL}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od; write(fact), < mem2, empty, empty >), \]
where \( mem2 = \{ < n, z >, < fact, 1 >, < i, 1 > \} \).

Let us now use the semantics of the statement list one more time:
\[ dsem_{PROG}(P, < z >) = \]
\[ out(dsem_{SL}(write(fact), dsem_{DO}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od, < mem2, empty, empty >)), \]
Now, we concentrate on the while loop:
\[ dsem_{DO}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od, < mem2, empty, empty >) = \]
\[ if dsem_{BOOL}(i \leq n, < mem2, empty, empty >) = \text{false} \]
then \( < mem2, empty, empty > \)
else \( dsem_{DO}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od, dsem_{SL}(fact := fact \cdot i; i := i + 1; od; write(fact), < mem2, empty, empty >)), \]
\[ if dsem_{BOOL}(i \leq n, < mem2, empty, empty >) = \text{false} \]
then \( < mem2, empty, empty > \)
else \( dsem_{DO}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od, < mem3, empty, empty >) \]
, where
\( mem3 = \{ < n, z >, < fact, mem2(fact) \cdot mem2(i) >, < i = mem2(i) + 1 > \} \).

The last equation above defines recursively a function \( dsem_{DO} \) from \( S \) to \( S \cup \{ \text{error} \} \). The result is the fixpoint of the function. We do not cover the theory of recursive functions in this course, but it should be evident that in this particular case, when the stopping condition \( (i > n) \) holds, \( fact = n! \) and therefore
\[ dsem_{DO}(\text{while} \ i \leq n \ do \ fact := fact \cdot i; i := i + 1; od, < mem2, empty, empty >) = \]
\( < mem4, empty, empty >, \) where
\( mem4 = \{ < n = z >, < fact = z! >, < i = z + 1 > \} \).

We therefore conclude that
\[ dsem_{PROG}(P, < z >) = out(dsem_{WR}(write(fact), < mem4, empty, empty >) = \]
\( out(< mem4, empty, < z! >) = z! \).