The Simple Programming Language
A Simple Language

• The Simple Language is based upon the *Unbounded Register Machine* of Shepherdson and Sturgis (JACM 10, 1963)

• It is a very simple machine, but retains features we recognize from modern traditional languages

• It is constructed to be Turing Complete

• We can construct “macros” that encode more sophisticated features that help us picture how complicated algorithms can be expressed.
Features

• Variables that take on the Natural numbers (0,1,2,3,...) as values.

• Simple assignment statements
  – $x := 0$, $x := \text{succ}(y)$, $x := \text{pred}(y)$
  – Assign values to variables but also perform only the most primitive kind of computation.

• Control
  – Sequencing \{ s1; s2; s3 \}
  – While loop
Macros

• More complicated statements can be built from combining the simple statements in a algorithmic way.

• The idea is to build up a library of “macros” that allow us to write high level programs that “macro-expand” intoi the simple language.
Assignment of one variable to another

• \( y := y \)

\{x := \text{succ}(y); \\
x := \text{pred}(x)\}
Assignment of a constant

- $X := 4$

\[
\{x := 0; \\
x := \text{succ}(x); \\
x := \text{succ}(x); \\
x := \text{succ}(x); \\
x := \text{succ}(x); \}
\]
Addition

• $X := X + Y$

\{i_0 := \text{succ}(y); \\
i_0 := \text{pred}(i_0); \\
\text{while } i_0 \neq 0 \\
\{x := \text{succ}(x); \\
i_0 := \text{pred}(i_0)\}\}
Multiplication

- \( \text{X} := \text{X} \times \text{Y} \)

\[
\begin{align*}
\text{x}_0 & := \text{succ}(\text{x}); \\
\text{x}_0 & := \text{pred}(\text{x}_0); \\
\text{y}_1 & := \text{succ}(\text{y}); \\
\text{y}_1 & := \text{pred}(\text{y}_1); \\
\text{ans}_2 & := 0; \\
\text{while} \ \text{x}_0 \neq 0 : & \\
& \{ \text{i}_3 := \text{succ}(\text{y}_1); \\
& \text{i}_3 := \text{pred}(\text{i}_3); \\
& \text{while} \ \text{i}_3 \neq 0 : \\
& \quad \{ \text{ans}_2 := \text{succ}(\text{ans}_2); \\
& \quad \text{i}_3 := \text{pred}(\text{i}_3); \} \\
& \text{x}_0 := \text{pred}(\text{x}_0); \\
\text{x} & := \text{succ}(\text{ans}_2); \\
\text{x} & := \text{pred}(\text{x}) \\
\end{align*}
\]

\[
\text{mult} \ \text{x} \ \text{y} = \\
\text{do} \ { \text{i} \leftarrow \text{gensym} \ "\text{x}" } \\
& \quad ; \ \text{j} \leftarrow \text{gensym} \ "\text{y}" \\
& \quad ; \ \text{ans} \leftarrow \text{gensym} \ "\text{ans}" \\
& \quad ; \ \text{seq}' [\text{varAssign} \ \text{i} \ \text{x} \\
& \quad , \ \text{varAssign} \ \text{j} \ \text{y} \\
& \quad , \ \text{assign} \ \text{ans} \ \text{Zero} \ \\
& \quad , \ \text{while} \ \text{i} \ \text{(seq'} [\text{add} \ \text{ans} \ \text{j} \\
& \quad \quad , \ \text{assign} \ \text{i} \ \text{(Pred} \ \text{i})]) \\
& \quad \quad , \ \text{varAssign} \ \text{x} \ \text{ans}]}
\]
{count_0 := succ(n);
count_0 := pred(count_0);
ans_1 := 0;
ans_1 := succ(ans_1);
while count_0 =/= 0
    {x_2 := succ(ans_1);
x_2 := pred(x_2);
y_3 := succ(count_0);
y_3 := pred(y_3);
ans_4 := 0;
while x_2 =/= 0
    {i_5 := succ(y_3);
i_5 := pred(i_5);
while i_5 =/= 0
        {ans_4 := succ(ans_4);
i_5 := pred(i_5)};
    x_2 := pred(x_2)};
ans_1 := succ(ans_4);
ans_1 := pred(ans_1);
count_0 := pred(count_0)};
n := succ(ans_1);
n := pred(n)
Notes

• Large things can be constructed from many small things.

• The introduction of new “local” variables is crucial, but is possible since the model does not limit how many variables there are.

• The new variables must be “fresh”

• While the results are large (and perhaps slow) we are only interested in what we can express.