How to translate a while-loop into \( \mu \)-recursion

**Example:**

```java
int n = 1;
while (n < a)
    n = n * 2;
```

**Difference:**

- **For-loop:** number of iterations known beforehand
- **While-loop:** number of iterations is unknown

If we knew how many iterations we need, we could describe the computation as a for-loop.

\# of iterations needed will be denoted "blo".

```java
int n = 1;
for (int i = 1; i <= a; i++)
    n = n * 2;
```

To PPL terms:

\( n(0) = 1 \) \( \text{before the for-loop} \)

\( n(m+1) = n(m) \times 2 \) \( \text{inside the loop} \)

\( e_0(0) = 1 \)

\( h = \Pi_2 \times 2 \)

\( n(m) = \text{PE}(0, 0, \text{mul}(\Pi_2, 2)) \)

Denote the result of while-loop as \( w(a) \)

\( \delta_0: w(3) = 4 \quad w(12) = 16 \)

\( w(3) = 2 \)
\( w(a) \) is equal to the result of applying a for-loop when \# of iterations is equal to \( t(a) \)

\[ w(a) = n(t(a)) \]
\[ w(a) = n(t), \text{ where } t = t(a) \]

While-loop is \# of iterations is the smallest \# at which the condition is no longer satisfied

condition is \( n(t) < a \)

\( t(a) \) is the smallest natural \# \( t \) for which \( n(t) < a \)

\[ t(a) = \mu t (\neg (n(t) < a)) \]

\[ w(a) = n(t(a)) \]

where \( t(a) = \mu t (\neg (n(t) < a)) \)

or, if we plug in \( t(a) \) into the formula

\[ w(a) = n(\mu t (\neg (n(t) < a))) \]

\[ \mu \text{Review of the exam.} \]

PR = primitive recursive (obtained by \( \emptyset, \Sigma, \Pi, \delta, \mu \) using \( \emptyset \) and PR)

PR = Primitive Recursion (a function that can be defined using

\[ g(n_1, \ldots, n_k), h(n_1, \ldots, n_k, m, f_1, \ldots, f_k) \]

Theorem. Union of r.e. sets is r.e.

\[ A \cup B \]

Proof: We run A-alg for 1 hr or B-alg for 2 hr?
Proof: We run A-\text{alg} for 1 hr? Print all
“ “ B-\text{alg} for 1 hr elements of
“ “ A-\text{alg} for 1 hr A \cap B
“ “ B-\text{alg} for 1 hr

Example
A odd #’s
B even #’s
8 #’s per hour

A = 1, 3, 5, 7, 9, 11
B = 2, 4, 6, 8, 10, 12

A \cup B = 1, 3, 5, 2, 4, 6, 7, 9, 11, 8, 10, 12

A \cap B = 1, 3, 5

Question: If n was produced by B-\text{alg} at hrs 5,
when will it be printed by (A \cup B)-\text{alg}?

A = 10

Intersection

The intersection A \cap B of r.e. sets is r.e.

run A-\text{alg} for 1 hr
find common
run B-\text{alg} for 1 hr elements & print then
(run again)
compare longer lists &
print common elements.

If n \in A \cap B
it will be printed by A-\text{alg}

it will be printed by B-\text{alg}

n \in A \cap B

at moment 26 \text{ B it will be produced in the B-list}
At a moment $z_{t_a-1}$ it will be printed in the $\mathbf{A}$-list.

By moment max ($z_{t_b}, z_{t_a-1}$) it will appear in both list $\mathbf{A}$ will therefore be printed.

Complement:

If $\mathbf{A}$ and $\mathbf{\neg A}$ are r.e. then $\mathbf{A}$ is decidable.

\[ \mathbf{A} \rightarrow \neg \mathbf{A} \]
run $\mathbf{A}$-alg for $1hr$

\[ \mathbf{\neg A} \rightarrow \neg \mathbf{\neg A} \]
run $\mathbf{\neg A}$-alg for $1hr$

and wait until $n$ is printed

if $n$ appears in $\mathbf{A}$-list: $n \in \mathbf{A}$

if $n$ appears in $\mathbf{\neg A}$-list: $n \notin \mathbf{A}$

Th. No algorithm is possible that, given a program $p$ and data $d$, checks whether $p$ halts on $d$.

\[ \mathbf{\operatorname{halt-checker}} \]
whenever $p, d$ work

\[ \frac{\text{yes if } p \text{ halts on } d}{\text{no if } p \text{ doesn't halt on } d} \]

Proof based on Church-Turing thesis.

Proof. Let's define a code of a Java program.

A Java program is a sequence of ASCII symbols.

\[ \text{ASCII for } \]
\[ \text{for } \]
\[ 01101...00...1 \]
- Append 1 in front
- Interpret as natural number (integer)
The integer is called a code of a Java program.

**Lemma:** There exists an algorithm that, given a natural number $c$:
- Checks whether $c$ is a code of a Java program.
- If yes, returns the exe file for that function.
The file is denoted $f_c$.

$$f(n) = \begin{cases} f(n)+1 & \text{if } n \text{ is a code of a Java program and half-checker } (f_{n,n}) = \text{"true"} \\ \emptyset & \text{otherwise} \end{cases}$$

We assume half-checker exists. Then $f$ is computable.

$f$ is computable.

So $f$ is computable by a Java program
let $c_f$ denote the code of the Java program that computes this function $f$.

$\forall n \ (f_{c_f}(n) = f(n))$ In particular, for
\[ n = \chi \text{ we get } f(c_\emptyset)(c_\emptyset) = f(c_\emptyset) \]

Check \( f(c_\emptyset) \) definition:

\[ f(c_\emptyset)(c_\emptyset) = f(c_\emptyset(c_\emptyset)) + 1 \]

\[ \emptyset = 1 \]

\[ \text{Contradiction!} \]

Homework part 1: Write prove that no half-checker is possible.

Part 2: If \( n \) appears at moment 13 in \( A_{\text{alg}} \) when will the \((A\cup B)\)-alg print it?

Part 3: If \( n \) appears at moment 12 in \( A_{\text{alg}} \) and at moment 11 at \( B_{\text{alg}} \), when will \((A\cup B)\)-alg print it?

Part 4: Check web site.