Many Functions and Constructions
Are Primitive Recursive

**Previous and subtraction.** We have shown that addition and multiplication are primitive recursive. The natural next-to-addition operation is subtraction $a \div b$, and, in particular, the operation of $a \div 1$ of subtracting 1, that produces a previous natural number (or 0 if the input was 0). For convenience, let us denote this subtracting-1 operation by $prev(a)$.

**Theorem 1.** The function $prev(a)$ is primitive recursive.

**Proof.** The function $prev(a)$ can be described by the following two equations:

\[
prev(0) = 0; \\
prev(m + 1) = m.
\]

In general, a function of one variable described by primitive recursion has the form

\[
h(0) = f; \\
h(m + 1) = g(m, h(m)).
\]

After renaming, the equations describing the function $prev(a)$ takes the following form:

\[
h(0) = 0; \\
h(m + 1) = m.
\]

So here, $f = 0$, $g = \pi_1^2$, hence $prev = PR(0, \pi_1^2)$. Thus, the function $prev(a)$ is indeed primitive recursive. The theorem is proven.

**Theorem 2.** Subtraction $sub(a,b) = a \div b$ is primitive recursive.

**Proof.** Subtracting $b$ means that we subtract one $b$ times:

$$a \div b = a \div 1 \div 1 \div \ldots \div 1 \ (b \text{ times}).$$

Thus, we can describe subtraction as a for-loop:

```c
int c = a;
for(int i = 1, i <= b; i++)
    {c = prev(c);}
```
Here,
\[ c(a, 0) = a; \]
\[ c(a, m + 1) = \text{prev}(c(a, m)). \]
In general, a function of two variables described by primitive recursion has the form
\[ h(n_1, 0) = f(n_1); \]
\[ h(n_1, m + 1) = g(n_1, m, h(n_1, m)). \]
After renaming, the equations describing the function \( \text{prev}(a) \) takes the following form:
\[ h(n_1, 0) = n_1; \]
\[ h(n_1, m + 1) = \text{prev}(h(n_1, m)). \]
So here, \( f = \pi_1^1 \) and \( g = \text{prev}(\pi_3^3) \), hence \( \text{sub} = PR(\pi_1^1, \text{prev} \circ \pi_3^3) \). Since \( \text{prev} \)
is primitive recursive, the function \( \text{sub}(a, b) \) is indeed primitive recursive. The theorem is proven.

**What next.** To show that division and remainder are primitive recursive, we first need to show that conditional statements are primitive recursive. A general conditional statement has the form

\[ \text{if(<condition>)}<\text{statement}> \text{ else } <\text{statement}> \]

Here, a statement can be equality or inequality between numerical values, or a boolean combination of such statements – obtained by using “and”, “or”, and “not”. Following the usual computer representation of truth values, we will assume that “false” is 0 and “true” is 1.

To prove that everything computed this way is primitive recursive, we will therefore follow the following steps:

- first, we will prove that boolean operations are primitive recursive;
- then, we will prove that equalities and inequalities are primitive recursive;
- finally, we will prove that every conditional statement is primitive recursive.

**Theorem 3.** Negation \( \text{not}(a) \) is primitive recursive.

**Proof.** Since “false” is 0 and “true” is 1, negation is described by the following two formulas: \( \text{neg}(0) = 1 \) and \( \text{neg}(1) = 0 \). One can easily check that in both cases, we have \( \text{neg}(a) = 1 - a \). Since 1 is primitive recursive and \(-\) is primitive recursive, negation is primitive recursive too.

**Theorem 4.** The “and”-operation \( \text{and}(a, b) \) is primitive recursive.

**Proof.** There are 4 possible combinations of truth values of \( a \) and \( b \), and the corresponding truth values are as follows:
\[ \text{and}(0, 0) = \text{and}(0, 1) = \text{and}(1, 0) = 0; \text{ and}(1, 1) = 1. \]
One can easily check that in all 4 cases, \( \text{and}(a, b) = a \cdot b \). Since multiplication is primitive recursive, the function \( \text{and}(a, b) \) is therefore also primitive recursive.

*Comment.* It is not accidental that in digital design, “and” is described as \( AB \) – the same way as multiplication: “and”-operation is multiplication. The only reason why in Java they are different is that Java is a typed language, so multiplication can be only applies to numbers, but not to truth values.

**Theorem 5.** The “or”-operation \( \text{or}(a, b) \) is primitive recursive.

**Proof.** By De Morgan laws, \( \text{or}(a, b) = \neg(\text{and}(\neg(a), \neg(b))) \). Since \( \text{and} \) and \( \neg \) are primitive recursive, we can conclude that the function \( \text{or} \) is also primitive recursive – as a composition of primitive recursive functions.

**Theorem 6.** The function \( \text{eq}0(a) \) – that checks whether \( a \) is equal to 0 – is primitive recursive.

**Proof.** Indeed, we have

\[
\begin{align*}
\text{eq}0(0) &= 1; \\
\text{eq}0(m + 1) &= 0.
\end{align*}
\]

In general, a function of one variable described by primitive recursion has the form

\[
\begin{align*}
h(0) &= f; \\
h(m + 1) &= g(m, h(m)).
\end{align*}
\]

After renaming, the equations describing the function \( \text{eq}0(a) \) takes the following form:

\[
\begin{align*}
h(0) &= 1; \\
h(m + 1) &= 0.
\end{align*}
\]

So here, \( f = 1 = \sigma \circ 0, \ g = 0, \) hence \( \text{eq}0 = PR(\sigma \circ 0, 0) \). Thus, the function \( \text{eq}0(a) \) is indeed primitive recursive. The theorem is proven.

**Theorem 7.** The function \( \text{leq}(a, b) \) – that checks whether \( a \leq b \) – is primitive recursive.

**Proof.** From the definition of \( a \div b \), it follows that \( a \div b \) is equal to 0 if and only if \( a \leq b \) is true. Thus, \( \text{leq}(a, b) = \text{eq}0(a \div b) \). Since \( \text{eq}0 \) and \( \div \) are primitive recursive, we can conclude that the function \( \text{leq} \) is also primitive recursive – as a composition of primitive recursive functions.

**Theorem 8.** The function \( \text{eq}(a, b) \) – that checks whether \( a = b \) – is primitive recursive.

**Proof.** One can easily check that \( a = b \) if and only if \( a \leq b \) and \( b \leq a \). Thus, \( \text{eq}(a, b) = \text{and}(\text{leq}(a, b), \text{leq}(b, a)) \). Since \( \text{and} \) and \( \text{leq} \) are primitive recursive, we can conclude that the function \( \text{eq} \) is also primitive recursive – as a composition of primitive recursive functions.
Theorem 9. The function $lt(a,b)$ – that checks whether $a < b$ – is primitive recursive.

Proof. One can easily check that $a < b$ if and only if $a \leq b$ and $b \not\leq a$. Thus, $lt(a,b) = \text{and}(\text{leq}(a,b), \text{not}(\text{leq}(b,a)))$. Since $\text{and}$, $\text{leq}$, and $\text{not}$ are primitive recursive, we can conclude that the function $lt$ is also primitive recursive – as a composition of primitive recursive functions.

Theorem 10. The function $\text{neq}(a,b)$ – that checks whether $a \not= b$ – is primitive recursive.

Proof. One can easily check that $a \not= b$ if and only if it is not true that $a = b$. Thus, $\text{neq}(a,b) = \text{not}(\text{eq}(a,b))$. Since $\text{not}$ and $\text{eq}$ are primitive recursive, we can conclude that the function $\text{neq}$ is also primitive recursive – as a composition of primitive recursive functions.

Definition. Let $P(n)$ be a function whose values are 0 or 1, and let $f(n)$ and $g(n)$ are functions. By an if-then-else function $h(n) = \text{if}(P(n)) f(n) \text{ else } g(n)$, we mean the following function:

- $h(n) = f(n)$ if $P(n) = 1$, and
- $h(n) = g(n)$ if $P(n) = 0$.

Theorem 11. If $P(n)$, $f(n)$, and $g(n)$ are primitive recursive, then the function

$$h(n) = \text{if}(P(n)) f(n) \text{ else } g(n)$$

is also primitive recursive.

Proof. Let us take $h(n) = P(n) \cdot f(n) + (1 - P(n)) \cdot g(n)$. Then:

- if $P(n) = 1$, then $1 - P(n) = 0$, so $h(n) = 1 \cdot f(n) + 0 \cdot g(n) = f(n)$;
- if $P(n) = 0$, then $1 - P(n) = 1$, so $h(n) = 0 \cdot f(n) + 1 \cdot g(n) = g(n)$.

So, we indeed have the desired function. Since $P(n)$, $f(n)$, $g(n)$, $\cdot$, $+$, and $\div$ are primitive recursive, the function $h(n)$ is indeed primitive recursive as a composition of primitive recursive functions.

Comment. The above formula makes sense:

- we get $f(n)$ when $P(n)$ is true, i.e., when $P(n) = 1$, and
- we get $g(n)$ when $P(n)$ is false, i.e., when $\text{not}P(n) = 1 \div P(n)$ is true, i.e., when $1 \div P(n) = 1$. 

4
Theorem 12. The remainder function $\text{rem}(a, b) = b \% a$ is primitive recursive.

Proof. Let us recall how remainder with respect to a fixed number $a$ – e.g., with respect to $a = 3$ – change when $a$ changes. We have:

\begin{align*}
0 \% 3 &= 0; \\
1 \% 3 &= 1; \\
2 \% 3 &= 2; \\
3 \% 3 &= 0; \\
4 \% 3 &= 1; \\
5 \% 3 &= 2; \\
6 \% 3 &= 0; \\
7 \% 3 &= 1, \text{etc.}
\end{align*}

In general, the remainder starts with 0, and then grows by 1 unless adding 1 will lead to $a$ – in this case the remainder is set back at 0.

This shows that the remainder can be described as follows:

\begin{align*}
\text{rem}(a, 0) &= 0; \\
\text{rem}(a, m + 1) &= \text{if} (\text{rem}(a, m) + 1 < a) \text{rem}(a, m) + 1 \text{ else } 0.
\end{align*}

Since $<$, $+$ (1), and if-then-else construction are all primitive recursive, the right-hand side of the last formula is also primitive recursive. Thus, the remainder function $\text{rem}$ is primitive recursive.

Theorem 13. The integer division function $\text{div}(a, b) = b / a$ is primitive recursive.

Proof. Let us recall how the result $b / a$ of dividing by a fixed number $a$ – e.g., by $a = 3$ – change when $a$ changes. We have:

\begin{align*}
0 / 3 &= 0; \\
1 / 3 &= 0; \\
2 / 3 &= 0; \\
3 / 3 &= 1; \\
4 / 3 &= 1; \\
5 / 3 &= 1; \\
6 / 3 &= 2; \\
7 / 3 &= 2, \text{etc.}
\end{align*}
In general, the value $b / a$ starts with 0, and then remains the same unless we face a number which is divisible by $a$ – i.e., for which $\text{rem}(a, b) = 0$.

This shows that the integer division function can be described as follows:

$$
\text{div}(a, 0) = 0;
$$

$$
\text{div}(a, m + 1) = \text{if}(0 < \text{rem}(a, m + 1)) \text{ div}(a, m) \text{ else div}(a, m) + 1.
$$

Since $<$, $+1$ ($\sigma$), and if-then-else construction are all primitive recursive, the right-hand side of the last formula if also primitive recursive. Thus, the division function $\text{div}$ is primitive recursive.