6 Euclid’s Algorithm

In this section, we present Euclid’s algorithm for the greatest common divisor of two integers. An extended version of this algorithm will furnish the one implication that is missing in Figure 6.

Reduction. An important insight is Euclid’s Division Theorem stated in Section 4. We use it to prove a relationship between the greatest common divisors of numbers \( j \) and \( k \) and of \( j \) and \( k \) mod \( j \).

**Lemma.** Let \( j, k, q, r > 0 \) with \( k = jq + r \). Then \( \text{gcd}(j, k) = \text{gcd}(r, j) \).

**Proof.** We begin by showing that every common factor of \( j \) and \( k \) is also a factor of \( r \). Letting \( d = \text{gcd}(j, k) \) and writing \( j = Jd \) and \( k = Kd \), we get

\[
    r = k - jq = (K - Jq)d.
\]

We see that \( r \) can be written as a multiple of \( d \), so \( d \) is indeed a factor of \( r \). Next, we show that every common factor of \( r \) and \( j \) is also a factor of \( k \). Letting \( d = \text{gcd}(r, j) \) and writing \( r = Rd \) and \( j = Jd \), we get

\[
    k = jq + r = (Jq + R)d.
\]

Hence, \( d \) is indeed a factor of \( k \). But this implies that \( d \) is a common factor of \( j \) and \( k \) iff it is a common factor of \( r \) and \( j \).

Euclid’s gcd algorithm. We use the Lemma to compute the greatest common divisor of integers \( j \) and \( k \). The algorithm is recursive and reduces the integers until the remainder vanishes. It is convenient to assume that both integers, \( j \) and \( k \), are positive and that \( j \leq k \).

```plaintext
integer GCD(j, k)
    q = k div j; r = k - jq;
    if r = 0 then return j
    else return GCD(r, j)
endif.
```

If we call the algorithm for \( j > k \) then the first recursive call is for \( k \) and \( j \), that is, it reverses the order of the two integers and keeps them ordered as assumed from then on. Note also that \( r < j \). In words, the first parameter, \( j \), shrinks in each iteration. There are only a finite number of non-negative integers smaller than \( j \) which implies that after a finite number of iterations the algorithm halts with \( r = 0 \). In other words, the algorithm terminates after a finite number of steps, which is something one should always check, in particular for recursive algorithms.

**Extended gcd algorithm.** We now modify the algorithm so it also returns integers \( x \) and \( y \) for which \( \text{gcd}(j, k) = jx + ky \). If \( r = 0 \), then Euclid’s gcd algorithm returns \( j \) as the gcd. In the extended version, we also return \( x = 1 \) and \( y = 0 \). Now suppose \( r > 0 \). In this case, we recurse and get

\[
    \text{gcd}(r, j) = rx' + jy'
    = (k - jq)x' + jy'
    = j(y' - qx') + kx' = j(y' - qx') + kx'.
\]

We thus return \( g = \text{gcd}(r, j) \) as well as \( x = y' - qx' \) and \( y = x' \). As before, we assume \( 0 < j \leq k \) when we call the algorithm.

```plaintext
integer \^3 GCD(j, k)
    q = k div j; r = k - jq;
    if r = 0 then return (j, 1, 0)
    else (g, x', y') = \^3 GCD(r, j);
        return (g, y' - qx', x')
endif.
```

To illustrate the algorithm, we run it for \( j = 14 \) and \( k = 24 \). The values of \( j, k, q, r, g = \text{gcd}(j, k), x, y \) at the various levels of recursion are given in Table 2.

<table>
<thead>
<tr>
<th>( j )</th>
<th>( k )</th>
<th>( q )</th>
<th>( r )</th>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>24</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>-5</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Running the extended gcd algorithm on \( j = 14 \) and \( k = 24 \).

**Computing inverses.** The extended gcd algorithm provides the final implication needed to finish what we set out to prove in the preceding section.

**Lemma E.** If \( \text{gcd}(a, n) = 1 \) then the linear equation \( ax + ny = b \) has a solution for every \( b \in \mathbb{Z} \).

**Proof.** Assuming \( \text{gcd}(a, n) = 1 \), the extended gcd algorithm returns integers \( X \) and \( Y \) such that \( aX + nY = 1. \)
Multiplying with \(b\) gives integers \(x = bX\) and \(y = bY\) such that \(ax + ny = b\), as required.

We can thus update the relationship between the statements I, II, III, IV listed at the beginning of Section 5; see Figure 7. Having established that the integer \(a\) has a multiplicative inverse in \(\mathbb{Z}_n\) iff \(\gcd(a, n) = 1\), we note that this is the case whenever \(n\) is prime and \(0 < a < n\).

**Corollary.** If \(n\) is prime then every non-zero \(a \in \mathbb{Z}_n\) has a multiplicative inverse.

Furthermore, we can use the extended gcd algorithm to compute the multiplicative inverse of \(a\). Specifically, we get \(ax + ny = 1\) implying that \(x \mod n\) is the multiplicative inverse of \(a\) in \(\mathbb{Z}_n\).

\[
\text{integer INVERSE}(a, n) \\
(g, x, y) = x \text{GCD}(a, n); \\
\text{assert } g = 1; \text{ return } x \mod n.
\]

The assert statement makes sure that \(a\) and \(n\) are indeed relative prime, for else the multiplicative inverse would not exist. We have seen that \(x\) can be negative so it is necessary to take \(x \mod n\) before we report it as the multiplicative inverse.

**Multiple moduli.** Sometimes, we deal with large integers, larger then the ones that fit into a single computer word (usually 32 or 64 bits). In this situation, we have to find a representation that spreads the integer over several words. For example, we may represent an integer \(x\) by its remainders modulo 3 and modulo 5, as shown in Table 3. We see that the first 15 non-negative integers correspond to different pairs of remainders. We generalize this insight to relative prime numbers \(m\) and \(n\). This generalization is one of the oldest non-trivial mathematical results known to humanity.

<table>
<thead>
<tr>
<th>(x)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \mod 3)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>(x \mod 5)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>...</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3: Mapping the integers from 0 to 15 to pairs of remainders after dividing with 3 and with 5.

**Chinese Remainder Theorem.** Let \(m, n > 0\) be relative prime. Then for every \(a \in \mathbb{Z}_m\) and \(b \in \mathbb{Z}_n\), the system of two linear equations

\[
x \mod m = a; \\
x \mod n = b
\]

has a unique solution in \(\mathbb{Z}_{mn}\).

**Proof.** The proof works as suggested by the example, namely by showing that \(f : \mathbb{Z}_{mn} \rightarrow \mathbb{Z}_m \times \mathbb{Z}_n\) defined by

\[
f(x) = (x \mod m, x \mod n)
\]

is injective. Since both \(\mathbb{Z}_{mn}\) and \(\mathbb{Z}_m \times \mathbb{Z}_n\) have size \(mn\), this implies that \(f\) is a bijection. Hence, \((a, b) \in \mathbb{Z}_m \times \mathbb{Z}_n\) has a unique preimage, the solution of the two equations.

To prove that \(f\) is injective, we assume it is not. Hence there are \(x < y\), both in \(\mathbb{Z}_{mn}\), with \(x \mod m = y \mod m\) and \(x \mod n = y \mod n\). Hence, \((y - x) \mod m = (y - x) \mod n = 0\). In other words, \(y - x\) is a common multiple of \(m\) and \(n\). However, the least common multiple of \(m\) and \(n\) is

\[
\text{lcm}(m, n) = \frac{m \cdot n}{\gcd(m, n)} = mn.
\]

Since \(y - x < mn\), this gives a contradiction to the assumption that \(f\) is not injective. We thus conclude that \(f\) is injective, which completes the proof.

There is a further generalization to more then two moduli that are pairwise relative prime. To use the Chinese Remainder Theorem, we would take two large integers, \(x\) and \(y\), and represent them as pairs, \((x \mod m, x \mod n)\) and \((y \mod m, y \mod n)\). Arithmetic operations can then be done on the remainders. For example, \(x\) times \(y\) would be represented by the pair

\[
xy \mod m = [(x \mod m)(y \mod m)] \mod m; \\
x y \mod n = [(x \mod n)(y \mod n)] \mod n.
\]

We would choose \(m\) and \(n\) small enough so that multiplying two remainders can be done using conventional, single-word integer multiplication.
Summary. We discussed Euclid’s algorithm for computing the greatest common divisor of two integers, and its extended version which provides the missing implication in Figure 6. We have also learned the Chinese Remainder Theorem which can be used to decompose large integers into digestible junks.