3 Unlimited Register Machines and the Halting Problem

We now introduce unlimited Register Machines (URMs) as a model of computation for which we will argue later that it exactly captures the general notion of computability. URMs, introduced by Cutland (course text), can be viewed as a simplified model of a procedural programming language (e.g. Pascal). Because URMs operates on a rather high level of abstraction they are easier to understand than the equivalent, but more low level Turing Machines, which we will study in chapter 4. Fundamental results like the undecidability of the Halting Problem (proved in this chapter), or the existence of a universal program (chapter 8) are more easily obtained with URMs than with Turing Machines. In chapter 7 we will prove that URMs and Turing Machines are equivalent, that is, any URM can be effectively translated into a Turing Machine computing the same function and vice versa.

3.1 Syntax and semantics of URMs

1. A URM consists of
   - a **program counter** PC, which can store a positive natural number,
   - finitely many **registers** $R_1, \ldots, R_N$, each of which can store a natural number,
   - a **URM program** $P$, which is a finite list $I_1 \ldots I_{|P|}$ of URM instructions.

2. There are three kinds of **URM instructions**:
   - **Zero instruction**: zero($n$) $(1 \leq n \leq N)$.
   - **Successor instruction**: succ($n$) $(1 \leq n \leq N)$.
   - **Jump instruction**: jump($m, n, q$) $(1 \leq m, n \leq N, 1 \leq q)$.

3. URM instructions are executed as follows
   - zero($n$): set register $R_n$ to zero and increment the program counter PC by one.
   - succ($n$): increment $R_n$ by one and increment PC by one.
   - jump($m, n, j$): if $R_m$ and $R_n$ hold the same value then set PC to $j$, otherwise increment PC by one.

4. Given a URM program $P = I_1 \ldots I_{|P|}$ the partial function
   \[
P^{(k)}: \mathbb{N}^k \rightarrow \mathbb{N}
   \]
   is computed at arguments $a_1, \ldots, a_k$ as follows
   - **Input**: Set the program counter to one and store $a_1, \ldots, a_k$ in the registers $R_1, \ldots, R_k$.
     All other registers are set to zero.
   - **Iteration**: As long as PC holds a value $j \in \{1, \ldots, |P|\}$, execute instruction $I_j$. Otherwise halts.
Output: When $P$ has halted read off the result from register $R_1$, i.e. if $R_1$ stores value $b$ then $P^{(k)}(a_1, \ldots, a_k) := b$. If $P$ does not halt $P^{(k)}(a_1, \ldots, a_k)$ is undefined.

5. A partial function $f: \mathbb{N}^k \to \mathbb{N}$ is called URM computable if $f = P^{(k)}$ for some URM program $P$.

3.1.1 Example
Let $P := I_1 I_2 I_3 I_4$ where
\begin{align*}
I_1 &= \text{jump}(3, 2, 5) \\
I_2 &= \text{succ}(1) \\
I_3 &= \text{succ}(3) \\
I_4 &= \text{jump}(1, 1, 1)
\end{align*}

$P^{(2)}(a_1, a_2) = a_1 + a_2$, because: initially the registers $R_1, R_2, R_3$ hold the values $a_1, a_2, 0$ respectively. $R_1$ and $R_2$ are repeatedly incremented until $R_3$ and $R_2$ eventually hold the same value, in which case $P$ halts. Hence the incrementation takes place $a_2$ times and when $P$ halts $R_1$ holds value $a_1 + a_2$.

This example shows that addition is URM computable.

Exercise. Let $n$ and $m$ be two different numbers. Write a URM program that copies the content of register $n$ into register $m$ and leaves all other registers unchanged.

3.2 The undecidability of the Halting Problem
The undecidability of the Halting Problem was first proved by Alan M. Turing in his paper “On computable numbers, with an application to the Entscheidungsproblem”, Proceedings of the London Mathematical Society, 2, 42, pp 230-265, 1936. We recast his proof here using URMs instead of Turing machines.

In the following we will by “computable” always mean “URM-computable”. This will be justified later when we will argue that URM-computability does indeed coincide with the intuitive notion of computability (Church’s Thesis).

3.2.1 Definition (Decidable and undecidable problems)
1. By a problem we mean an $n$-ary predicate $M(x_1, \ldots, x_n)$ of natural numbers, i.e. a property of $n$-tuples of natural numbers.

An example is the binary predicate
\[
\text{Mult}(x, y) \iff x \text{ is a multiple of } y
\]

2. A problem $M$ is called decidable if the characteristic function of $M$ defined by
\[
\epsilon_M(x_1, \ldots, x_n) := \begin{cases} 
1 & \text{if } M(x_1, \ldots, x_n) \text{ holds} \\
0 & \text{if } M(x_1, \ldots, x_n) \text{ doesn’t hold}
\end{cases}
\]
is computable. This means that we are able to decide whether or not $M(x_1, \ldots, x_n)$ holds, by computing $c_M(x_1, \ldots, x_n)$. The result “1” means “Yes”, whereas “0” means “No”. If $M(x_1, \ldots, x_n)$ is not decidable then it is called an **undecidable problem**. Hence $M(x_1, \ldots, x_n)$ is undecidable if there is no algorithm that decides for every $n$-tuple $(x_1, \ldots, x_n)$ whether or not $M(x_1, \ldots, x_n)$ holds.

For example the predicate Mult$(x, y)$ considered above is decidable because its characteristic function

$$c_{\text{Mult}}(x, y) := \begin{cases} 1 & \text{if } x \text{ is a multiple of } y \\ 0 & \text{if } x \text{ isn't a multiple of } y \end{cases}$$

clearly is computable.

3. **The Halting Problem** is the following binary predicate:

$$\text{Halt}(x, y) \iff x \text{ codes a URM program } P \text{ that halts on input } y,$$

i.e. $x = \text{code}(P)$ and $P^{(1)}(y)$ is defined

In this definition we view a program as a string over the alphabet

$$\{s, j, 0, 1, (, )\},$$

where ‘s’, ‘j’, ‘i’ stand for zero, succ, jump, respectively and 0,1 are used to write numbers in binary notation.

**Example.** Let $P$ be the URM program consisting of the single instruction $\text{jump}(1, 2, 1)$. If initially $R_1$ is set to 7 and all other registers are set to 0, then clearly $P$ halts (after one computation step). Hence $P^{(1)}(7)$ is defined (the result is 7). Hence, if $e := \text{code}(P)$, then $\text{Halt}(e, 7)$ holds. On the other hand $\text{Halt}(e, 0)$ does not hold, since when all registers are set to 0, then clearly $P$ loops, and therefore $P^{(1)}(0)$ is undefined.

Although, as we have just seen, certain instances of the halting problem are easy to decide, we will now prove that no algorithm can exist that would decide all instances of the halting problem.

### 3.2.2 Theorem

The halting problem is undecidable.

**Proof.** Our assumption is that there is a program $P$ deciding the halting problem. This means that $P$ has the property that for all $x, y \in \mathbb{N}$

$$P^{(2)}(x, y) = \begin{cases} 1 & \text{if } x \text{ codes a URM program } Q \text{ such that } Q^{(1)}(y) \text{ is defined} \\ 0 & \text{otherwise} \end{cases}$$

We now show that this assumption must be false.

Using our assumed program $P$ we can easily construct another URM program $R$ which has the property that for all $x \in \mathbb{N}$

$$R^{(1)}(x) = \begin{cases} 0 & \text{if } P^{(2)}(x, x) = 0 \\ \text{undefined} & \text{if } P^{(2)}(x, x) = 1 \end{cases}$$

The construction of $R$ is as follows: Let $P = I_1 \ldots I_q$, (where $q := |P|$). Then $R$ is the following program of (length $q + 6$):
jump(1,2,4) \hspace{1cm} (1)
succ(2) \hspace{1cm} (2)
jump(1,1,1) \hspace{1cm} (3)
I'_1 \hspace{1cm} (4)
\cdot \hspace{1cm} \cdot 
\cdot \hspace{1cm} \cdot 
I'_{[P]} \hspace{1cm} (q + 3) 
zero(2) \hspace{1cm} (q + 4) 
succ(2) \hspace{1cm} (q + 5) 
jump(1,2,q + 6) \hspace{1cm} (q + 6) 

Here \( a^n \) indicates an appropriate adjustments of addresses in jump instructions, i.e. \( \text{jump}(m,n,j)^i := \text{jump}(m,n,j + 3) \) if \( j \leq q \), \( \text{jump}(m,n,j)^i := \text{jump}(m,n,[P] + 5) \) if \( j > q \), and \( I'_k = I_k \) if \( I_k \) is not a jump instruction.

From the construction of \( R \) it follows that for every URM program \( Q \) the following equivalence holds:

\[
(+) \quad R^{(1)}(\text{code}(Q)) \text{ is defined} \iff Q^{(1)}(\text{code}(Q)) \text{ is undefined}
\]

\[
(R^{(1)}(\text{code}(Q)) \text{ defined} \iff P^{(2)}(\text{code}(Q), \text{code}(Q)) = 0 \iff Q^{(1)}(\text{code}(Q)) \text{ undefined})
\]

Since \( (+) \) holds for all URM programs \( Q \), it particularly holds for \( Q := R \). This gives us

\[
R^{(1)}(\text{code}(R)) \text{ is defined} \iff R^{(1)}(\text{code}(R)) \text{ is undefined}
\]

which is absurd.

This contradiction shows that a URM program \( P \) deciding the Halting Problem cannot exist. \( \square \)

**Remarks.**

1. Note that the proof given above does not use any particular properties of the coding function \( \text{code} \).

2. The only property of our programming language URM we used was the fact that we can pass from the program \( P \) to the program \( R \) which is related to \( P \) as stated in the proof. It is clear that this passage is possible in any reasonable programming language. Hence, essentially the same proof works for any programming language.

3. The essential idea in the construction of the program \( R \) is that \( R \) takes an input \( x \) and runs \( P \) with arguments \( (x,x) \). This process is called diagonalization. Diagonalization is also the essential idea in Russell’s paradox and in the proof that \( A \not\subset P(A) \) (Theorem 2.1.2).

4. The undecidability of the Halting problem not only is an interesting fact on its own, but it also allows us to prove the undecidability of other important problems. These problems and their impact on e.g. Artificial Intelligence will be discussed later on in the course.