Computability and Logic

Slides by Prof. Yuxi Fu

BASICS, Department of Computer Science
Shanghai Jiao Tong University

October 26, 2023

Outline

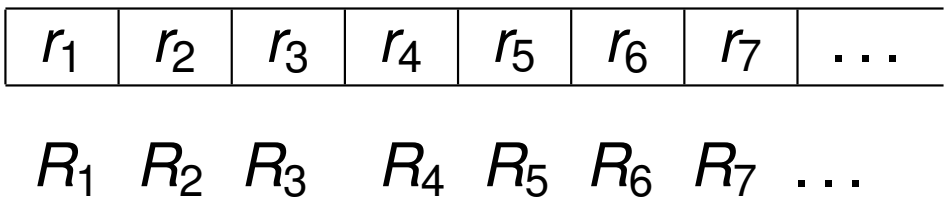
Unlimited Register Machine

Unlimited Register Machine

An Unlimited Register Machine (URM) is an idealized computer.

Register

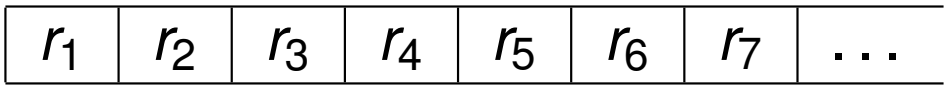
A URM has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.



Every register can hold a *natural number* at any moment.

Register

A URM has an infinite number of register labeled $R_1, R_2, R_3, \ldots$.



R_1 R_2 R_3 R_4 R_5 R_6 R_7 ...

Every register can hold a *natural number* at any moment.

The registers can be equivalently written as for example

$$[r_1 r_2 r_3]_1^3 [r_4]_4^4 [r_5 r_6 r_7 \ldots]_5^\infty$$

Program

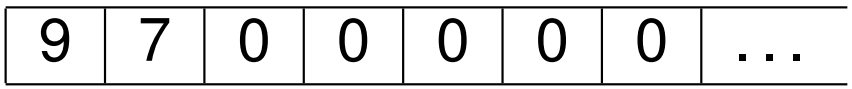
A URM also has a program, which is a finite list of instructions.

Instruction

Type	Instruction	Response of the URM
Zero	$Z(n)$	Replace r_n by 0.
Successor	$S(n)$	Add 1 to r_n.
Transfer	$T(m,n)$	Copy r_m to R_n.
Jump	$J(m,n,q)$	If $r_m = r_n$, go to the q-th instruction; otherwise go to the next instruction.

Computation

The registers:



R_1 R_2 R_3 R_4 R_5 R_6 R_7

The program:

$I_1 : J(1,2,6)$
$I_2 : S(2)$
$I_3 : S(3)$
$I_4 : J(1,2,6)$
$I_5 : J(1,1,2)$
$I_6 : T(3,1)$

Configuration and Computation

Configuration:
the contents of the registers + the current instruction number.

Configuration and Computation

Configuration:
the contents of the registers + the current instruction number.

Initial configuration, computation, final configuration.

Some Notation

Suppose P is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.

Some Notation

Suppose P is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots) \downarrow$ means that the computation converges.

Some Notation

Suppose P is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots)\downarrow$ means that the computation converges.
- $P(a_1, a_2, a_3, \ldots)\uparrow$ means that the computation diverges.

Some Notation

Suppose P is the program of a URM and $a_1, a_2, a_3, \ldots$ are the numbers stored in the registers.

- $P(a_1, a_2, a_3, \ldots)$ is the initial configuration.
- $P(a_1, a_2, a_3, \ldots)\downarrow$ means that the computation converges.
- $P(a_1, a_2, a_3, \ldots)\uparrow$ means that the computation diverges.
- $P(a_1, a_2, \ldots, a_m)$ is $P(a_1, a_2, \ldots, a_m, 0, 0, \ldots)$.

URM-Computable Function

What does it mean that a URM computes a (partial) n-ary function f?

URM-Computable Function

Let f be a partial n-ary function.

Suppose P is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$. The computation $P(a_1, \ldots, a_n)$ converges to b if

$$P(a_1, \ldots, a_n) \downarrow \textit{ and } r_1 = b \textit{ in the final configuration.}$$

We write $P(a_1, \ldots, a_n) \downarrow b$ in this case.

URM-Computable Function

Let f be a partial n-ary function.

Suppose P is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$. The computation $P(a_1, \ldots, a_n)$ converges to b if

$$P(a_1, \ldots, a_n) \downarrow \textit{ and } r_1 = b \textit{ in the final configuration.}$$

We write $P(a_1, \ldots, a_n) \downarrow b$ in this case.

P URM-computes f if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$, $P(a_1, \ldots, a_n) \downarrow b$ iff $f(a_1, \ldots, a_n) = b$.

URM-Computable Function

Let f be a partial n-ary function.

Suppose P is the program of a URM and $a_1, \ldots, a_n, b \in \mathbb{N}$. The computation $P(a_1, \ldots, a_n)$ converges to b if

$$P(a_1, \ldots, a_n) \downarrow \textit{ and } r_1 = b \textit{ in the final configuration.}$$

We write $P(a_1, \ldots, a_n) \downarrow b$ in this case.

P URM-computes f if, for all $a_1, \ldots, a_n, b \in \mathbb{N}$, $P(a_1, \ldots, a_n) \downarrow b$ iff $f(a_1, \ldots, a_n) = b$.

The function f is URM-computable if there is a program that URM-computes f.

We shall abbreviate “URM-computable” to “computable”.

Let

$$\mathcal{C}$$

be the set of computable functions and

$$\mathcal{C}_n$$

be the set of n-ary computable functions.

Examples

Construct a URM that computes $x + y$.

Examples

Construct a URM that computes $x + y$.

$I_1 : \ J(3, 2, 5)$
$I_2 : \ S(1)$
$I_3 : \ S(3)$
$I_4 : \ J(1, 1, 1)$

Examples

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

Examples

Construct a URM that computes $x \dot{-} 1 = \begin{cases} x - 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$

$I_1 : J(1, 4, 8)$
$I_2 : S(3)$
$I_3 : J(1, 3, 7)$
$I_4 : S(2)$
$I_5 : S(3)$
$I_6 : J(1, 1, 3)$
$I_7 : T(2, 1)$

Examples

Construct a URM that computes

$$x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$$

Examples

Construct a URM that computes

$$x \div 2 = \begin{cases} x/2, & \text{if } x \text{ is even,} \\ \text{undefined}, & \text{if } x \text{ is odd.} \end{cases}$$

$I_1 : J(1,2,6)$
$I_2 : S(3)$
$I_3 : S(2)$
$I_4 : S(2)$
$I_5 : J(1,1,1)$
$I_6 : T(3,1)$

Church-Turing Thesis

Two Questions

1. How do different models of computation compare to each other?

Two Questions

1. How do different models of computation compare to each other?

2. How do these models characterize the informal notion of effective computability?

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.

2. Turing (1936): Turing machines.

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936): λ-terms.

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936): λ-terms.
4. Post (1943): Post systems.

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936): λ-terms.
4. Post (1943): Post systems.
5. Markov (1951): Variants of the Post systems.

Other Approaches to Computability

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936): λ-terms.
4. Post (1943): Post systems.
5. Markov (1951): Variants of the Post systems.
6. Shepherdson-Sturgis (1963): URM-computable functions.

Fundamental Result

Each of the above proposals for a characterization of the notion of effective computability gives rise to the same class of functions.

Church's Thesis

The intuitively and informally defined class of effectively computable partial functions coincides exactly with the class $\mathcal{C}$ of URM-computable functions.

Church-Turing Thesis

1. All models define the same set of functions.

Church-Turing Thesis

1. All models define the same set of functions.

2. $\mathcal{C}$ is very complicated.

Church-Turing Thesis

1. All models define the same set of functions.

2. $\mathcal{C}$ is very complicated.

3. No one has contrived an intuitively computable function that does not belong to $\mathcal{C}$.

Church-Turing Thesis

1. All models define the same set of functions.

2. $\mathcal{C}$ is very complicated.

3. No one has contrived an intuitively computable function that does not belong to $\mathcal{C}$.
(When you are convincing people of the computability of your functions, you are constructing an interpretation from your model to a well-known model.)

Proof by Church-Turing Thesis

Church-Turing Thesis allows us to give an informal argument for the computability of a function.

Gödel Number

1931

General Remark

The set of the programs are countable.

General Remark

The set of the programs are countable.

More importantly, every program can be coded up effectively by a number in such a way that a unique program can be recovered from the number.

Denumerability and Enumerability

A set X is denumerable if there is a bijection $f : X \to \mathbb{N}$.

Denumerability and Enumerability

A set X is denumerable if there is a bijection $f : X \to \mathbb{N}$.

An enumeration of a set X is a surjection $g : \mathbb{N} \to X$;
this is often represented by writing $\{x_0, x_1, x_2, \ldots\}$.
It is an enumeration *without repetitions* if g is injective.

Denumerability and Enumerability

A set X is denumerable if there is a bijection $f : X \to \mathbb{N}$.

An enumeration of a set X is a surjection $g : \mathbb{N} \to X$;
this is often represented by writing $\{x_0, x_1, x_2, \ldots\}$.
It is an enumeration *without repetitions* if g is injective.

Let X be a set of "finite objects".
Then X is effectively denumerable if there is a bijection
$f : X \to \mathbb{N}$ such that both f and f^{-1} are effectively computable functions.

Effective Denumerability

Fact. $\mathbb{N} \times \mathbb{N}$ is effectively denumerable.

Effective Denumerability

Fact. $\mathbb{N} \times \mathbb{N}$ is effectively denumerable.

Proof. A bijection $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is defined by

$$\begin{aligned} \pi(m, n) &\stackrel{\text{def}}{=} 2^m(2n+1) - 1, \\ \pi^{-1}(l) &\stackrel{\text{def}}{=} (\pi_1(l), \pi_2(l)), \end{aligned}$$

where

$$\begin{aligned} \pi_1(x) &\stackrel{\text{def}}{=} (x+1)_1, \\ \pi_2(x) &\stackrel{\text{def}}{=} ((x+1)/2^{\pi_1(x)} - 1)/2. \end{aligned}$$

Effective Denumerability

Fact. $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$ is effectively denumerable.

Effective Denumerability

Fact. $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$ is effectively denumerable.

Proof. A bijection $\zeta : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$ is defined by

$$\begin{aligned} \zeta(m,n,q) &\stackrel{\text{def}}{=} \pi(\pi(m-1,n-1),q-1), \\ \zeta^{-1}(l) &\stackrel{\text{def}}{=} (\pi_1(\pi_1(l))+1, \pi_2(\pi_1(l))+1, \pi_2(l)+1). \end{aligned}$$

Effective Denumerability

Fact. $\bigcup_{k>0} \mathbb{N}^k$ is effectively denumerable.

Effective Denumerability

Fact. $\bigcup_{k>0} \mathbb{N}^k$ is effectively denumerable.

Proof. A bijection $\tau : \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\tau(a_1, \ldots, a_k) \stackrel{\text{def}}{=} 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots + 2^{a_1+a_2+a_3+\ldots,a_k+k-1} - 1.$$

Effective Denumerability

Fact. $\bigcup_{k>0} \mathbb{N}^k$ is effectively denumerable.

Proof. A bijection $\tau : \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\tau(a_1, \ldots, a_k) \stackrel{\text{def}}{=} 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \ldots + 2^{a_1+a_2+a_3+\ldots,a_k+k-1} - 1.$$

Now given x we can find a unique expression of the form

$$2^{b_1} + 2^{b_2} + 2^{b_3} + \ldots + 2^{b_k}$$

that equals to $x + 1$. It is then clear how to define $\tau^{-1}(x)$.

Gödel Encoding

Let $\mathcal{I}$ be the set of all instructions.

Let $\mathcal{P}$ be the set of all programs.

Gödel Encoding

Let $\mathcal{I}$ be the set of all instructions.

Let $\mathcal{P}$ be the set of all programs.

The objects in $\mathcal{I}$, and $\mathcal{P}$ as well, are 'finite objects'.
They must be effectively denumerable.

Gödel Encoding

Theorem. $\mathcal{I}$ is effectively denumerable.

Gödel Encoding

Theorem. $\mathcal{I}$ is effectively denumerable.

Proof. The bijection $\beta : \mathcal{I} \to \mathbb{N}$ is defined as follows:

$$\begin{aligned}
\beta(Z(n)) &= 4(n-1), \\
\beta(S(n)) &= 4(n-1)+1, \\
\beta(T(m,n)) &= 4\pi(m-1,n-1)+2, \\
\beta(J(m,n,q)) &= 4\zeta(m,n,q)+3.
\end{aligned}$$

Gödel Encoding

Theorem. $\mathcal{I}$ is effectively denumerable.

Proof. The bijection $\beta : \mathcal{I} \to \mathbb{N}$ is defined as follows:

$$\begin{aligned} \beta(Z(n)) &= 4(n-1), \\ \beta(S(n)) &= 4(n-1)+1, \\ \beta(T(m,n)) &= 4\pi(m-1,n-1)+2, \\ \beta(J(m,n,q)) &= 4\zeta(m,n,q)+3. \end{aligned}$$

The converse β^{-1} is easy.

Gödel Encoding

Theorem. $\mathcal{P}$ is effectively denumerable.

Gödel Encoding

Theorem. $\mathcal{P}$ is effectively denumerable.

Proof. The bijection $\gamma : \mathcal{P} \to \mathbb{N}$ is defined as follows:

$$\gamma(P) \quad = \quad \tau(\beta(I_1), \ldots, \beta(I_s)),$$

assuming $P = I_1, \ldots, I_s$.

Gödel Encoding

Theorem. $\mathcal{P}$ is effectively denumerable.

Proof. The bijection $\gamma : \mathcal{P} \to \mathbb{N}$ is defined as follows:

$$\gamma(P) \quad = \quad \tau(\beta(I_1), \ldots, \beta(I_s)),$$

assuming $P = I_1, \ldots, I_s$.

The converse γ^{-1} is obvious.

Gödel Encoding

The number $\gamma(P)$ is called the Gödel number of P.

Gödel Encoding

The number $\gamma(P)$ is called the Gödel number of P.

$$\begin{aligned} P_n &= \text{the program with Godel number } n \\ &= \gamma^{-1}(n) \end{aligned}$$

Gödel Encoding

Let P be the program $T(1,3), S(4), Z(6)$.

Gödel Encoding

Let P be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$, $\beta(S(4)) = 13$, $\beta(Z(6)) = 20$.

Gödel Encoding

Let P be the program $T(1,3), S(4), Z(6)$.

$\beta(T(1,3)) = 18$, $\beta(S(4)) = 13$, $\beta(Z(6)) = 20$.

$\gamma(P) = 2^{18} + 2^{32} + 2^{53} - 1$.

Gödel Encoding

Consider P_{4127}.

Gödel Encoding

Consider P_{4127}.

$4127 = 2^5 + 2^{12} - 1.$

Gödel Encoding

Consider P_{4127}.

$$4127 = 2^5 + 2^{12} - 1.$$

$$\beta(I_1) = 5 = 4 \times 1 + 1,$$
$$\beta(I_2) = 12 - 5 - 1 = 4 \times 1 + 2 = 4\pi(1,0) + 2.$$

Gödel Encoding

Consider P_{4127}.

$4127 = 2^5 + 2^{12} - 1.$

$\beta(I_1) = 5 = 4 \times 1 + 1,$
$\beta(I_2) = 12 - 5 - 1 = 4 \times 1 + 2 = 4\pi(1,0) + 2.$

So P_{4127} is $\quad S(2); T(2,1).$

Gödel Encoding

We shall fix this particular coding function γ throughout.