Fundamentele Informatica 3

voorjaar 2016

http://www.liacs.leidenuniv.nl/~vlietrvan1/fi3/

Rudy van Vliet
kamer 124 Snellius, tel. 071-527 5777
rvvliet(at)liacs(dot)nl

college 12, 25 april 2016

9. Undecidable Problems
9.5. Undecidable Problems
Involving Context-Free Languages
10. Computable Functions
10.1. Primitive Recursive Functions

A slide from lecture 11

Theorem 9.20.
These two problems are undecidable:

1. *CFGNonEmptyIntersection*:
   Given two CFGs G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?

2. *IsAmbiguous*:
   Given a CFG G, is G ambiguous?

Proof. . .

A slide from lecture 11

Definition 9.21. Valid Computations of a TM

Let $T = (Q, \Sigma, \Gamma, q_0, \delta)$ be a Turing machine.

A *valid computation* of T is a string of the form

$$z_0 \# z_1^r \# z_2 \# z_3^r \ldots \# z_n \#$$

if n is even, or

$$z_0 \# z_1^r \# z_2 \# z_3^r \ldots \# z_n^r \#$$

if n is odd,
where in either case, # is a symbol not in Γ,
and the strings z_i represent successive configurations of T on some input string x, starting with the initial configuration z_0 and ending with an accepting configuration.

The set of valid computations of T will be denoted by C_T.

A slide from lecture 11

Theorem 9.22.

For a TM $T = (Q, \Sigma, \Gamma, q_0, \delta)$,
• the set C_T of valid computations of T is the intersection of two context-free languages,
• and its complement C'_T is a context-free language.

Proof. . .

Corollary.

The decision problem

CFGNonEmptyIntersection:
Given two CFGs G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?

is undecidable (cf. Theorem 9.20(1)).

Proof.

Let
AcceptsSomething: Given a TM T, is $L(T) \neq \emptyset$?

Prove that *AcceptsSomething* $\leq$ *CFGNonEmptyIntersection*

Theorem 9.23. The decision problem

> *CFGGeneratesAll*: Given a CFG G with terminal alphabet Σ, is $L(G) = \Sigma^*$?

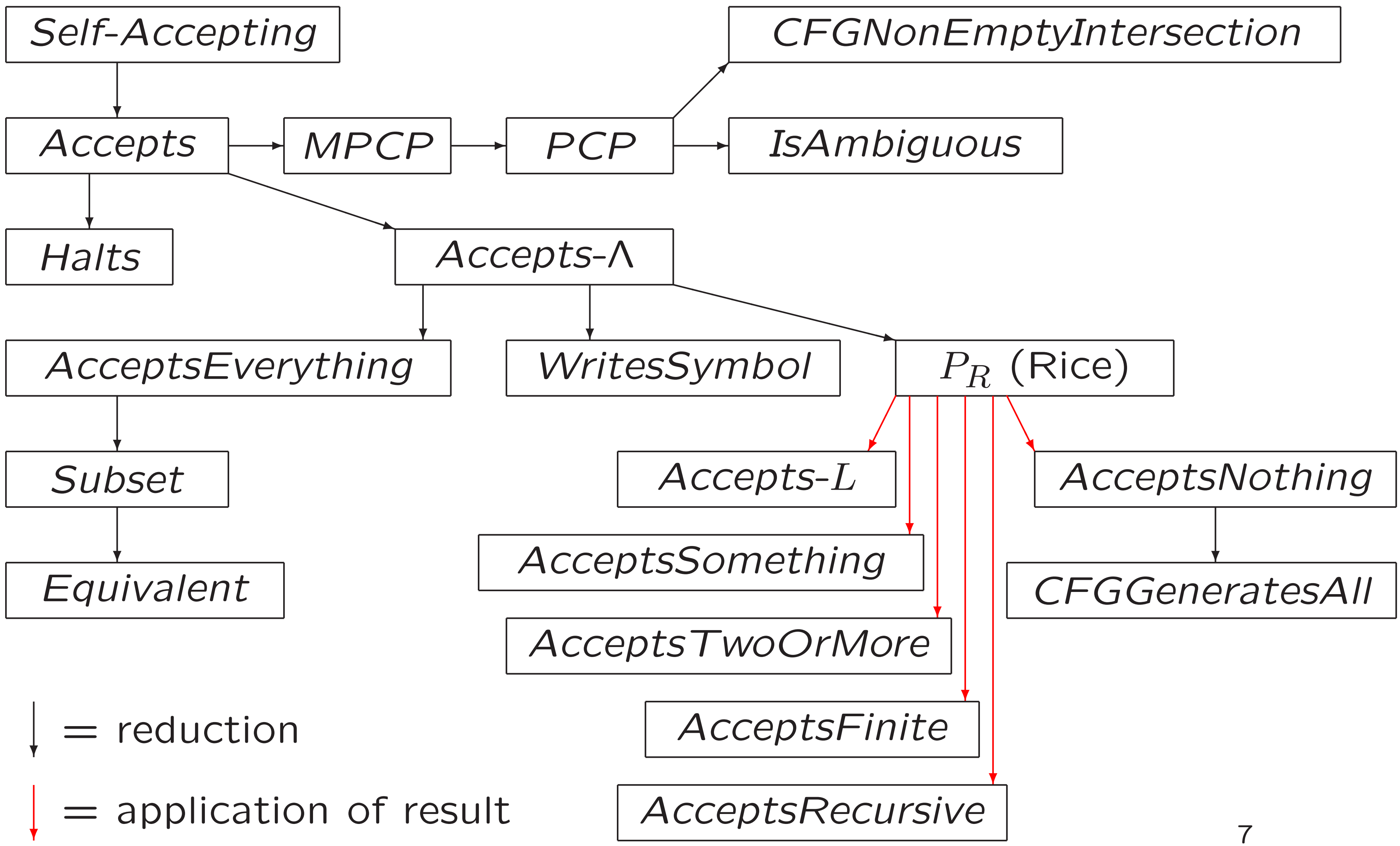
is undecidable.

Proof.

Let
AcceptsNothing: Given a TM T, is $L(T) = \emptyset$?

Prove that *AcceptsNothing* $\leq$ *CFGGeneratesAll* . . .

Undecidable Decision Problems (we have discussed)



10. Computable Functions

10.1. Primitive Recursive Functions

Exercise 10.1.

Let F be the set of partial functions from $\mathbb{N}$ to $\mathbb{N}$. Then $F = C \cup U$, where the functions in C are computable and the ones in U are not.

Show that C is countable and U is not.

Exercise 7.37.

Show that if there is a TM T computing the function $f : \mathbb{N} \to \mathbb{N}$, then there is another one, T', whose tape alphabet is $\{1\}$.

Exercise 7.37.

Show that if there is a TM T computing the function $f : \mathbb{N} \to \mathbb{N}$, then there is another one, T', whose tape alphabet is $\{1\}$.

Suggestion: Suppose T has tape alphabet $\Gamma = \{a_1, a_2, \ldots, a_n\}$. Encode Δ and each of the a_i's by a string of 1's and Δ's of length $n + 1$ (for example, encode Δ by $n + 1$ blanks, and a_i by $1^i\Delta^{n+1-i}$). Have T' simulate T, but using blocks of $n + 1$ tape squares instead of single squares.

Exercise.

How many Turing machines are there having n nonhalting states $q_0, q_1, \ldots, q_{n-1}$ and tape alphabet $\{0, 1\}$?

Exercise 10.2.

The *busy-beaver function* $b : \mathbb{N} \to \mathbb{N}$ is defined as follows.
The value $b(0)$ is 0.
For $n > 0$, there are only a finite number of Turing machines having n nonhalting states $q_0, q_1, \ldots, q_{n-1}$ and tape alphabet $\{0, 1\}$. Let $T_0, T_1, \ldots, T_m$ be the TMs of this type that eventually halt on input 1^n, and for each i, let n_{T_i} be the number of 1's that T_i leaves on its tape when it halts after processing the input string 1^n. The number $b(n)$ is defined to be the maximum of the numbers $n_{T_0}, n_{T_1}, \ldots, n_{T_m}$.

Show that the total function $b : \mathbb{N} \to \mathbb{N}$ is not computable.

Exercise 10.2.

The *busy-beaver function* $b : \mathbb{N} \to \mathbb{N}$ is defined as follows.
The value $b(0)$ is 0.
For $n > 0$, there are only a finite number of Turing machines having n nonhalting states $q_0, q_1, \ldots, q_{n-1}$ and tape alphabet $\{0, 1\}$. Let $T_0, T_1, \ldots, T_m$ be the TMs of this type that eventually halt on input 1^n, and for each i, let n_{T_i} be the number of 1's that T_i leaves on its tape when it halts after processing the input string 1^n. The number $b(n)$ is defined to be the maximum of the numbers $n_{T_0}, n_{T_1}, \ldots, n_{T_m}$.

Show that the total function $b : \mathbb{N} \to \mathbb{N}$ is not computable.
Suggestion: Suppose for the sake of contradiction that T_b is a TM that computes b. Then we can assume without loss of generality that T_b has tape-alfabet $\{0, 1\}$.

Definition 10.1. Initial Functions

The initial functions are the following:

1. *Constant* functions: For each $k \geq 0$ and each $a \geq 0$, the constant function $C_a^k : \mathbb{N}^k \rightarrow \mathbb{N}$ is defined by the formula

$$C_a^k(X) = a \quad \text{for every } X \in \mathbb{N}^k$$

Definition 10.1. Initial Functions

The initial functions are the following:

1. *Constant* functions: For each $k \geq 0$ and each $a \geq 0$, the constant function $C_a^k : \mathbb{N}^k \to \mathbb{N}$ is defined by the formula

$$C_a^k(X) = a \quad \text{for every } X \in \mathbb{N}^k$$

2. The *successor* function $s : \mathbb{N} \to \mathbb{N}$ is defined by the formula

$$s(x) = x + 1$$

Definition 10.1. Initial Functions

The initial functions are the following:

1. *Constant* functions: For each $k \geq 0$ and each $a \geq 0$, the constant function $C_a^k : \mathbb{N}^k \to \mathbb{N}$ is defined by the formula

$$C_a^k(X) = a \quad \text{for every } X \in \mathbb{N}^k$$

2. The *successor* function $s : \mathbb{N} \to \mathbb{N}$ is defined by the formula

$$s(x) = x + 1$$

3. *Projection* functions: For each $k \geq 1$ and each i with $1 \leq i \leq k$, the projection function $p_i^k : \mathbb{N}^k \to \mathbb{N}$ is defined by the formula

$$p_i^k(x_1, x_2, \ldots, x_k) = x_i$$

Definition 10.2. The Operations of Composition and Primitive Recursion

1. Suppose f is a partial function from $\mathbb{N}^k$ to $\mathbb{N}$, and for each i with $1 \leq i \leq k$, g_i is a partial function from $\mathbb{N}^m$ to $\mathbb{N}$.
   The partial function obtained from f and $g_1, g_2, \ldots, g_k$ by composition is the partial function h from $\mathbb{N}^m$ to $\mathbb{N}$ defined by the formula

$$h(X) = f(g_1(X), g_2(X), \ldots, g_k(X)) \text{ for every } X \in \mathbb{N}^m$$

Definition 10.2. The Operations of Composition and Primitive Recursion (continued)

2. Suppose $n \geq 0$ and g and h are functions of n and $n+2$ variables, respectively. (By "a function of 0 variables," we mean simply a constant.)
   The function obtained from g and h by the operation of *primitive recursion* is the function $f : \mathbb{N}^{n+1} \to \mathbb{N}$ defined by the formulas

$$\begin{aligned} f(X,0) &= g(X) \\ f(X,k+1) &= h(X,k,f(X,k)) \end{aligned}$$

   for every $X \in \mathbb{N}^n$ and every $k \geq 0$.

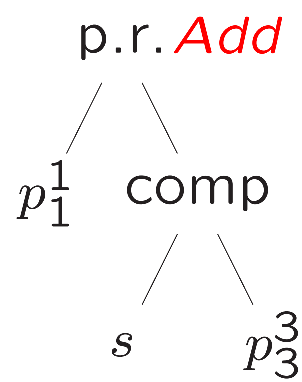
Example 10.5. Addition, Multiplication and Subtraction

$$\mathit{Add}(x, y) = x + y$$

Example 10.5. Addition, Multiplication and Subtraction

$$Add(x, y) = x + y$$

Structure tree:



Definition 10.3. Primitive Recursive Functions

The set *PR* of *primitive recursive* functions is defined as follows.

1. All initial functions are elements of *PR*.
2. For every $k \geq 0$ and $m \geq 0$, if $f : \mathbb{N}^k \to \mathbb{N}$ and $g_1, g_2, \ldots, g_k : \mathbb{N}^m \to \mathbb{N}$ are elements of *PR*, then the function $f(g_1, g_2, \ldots, g_k)$ obtained from f and $g_1, g_2, \ldots, g_k$ by composition is an element of *PR*.
3. For every $n \geq 0$, every function $g : \mathbb{N}^n \to \mathbb{N}$ in *PR*, and every function $h : \mathbb{N}^{n+2} \to \mathbb{N}$ in *PR*, the function $f : \mathbb{N}^{n+1} \to \mathbb{N}$ obtained from g and h by primitive recursion is in *PR*.

In other words, the set *PR* is the smallest set of functions that contains all the initial functions and is closed under the operations of composition and primitive recursion.

Example 10.5. Addition, Multiplication and Subtraction

$$\mathit{Mult}(x, y) = x * y$$

Example 10.5. Addition, Multiplication and Subtraction

$$\mathit{Sub}(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

$x \dot{-} y$

Example 10.5. Addition, Multiplication and Subtraction

$$\mathit{Sub}(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

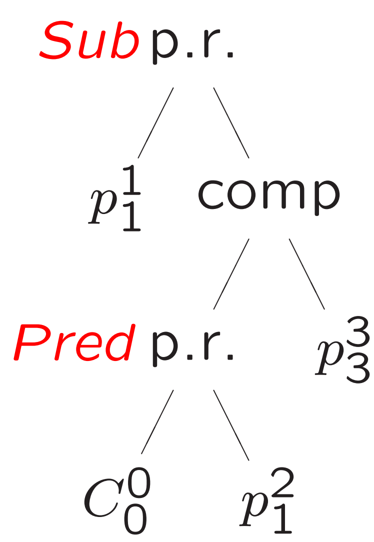
$x \dot{-} y$

$$\begin{aligned} \mathit{Sub}(x, 0) &= x && (\text{so } g = p_1^1) \\ \mathit{Sub}(x, k+1) &= \mathit{Pred}(\mathit{Sub}(x, k)) \\ & && (= h(x, k, \mathit{Sub}(x, k)), \text{ so } h = \mathit{Pred}(p_3^3)) \end{aligned}$$

Example 10.5. Addition, Multiplication and Subtraction

$$Sub(x,y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} Sub(x,0) &= x && (\text{so } g = p_1^1) \\ Sub(x,k+1) &= Pred(Sub(x,k)) \\ & && (= h(x,k,Sub(x,k)), \text{ so } h = Pred(p_3^3)) \end{aligned}$$



Theorem 10.4.

Every primitive recursive function is total and computable.

Proof. . .

Theorem 10.4.

Every primitive recursive function is total and computable.

```
i = 0;
v = g(x)
while (i<k)
{ v = h(x,i,v)
  i ++;
}
```

Theorem 10.4.

Every primitive recursive function is total and computable.

PR:
total and computable

Turing-computable functions:
not necessarily total

Example 10.5. Addition, Multiplication and Subtraction

$$Sub(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

$x \dot{-} y$

n-*place predicate* P is function from $\mathbb{N}^n$ to $\{\text{true}, \text{false}\}$

characteristic function χ_P defined by

$$\chi_P(X) = \begin{cases} 1 & \text{if } P(X) \text{ is true} \\ 0 & \text{if } P(X) \text{ is false} \end{cases}$$

We say P is primitive recursive...

Theorem 10.6.

The two-place predicates *LT*, *EQ*, *GT*, *LE*, *GE*, and *NE* are primitive recursive.
(*LT* stands for "less than," and the other five have similarly intuitive abbreviations.)
If P and Q are any primitive recursive n-place predicates, then $P \wedge Q$, $P \vee Q$ and $\neg P$ are primitive recursive.

Proof. . .