Fundamentele Informatica 3
Antwoorden op geselecteerde opgaven uit Hoofdstuk 11

John Martin: Introduction to Languages and the Theory of Computation

Jetty Kleijn

Najaar 2008

11.1 Show that the relation $\leq$ (reducibility, for languages or decision problems) is reflexive and transitive. Give an example to show that it is not symmetric.
Recall: for two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, we write $L_1 \leq L_2$ if there is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$: $x \in L_1$ if and only if $x \in L_2$.
Reflexivity. $L \leq L$ always holds: take for f the identity mapping.
Transitivity. Assume $L_1 \leq L_2$ and $L_2 \leq L_3$, then we have to prove that $L_1 \leq L_3$. Let $f_1 : \Sigma_1^* \to \Sigma_2^*$ be the reduction from L_1 to L_2 and let $f_2 : \Sigma_2^* \to \Sigma_3^*$ be the reduction from L_2 to L_3.
Then, for all $x \in \Sigma_1^*$: $x \in L_1$ iff $f_1(x) \in L_2$ iff $f_2(f_1(x)) \in L_3$.
The function $f_2 \circ f_1$ obviously is computable and thus $L_1 \leq L_3$.
Not symmetric. A simple example are $L_1 = \{a\}^*$ and $L_2 = \{\Lambda\} \subseteq \{a\}^*$. The function $f : \{a\}^* \to \{a\}^*$ defined by $f(a^k) = \Lambda$ for all $k \geq 0$ is computable and has the property that for all $w \in \{a\}^*$ it holds that $w \in L_1$ if and only if $f(w) = \Lambda \in L_2$. Thus $L_1 \leq L_2$, but $L_2 \leq L_1$ does not hold: there exists no (total) function $g : \{a\}^* \to \{a\}^*$ such that if $w \neq \Lambda$, then $w \notin L_1 = \{a\}^*$.

Extras

As another example that $\leq$ is not symmetric we use the idea that whenever $L \leq L'$, then if L is not recursive (or recursively enumerable), then also L' is not recursive (recursively enumerable, respectively), see Theorem 11.4 and Exercise 11.3. Thus $L \leq L'$ implies that L cannot be harder to solve than L'.

Now let $L_1 = \{a\}^+$ and let $L_2 = SA = \{w \in \{0,1\}^* \mid w = e(T)$ for some TM T and $w \in L(T)\}$. It appears that L_1 is an "easier" language than L_2. Indeed, let T_0 be the (trivial) Turing machine which accepts $\{0,1\}^+$. Note that $e(T_0) \in L_2$. Then $L_1 \leq L_2$ using the reduction $f(\Lambda) = \Lambda$ and $f(a^k) = e(T_0)$ for all $k \geq 1$. Clearly, f is Turing-computable and we have, for all a^k, that $a^k \in L_1$ if and only if $f(a^k) = e(T_0) \in L_2$.
Conversely, there cannot exist a reduction $g : \{0,1\}^* \to \{a\}^+$ from L_2 to L_1 because L_1 is a recursive language and $L_2 = SA$ is not recursive: the existence of such a g would imply that we could decide membership of SA by reducing it to L_1.

Despite the suggestive notation, $\leq$ is *not* a partial ordering, because it is not asymmetric:
Let $L_1 = \{a^{2n} \mid n \geq 0\}$ and $L_2 = \{a^{2n+1} \mid n \geq 0\}$. Then $L_1 \leq L_2$ via the reduction f_1 defined by $f_1(a^k) = a^{k+1}$ for all $k \geq 0$. Clearly $a^k \in L_1$ if and only if $f_1(a^k) = a^{k+1} \in L_2$. Also, $L_2 \leq L_1$, now via the reduction f_2 defined by $f_2(a^k) = a^{k+1}$ for all $k \geq 0$. Clearly $a^k \in L_2$ if and only if $f_2(a^k) = a^{k+1} \in L_1$.
Consequently, $L_1 \leq L_2$ and $L_2 \leq L_1$, but $L_1 = L_2$ does not hold.
We could say that two languages (problems) are "equivalent" if they can be reduced to one another: $L_1 \sim L_2$ holds if $L_1 \leq L_2$ and $L_2 \leq L_1$. It is now easy to see that $\sim$ is indeed an equivalence relation indicating that one language is "as difficult" as the other.
See also the Exercises 11.10 and 11.11.

11.2 Given is the decision problem P_2:

$$\text{Given } n \in \mathbb{N}\text{; is } n = 2k \text{ for some } k \in \mathbb{N}?$$

Consider $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = 5n$ for all n.
a Determine a property X such that for all $n \in \mathbb{N}$:

$$n = 2k \text{ if and only if } f(n) = 5n \text{ satisfies } X.$$

X is "divisibility by 10" and so $P_2 \leq P_X$ via f with P_X:

$$\text{Given } n \in \mathbb{N}\text{; is } n = 10m \text{ for some } m \in \mathbb{N}?$$

b Give a (total) computable function $g : \mathbb{N} \to \mathbb{N}$ which reduces P_X to P_2.
Define g by $g(n) = k$ if $n = 5k$ for some k and $g(n) = 1$ otherwise.
Clearly, g is computable.
Moreover, for all $n \in \mathbb{N}$ we have:
if $n = 10m$ (a yes-instance of P_X), then $g(n) = 2m$ (a yes-instance of P_2),

and if n is not divisible by 10 (a no-instance of P_X), then
either it is not divisible by 5 and $g(n) = 1$ (a no-instance of P_2)
or it is not divisible by 2, in which case also $g(n)$ is not divisible by 2 (a no-instance of P_2).
Consequently n is a yes-instance of P_X if and only if $g(n)$ is a yes-instance of P_2. Thus also $P_X \leq P_2$.

11.3 Let $L_1, L_2 \subseteq \Sigma^*$ be two languages such that $L_1 \leq L_2$ and L_2 is recursively enumerable. Prove that L_1 is also recursively enumerable.
Let $f : \Sigma^* \to \Sigma^*$ be a function that reduces L_1 to L_2 and let T_f be a Turing machine that computes f. Let T_2 be a Turing machine such that $L(T_2) = L_2$. Consider the composite TM $T_f T_2$. When given a word $x \in \Sigma^*$ as input, it transforms first x into $f(x)$ which is input to T_2 and thus accepted if and only if $f(x) \in L_2$. Since $f(x) \in L_2$ if and only if $x \in L_1$ it follows that $T_f T_2$ accepts x if and only if $x \in L_1$.
In other words $L_1 = L(T_f T_2)$ and so L_1 is recursively enumerable.

11.4 Let $L \subseteq \Sigma^*$ be a language such that $L \neq \emptyset$ and $L \neq \Sigma^*$.
Show that any recursive language can be reduced to L.
Let $u, v \in \Sigma^*$ be such that $u \in L$ and $v \notin L$.
Now let L' be a recursive language over Σ. Define $f : \Sigma^* \to \Sigma^*$ by

$$f(w) = u \text{ if } w \in L' \text{ and } f(w) = v \text{ if } w \notin L'.$$

It is immediate that, for all $w \in \Sigma^*$, we have that $w \in L'$ iff $f(w) \in L$.
Moreover, f is computable, because L' is recursive.
Thus $L' \leq L$.

11.5 (see also Exercise 10.8c)
Given an (effective) enumeration of 4-tuples (n, x, y, z) consisting of positive integers with $n \geq 3$, one can build a Turing machine that tests these 4-tuples for the equality $x^n + y^n = z^n$ and stops successfully as soon as the equality is satisfied. Therefore, deciding whether this TM stops given an empty tape as input, is the same as disproving Fermat's last theorem.

11.6 $Acc = \{e(T)e(w) \mid T \text{ is a TM and } T \text{ accepts } w\}$.
Let L be any recursively enumerable language over some alphabet Σ.
Then $L \leq Acc$ which can be seen as follows.
Let T_0 be a Turingmachine accepting L. Define $f : \Sigma^* \to \{0,1\}^*$ by

$$f(x) = e(T_0)e(x) \text{ for all } x \in \Sigma^*.$$

Clearly f is computable (a simple application of e).
Furthermore, for all $x \in \Sigma^*$, we have
$x \in L$ if and only if T_0 accepts x if and only if $f(x) = e(T_0)e(x) \in Acc$.

11.7 Let L be a language and T a Turingmachine such that $L(T) = L$. Assume that the problem

Given a string w; does T accept w?

is solvable. Then L is a recursive language: to decide whether a word $x \in L$ we simply use the algorithm (Turing machine) for the given problem and decide whether T accepts x, that is whether $x \in L(T) = L$.
Consequently, if Turing machine M is such that $L(M)$ is not recursive, it must be the case that the problem

Given a string w; does M accept w?

is unsolvable.

11.8 Show that for any word $x \in \Sigma^*$, the problem Accepts:

Given TM T and string w; is $w \in L(T)$?

can be reduced to the problem Accepts-x:

Given TM T; is $x \in L(T)$?

To prove this we have to transform each instance (T, w) of Accepts to an instance T' of Accepts-x such that $w \in L(T)$ iff $x \in L(T')$.
The function F yields, given a pair (T, w), the Turingmachine $F(T, w) = T'$ which operates as follows:
given input y, T' begins with comparing y with x;
if $y = x$, then
  it erases the tape,
  writes w on the tape from cell 1 onwards, and
  then simulates T on w;

thus T' accepts x if and only if $w \in L(T)$

if $y \neq x$, then the behaviour of T' is not relevant, let us say it moves to h_a.
Clearly, this is an algorithmic procedure to obtain $T' = F(T, w)$.
So, Accepts reduces to Accepts-x.
Since Accepts is unsolvable, it follows that Accepts-x is unsolvable.

11.9 Construct a reduction from the problem Accepts-Λ:

$$\text{Given TM } T\text{; is } \Lambda \in L(T)?$$

to the problem Accepts-$\{\Lambda\}$:

$$\text{Given TM } T\text{; is } L(T) = \{\Lambda\}?$$

We have to provide an algorithm which when given a TM T transforms it into a TM T' such that $\Lambda \in L(T)$ if and only if $L(T') = \{\Lambda\}$.
Let T' be the Turing machine which behaves as T when given input Λ and immediately rejects every other input.
Thus $L(T') = \emptyset$ if $\Lambda \notin L(T)$ and $L(T') = \{\Lambda\}$ if $\Lambda \in L(T)$.
We conclude that $\Lambda \in L(T)$ if and only if $L(T') = \{\Lambda\}$.

11.10
a Let $C = A \cup B$ and $D = A \cap B$. Then $A = B$ if and only if $C \subseteq D$.
b Show that the problem Equivalent:

$$\text{Given two TMs } T_1 \text{ and } T_2\text{; is } L(T_1) = L(T_2)?$$

can be reduced to the problem Subset:

$$\text{Given two TMs } T_1 \text{ and } T_2\text{; is } L(T_1) \subseteq L(T_2)?$$

Now we can use the proof of Theorem 10.3. There, constructions are provided which, given two arbitrary Turing machines T_1 and T_2 yield a TM $T_\cup$ and a TM $T_\cap$ such that $L(T_\cup) = L(T_1) \cup L(T_2)$ and $L(T_\cap) = L(T_1) \cap L(T_2)$. By **a**, these constructions together provide a reduction from Equivalent to Subset: transform any instance (T_1, T_2) of Equivalent into the instance $(T_\cup, T_\cap)$ of Subset of the corresponding union and intersection TMs. Then $L(T_1) = L(T_2)$ if and only if $L(T_\cup) \subseteq L(T_\cap)$.

11.11
a Let $C = A \cap B$ and $D = A$. Then $A \subseteq B$ if and only if $C = D$.
b Show that the problem Subset:

$$\text{Given two TMs } T_1 \text{ and } T_2\text{; is } L(T_1) \subseteq L(T_2)?$$

can be reduced to the problem Equivalent:

$$\text{Given two TMs } T_1 \text{ and } T_2\text{; is } L(T_1) = L(T_2)?$$

We use the proof of Theorem 10.3. There, a construction is described which, given two arbitrary Turing machines T_1 and T_2 yields a TM $T_\cap$ such that

$L(T_\cap) = L(T_1) \cap L(T_2)$. Thus given an instance (T_1, T_2) of Subset, we construct $T_\cap$ from T_1 and T_2 and we let $(T_\cap, T_1)$ be the corresponding instance of Equivalent. Then, by **a**, $L(T_1) \subseteq L(T_2)$ if and only if $L(T_\cap) = L(T_1)$.

11.12

a decidable

Let T be an arbitrary TM. We have to decide whether it ever reaches another of its states than its initial state when started with a blank tape.

Execute T with empty input,

then we know the answer and stop the procedure as soon as

T changes state at some moment; stop with answer YES

otherwise we encounter one of the following situations:

T halts (h_a and h_r are not considered to be states of T); stop with answer NO

T moves its head to the right; since it still is in q_0, it is in an infinite loop; stop with answer NO

T scans cell 0 and sees in that cell a tape symbol it has seen there before; since it still is in q_0, it is in an infinite loop; stop with answer NO.

b and **c** are both undecidable.

SKETCH of proof: for both, this can be shown by transforming any given TM into an equivalent one (defining the same language) with a new dummy state (q) just before the transitions to h_a. Then Accepts-Λ can be shown to reduce to the problem of **b** and AcceptsSomething to the problem of **c**.

d and **e** are both undecidable.

SKETCH of proof: every Turingmachine can be transformed into one which defines the same language and in which all finite computations consist of an even number of steps. Then Accepts-Λ can be shown to reduce to the problem of **d**, and AcceptsSomething can be shown to reduce to the problem of **e**.

f and **g** are undecidable (HINT: every TM can be effectively transformed into an equivalent one which for each input either stops successfully or enters an infinite computation (it never crashes or enters h_r), see exercise 9.11 and the proof of Theorem 11.6; then use the negations of Accepts and of AcceptsSomething, respectively).

h and **i** are undecidable (HINT: T rejects input w means that the computation of T on w will eventually halt unsuccessfully: it "crashes" or enters h_r; give a transformation which given a TM interchanges crashing and successfully halting; then use Accepts and AcceptsSomething respectively).

j and **k** are decidable (HINT: within 10 steps a TM cannot have seen more than the first 10 symbols of its input).

l The problem P_l

$$\text{given two TMs } T_1,\, T_2;\text{ is } L(T_1) \subseteq L(T_2) \text{ or } L(T_2) \subseteq L(T_1)?$$

is undecidable. This is a decision problem: for every instance (T_1, T_2) the answer is either "yes" if at least one of the inclusions hold or "no" if neither of them is true. Despite the fact thatP_l has instances which consist of a pair of TMs rather than a single one, its undecidability can be proved using Rice's theorem:
If R is a non-trivial property for recursively enumerable languages, then the problem P_R is undecidable:

$$\text{given a TM } T;\text{ does } L(T) \text{ have property } R?$$

For R we choose the property "$L \subseteq \{\Lambda\}$ or $\{\Lambda\} \subseteq L$". This is a non-trivial property for recursively enumerable languages and so the problem P_Λ:

$$\text{given a TM } T;\text{ is } L(T) \subseteq \{\Lambda\} \text{ or } \{\Lambda\} \subseteq L(T)?$$

is undecidable. This problem easily reduces to P_l: The transformation F when given an instance T of P_Λ transforms it into the pair (T, T_Λ), where T_Λ is a Turingmachine which accepts $\{\Lambda\}$: when started it moves its head from cell 0 to cell 1 and it checks the contents of cell 1; if that is Δ, it accepts; otherwise it rejects. Thus F is computable. Moreover, $L(T) \subseteq \{\Lambda\}$ or $\{\Lambda\} \subseteq L(T)$ if and only if $L(T) \subseteq L(T_\Lambda)$ or $L(T_\Lambda) \subseteq L(T)$. That is, T is a yes-instance of P_Λ if and only if $F(T) = (T, T_\Lambda)$ is a yes-instance of P_l.
Since P_Λ is undecidable, it follows that P_l is undecidable.

11.14 Four decision problems are given involving unrestricted grammars. The proof of Theorem 10.9 shows that for every Turingmachine T an unrestricted grammar G_T can be constructed generating $L(T)$. Consequently, to each of the given problems the corresponding problem for Turingmachines can be reduced. Since these problems (Accepts, AcceptsSomething, AcceptsEverything, and Equivalent) are unsolvable, the given problems are also unsolvable.

11.15 Given is the problem WritesNonblank:

Given a TM T;
does T ever write a nonblank symbol, when started with a blank tape?

This problem is decidable: let n be the number of states of the given T. If we let T run on a blank tape, then one of the following cases must occur within n moves: it halts (successfully or not) or it enters a state for the second time. In the first case we can observe whether or not a nonblank has been written. In the second case: if it has not yet written a nonblank symbol it will never do so anymore (T has entered an infinite loop).

11.16 A "proof" is given that the problem WritesNonblank of the previous exercise is not decidable. Find the flaw.
The construction is a reduction of WritesNonblank to the unsolvable WritesSymbol which does not prove anything (the reduction is in the wrong direction!).

11.17 Given is the instance of PCP obtained from the Turingmachine in Example 11.2 with input ab. Note that ab is accepted. Thus the instance of PCP has a solution. Give this solution.

First pair:	$(\#, \#q_0\Delta ab\#)$
Instruction (pair of type 2):	$(q_0\Delta, \Delta q_1)$
Matching (pairs of type 1):	(a, a)
and	(b, b)
and	$(\#, \#)$
and	(Δ, Δ)
Instruction (pair of type 2):	$(q_1 a, aq_1)$
Matching (pairs of type 1):	(b, b)
and	$(\#, \#)$
and	(Δ, Δ)
and	(a, a)
Instruction (pair of type 2):	$(q_1 b, bq_1)$
Matching (pairs of type 1):	$(\#, \#)$
and	(Δ, Δ)
and	(a, a)
Instruction (pair of type 2):	$(bq_1\#, q_2 b\Delta\#)$
Matching (pairs of type 1):	(Δ, Δ)
and	(a, a)
Instruction (pair of type 2):	$(q_2 b, h_a\Delta)$
Matching (pairs of type 1):	(Δ, Δ)
and	$(\#, \#)$
and	(Δ, Δ)
Termination (pair of type 3):	$(ah_a\Delta, h_a)$
Matching (pairs of type 1):	(Δ, Δ)
and	$(\#, \#)$
Termination continued (pair of type 3):	$(\Delta h_a\Delta, h_a)$

Matching (pair of type 1): $(\#, \#)$
Finally, the pair of type 4: $(h_a\#\#, \#)$
Both the sequence of α's and that of the β's equal:
$\#q_0\Delta ab\#\Delta q_1 ab\#\Delta a q_1 b\#\Delta ab q_1\#\Delta a q_2 b\Delta\#\Delta a h_a\Delta\Delta\#\Delta h_a\Delta\# h_a\#\#$.

11.18 Give a solution or show that none exists for each of the following two instances of PCP:
a $(\alpha_1, \beta_1) = (100, 10)$, $(\alpha_2, \beta_2) = (101, 01)$, $(\alpha_3, \beta_3) = (110, 1010)$.
Any solution has to begin with the first pair $(100, 10)$;
this should then be followed by a pair the second component of which starts with a 0; only pair 2 qualifies and we obtain $(100101, 1001)$;
once more we have to continue with pair 2 and we obtain $(100101101, 100101)$;
the second component is now a string 101 "behind", thus we have to continue with pair 1 or with pair 3.
In the first case we get $(100101101100, 10010110)$ and we are stuck, because the second component is now 1100 behind and none of the β's fit this pattern.
In the second case we get $(100101101110, 1001011010)$ which is a mistake because the 10th position is a 1 in the first word, but a 0 in the second word. Thus this instance has no solution.
b $(\alpha_1, \beta_1) = (1, 10)$, $(\alpha_2, \beta_2) = (01, 101)$, $(\alpha_3, \beta_3) = (0, 101)$, en $(\alpha_4, \beta_4) = (001, 0)$.
Each solution has to start with the first or the fourth pair.
One solution is the sequence 1,4,2: $\alpha_1\alpha_4\alpha_2 = 100101 = \beta_1\beta_4\beta_2$.
Can you find still other ones?

11.19 Restricting PCP to instances in which the alphabet consists of at most two symbols does not lead to a decidable problem, because the general problem can be reduced to this simplified version by a binary encoding:
Let $(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)$ be an instance of PCP with each $\alpha_i, \beta_i \in \Sigma^*$ where Σ is an alphabet consisting of $m \geq 1$ symbols.
We encode $\Sigma = \{a_1, \ldots, a_m\}$ as follows: $c(a_i) = 0^i 1$ for all $i \in \{1, \ldots, m\}$.
This encoding is extended to words by applying it to each letter in the word. Note that it is injective, in the sense that, for all words $u, v \in \Sigma^*$, we have $c(u) = c(v)$ if and only if $u = v$.
Consequently we have mapped the instance $(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)$ over Σ to the instance $(c(\alpha_1), c(\beta_1)), (c(\alpha_2), c(\beta_2)), \ldots, (c(\alpha_n), c(\beta_n))$ of PCP over the binary alphabet $\{0, 1\}$.
It is not difficult to see that the original instance has a solution if and only if its encoding has a solution. Thus if PCP with (at most) binary alphabets

would be decidable, then also the general Post Correspondence Problem, a contradiction.

11.20 In contrast to the previous exercise, restricting PCP to instances in which the alphabet consists of one symbol does lead to a decidable problem. Words over a unary alphabet can differ only with respect to their length. Words of the same length are equal. This is the basis of the algorithm below.
Let Σ be an alphabet consisting of one symbol.
Let $(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)$ be an instance of PCP with each $\alpha_i, \beta_i \in \Sigma^*$. Assume that $\Sigma = \{0\}$. Thus for each $l \in \{1, \ldots, n\}$ there are $l_1, l_2 \geq 1$ such that $\alpha_l = 0^{l_1}$ and $\beta_l = 0^{l_2}$.
Now first check whether there exists an l such that $l_1 = l_2$. If yes, then a solution has been found ($\alpha_l = \beta_l$).
If no, then for all l we have $l_1 \neq l_2$. Now check whether $l_1 > l_2$ for all l.
If yes, then the instance has no solution (any sequence of α's will be longer than the corresponding sequence of β's).
If no, then check whether $l_1 < l_2$ for all l. If yes, then the instance has no solution (any sequence of α's will be shorter than the corresponding sequence of β's).
The only remaining case is that there exist two different indices j and k in $\{1, \ldots, n\}$ such that $j_1 > j_2$ and $k_1 < k_2$ and in this case the instance always has a solution:
Let $p = j_1 - j_2$ and $q = k_2 - k_1$. Then r times the pair (α_j, β_j) leaves the β-sequence $r.p$ symbols behind, while s times the pair (α_k, β_k) adds $s.q$ symbols more to the β-sequence than to the α-sequence. Thus if we let $r = q$ and $s = p$, then the α-sequence and the β-sequence are of the same length. Thus a solution is $i_1 = j, \ldots, i_q = j, i_{q+1} = k, \ldots, i_{q+p} = k$, because $(0^{j_1})^q (0^{k_1})^p = (0^{j_2})^q (0^{k_2})^p$.

11.21 Show that each of the following problems for context-free grammars is undecidable. We do this in each case by a reduction from the (undecidable) problem CFG-GeneratesAll:

$$\text{Given a CFG } G \text{ with terminal alphabet } \Sigma\text{; is } L(G) = \Sigma^*?$$

a CFG-Equivalence:

$$\text{Given two CFGs } G_1 \text{ and } G_2\text{; is } L(G_1) = L(G_2)?$$

Let G be a CFG with terminal alphabet Σ. Define the CFG G_Σ by the productions $S \to \Lambda$ and $S \to aS$ for all $a \in \Sigma$. Thus $L(G_\Sigma) = \Sigma^*$. With each instance G of CFG-GeneratesAll, we thus associate the instance (G, G_Σ)

of CFG-Equivalence. This is clearly algorithmic, and since $L(G) = \Sigma^*$ if and only if $L(G) = L(G_\Sigma)$ we have reduced CFG-GeneratesAll to CFG-Equivalence.

b CFG-Subset:

Given two CFGs G_1 and G_2; is $L(G_1) \subseteq L(G_2)$?

Let G be a CFG with terminal alphabet Σ. Define the CFG G_Σ as above. Thus $L(G_\Sigma) = \Sigma^*$. With each instance G of CFG-GeneratesAll, we associate the instance (G_Σ, G) of CFG-Subset. This is clearly algorithmic, and since $L(G) = \Sigma^*$ if and only if $L(G_\Sigma) \subseteq L(G)$ we have reduced CFG-GeneratesAll to CFG-Subset.

c CFG-Regularity:

Given CFG G and regular language R; is $L(G) = R$?

With each instance G with terminal alphabet Σ of CFG-GeneratesAll we associate the instance (G, Σ^*) of CFG-Regularity. (Σ^* is a regular language.) This is clearly algorithmic, and since obviously $L(G) = \Sigma^*$ if and only if $L(G) = \Sigma^*$ we have reduced CFG-GeneratesAll to CFG-Regularity.

---

versie 06 januari 2009