Infinite Automata 2025/26
Lecture Notes 12

Henry Sinclair-Banks

This lecture is based on the paper "Improved Ackermannian Lower Bound for the Petri Nets Reachability Problem" by Sławomir Lasota, 2022.

Definition 12.1. *Fast growing functions.* The first *fast growing function* $f_1 : \mathbb{N} \to \mathbb{N}$ is defined by $f_1(n) \coloneqq 2n$, and the k-th *fast growing function* is defined inductively as

$$f_k(n) = \underbrace{f_{k-1} \circ \ldots \circ f_{k-1}}_{n \text{ times}}(1).$$

For example,

$$f_2(n) = \underbrace{f_1 \circ \ldots \circ f_1}_{n \text{ times}}(1) = \underbrace{2 \cdot \ldots \cdot 2}_{n \text{ times}} = 2^n,$$

and

$$f_3(n) = \underbrace{f_2 \circ \ldots \circ f_2}_{n \text{ times}}(1) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{n \text{ times}} = \mathsf{Tower}(n).$$

We define the *Ackermann function* as $A(n) \coloneqq F_n(n)$, for all $n \in \mathbb{N}$.

It will be convenient to use a slightly different family of fast growing functions defined over the multiples of 4; $F_i : \mathbb{N} \to \mathbb{N}$ are defined by the following. $F_1(n) = 2n$, and

$$F_k(n) = \underbrace{F_{k-1} \circ \ldots \circ F_{k-1}}_{\frac{n}{4} \text{ times}}(4).$$

These fast growing functions only differ from the "standard" fast growing functions (Definition 12.1) by a linear shift.

Claim 12.2. $F_k(4n) = 4f_k(n)$ for all $k, n \in \mathbb{N}$.

Proof hint. Prove this claim using induction on k. □

Naturally, we will can fast growing functions to define complexity classes.

Definition 12.3. *Higher order complexity classes.* The k-th *fast growing function complexity class* F_k contains all problems that can be decided in $f_k(n)$ time and is closed under reductions computable in lower-order time $f_{k-1}^m(n)$ for some fixed $m \in \mathbb{N}$. For example, $\mathsf{F}_3 = \mathsf{Tower}$ contains problems that are decidable in time $\mathsf{Tower}(2^n)$ because $\mathsf{Tower}(2^n) = f_3 \circ f_2(n)$. We also accordingly define the complexity class $\mathsf{Ackermann}$ which contains all problems that can be decided in time $A(n)$ is closed under primitive recursive reductions.

Our goal is now to prove that reachability in VASS is $\mathsf{Ackermann}$-hard. To achieve this, we will start with an problem that is known to be $\mathsf{Ackermann}$-hard and reduce VASS reachability to it.

Definition 12.4. Reachability in counter machines with F_k zero tests.

Input. A counter machine with two zero-tested counters M of size n (without loss of generality, assume that $n \in 4 \cdot \mathbb{N}$), an initial state p and a target state q.

Question. Does there exist a run from $(p, \mathbf{0})$ to $(q, \mathbf{0})$ which uses at most $\frac{F_k(n)-1}{2}$ zero tests?

Claim 12.5. Reachability in counter machines with F_k zero tests is F_k-hard for all $k \in \mathbb{N}$.

Proof hint. Reachability in counter machines with F_k zero tests reduces to the reachability problem in two-counter machines which asks whether there is a run of length at most $F_k(n)$ from $(p, \mathbf{0})$ to $(q, \mathbf{0})$. Suppose M has d counters $\mathsf{x}_1, \mathsf{x}_2, \ldots, \mathsf{x}_\mathsf{d}$. One can use the the Gödel encoding $\mathsf{y} = 2^{\mathsf{x}_1} 3^{\mathsf{x}_2} \cdot \ldots \cdot p_d^{\mathsf{x}_\mathsf{d}}$ with an ancillary (secondary) counter z. Notice that in order to update the counter x_i, one needs to multiply or divide y be some multiple of p_i and this requires a constant number of zero tests. □

It will be convenient for us to work with counter programs instead of VASS as state and transition diagrams. Here is an example program (without zero tests) with counters $\mathsf{C} = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$.

```
1. loop:
2.     x −= 1
3.     y += 1
4.     z += 2
5. z += 1
```

At times we will use counter programs with zero tests and other times we will use counter programs *without* zero tests; the latter exactly correspond to VASS.

Definition 12.6. A *counter valuation* is a vector $\mathbf{v} \in \mathbb{N}^\mathsf{C}$. Let P be some counter program with counters C and let $\mathsf{X} \subseteq \mathsf{C}$ be a set of counters. An X-*zeroing run* from $V \subseteq \mathbb{N}^\mathsf{C}$ in P is a run from the first line of P with some counter valuation $\mathbf{v} \in V$ to the last line of P ending with a counter valuation $\mathbf{w} \in \mathbb{N}^\mathsf{C}$ such that $\mathbf{w}[\mathsf{x}] = 0$, for every $\mathsf{x} \in \mathsf{X}$. For example, let $V = \{(10, 0, 0)\}$, there is only one $\{\mathsf{x}\}$-zeroing run from $(10, 0, 0)$ in the above example counter program; this run ends at the counter valuation $(0, 10, 21)$.

Let $\mathsf{X} \subseteq \mathsf{C}$ and $V \subseteq \mathbb{N}^\mathsf{C}$. The set of counter valuations reachable by X-zeroing from V in P is called the X-*computed* set (of counter valuations). For example, the set of counter valuations that is $\{\mathsf{x}\}$-computed from $\{(x, 0, 0) : x \in \mathbb{N}\}$ in the above example counter program is $\{(0, y, z) : z = 2y + 1\}$. We note that if the set that is $\emptyset$-computed from V in P consist of the counter valuations that can be reached by complete runs (executes the final line of P) that need not have any counters equal zero.

Our goal is to prove the following theorem, which implies that reachability in VASS is Ackermann-hard.

Theorem 12.7. Reachability (i.e. deciding whether the C-computed set of runs from the initial counter valuation is non-empty) in programs without zero tests with $3k + 2$ counters is F_k-hard.

We shall now explain the technique of using triples to simulate zero tests (see also Exercises 12.1 and 12.2).

Definition 12.8. Let $B \in \mathbb{N}$ and C be a set of counters, and let $\mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathsf{C}$ be some distinguished triplet of counters. $\mathit{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}) \coloneqq \{\mathbf{v} \in \mathbb{N}^\mathsf{C} : \mathbf{v}[\mathsf{b}] = B, \mathbf{v}[\mathsf{c}] = c \in \mathbb{N}, \text{ and } \mathbf{v}[\mathsf{d}] = Bc\}$.

Suppose that P is some counter program with zero tests. We can introduce three new counter b, c, and d which will be initialised so that $\mathsf{b} = B$, $\mathsf{c} = c$, and $\mathsf{d} = Bc$. These counters will be used by a certain gadget that will replace the zero tests in P in such a way that b will control the number of zero tests performed, c will aid in the simulation of zero tests, and d will only be able to reach 0 (at the end of the program) if the specified number of zero tests were all faithfully performed. There will be three modifications: first P is transformed into P', then into P'', and finally in P'''. For simplicity, we shall assume that P only zero-tests two counters: x and y.

The first modification is to introduce the three new counters b, c, and d, which are not used in P. The role of c will be to maintain the invariant

$$\mathsf{c} + \mathsf{x} + \mathsf{y} = s, \tag{1}$$

where $s \in \mathbb{N}$ is some constant. Specifically, s is chosen to be at least the greatest sum of x and y and whenever x or y are updated in P, we are sure to match this with opposing updates to c in P'.

Original update in P	Replaced by in P'
$\mathsf{x} \mathrel{+}= 1$	$\mathsf{x} \mathrel{+}= 1, \mathsf{c} \mathrel{-}= 1$
$\mathsf{x} \mathrel{-}= 1$	$\mathsf{x} \mathrel{-}= 1, \mathsf{c} \mathrel{+}= 1$
$\mathsf{y} \mathrel{+}= 1$	$\mathsf{y} \mathrel{+}= 1, \mathsf{c} \mathrel{-}= 1$
$\mathsf{y} \mathrel{-}= 1$	$\mathsf{y} \mathrel{-}= 1, \mathsf{c} \mathrel{+}= 1$

The second modification is to replace the zero tests of x and y with gadgets that use counters b, c, and d. The following gadget is called `zero-test(x)`.

```
1. loop:
2.     y −= 1, x += 1, d −= 1
3. loop:
4.     c −= 1, y += 1, d −= 1
5. loop:
6.     y −= 1, c += 1, d −= 1
7. loop:
8.     x −= 1, y += 1, d −= 1
9. b −= 2
```

We also define the gadget `zero-test(y)` which is the same as `zero-test(x)` with counters x and y swapped. The program P'' is obtained by replacing, in P', all occurrences of $\mathsf{x}=?\,0$ and $\mathsf{x}=?\,0$ with `zero-test(x)` and `zero-test(y)`, respectively. There are two crucial observations about the zero-test gadgets, the first is that any time this gadget is taken b decreases by 2, this means that if $\mathsf{b} = 2m$ originally, then at most m zero tests can be simulated in P''. The second is that when the `zero-test(x)` gadget is taken once, then d decreases by at most $2s$. In fact, observe that the only way for d to decrease by $2s$ is if the four loops are all executed maximally and if $\mathsf{x} = 0$ to begin with. Therefore, if initially $\mathsf{d} = \mathsf{b} \cdot \mathsf{c}$, then the only way to maintain this invariant after a zero-test gadget is used is if (i) the four loops are iterated maximally, and (ii) the counter was actually zero at the start of the gadget. If a zero-test gadget is incorrectly used (i.e. the counter in question was not zero to begin with or if the loops are not all iterated maximally), then $\mathsf{d} > \mathsf{b} \cdot \mathsf{c}$ and the equality $\mathsf{d} = \mathsf{b} \cdot \mathsf{c}$ is unrecoverable because we know that b is always decreased by 2 and the greatest possible decrease of d is $2s$; this means that $\mathsf{d} = 0$ cannot be achieved at the end of the run if *any* of the zero tests are not faithfully simulated.

The third modification is to add a final gadget at the end of the run to flush the value of c to 0. Let `zero-test(c)` be the zero-test gadget that is obtained by swapping x and c in `zero-test(x)`. The following gadget is called `flush(c)`.

```
1. loop:
2.     c −= 1, d −= 2
3. zero-test(c)
```

By appending `flush(c)` to the end of P'', one obtains the final program P'''. Please note the following two observations. The first is that `flush(c)` uses one zero test (gadget) to get the value of c down to zero; so if to begin with $\mathsf{b} = B$, then at most $\frac{B-1}{2}$ zero tests of x and y can be simulated (one zero test is saved for c at the end). The second is that the invariant $\mathsf{x} + \mathsf{y} + \mathsf{c} = s$ (Equation 1) is violated by `flush(c)`, but this is not an issue because this gadget is only taken at the every end of the run after all zero tests on x and y have been simulated.

Altogether, the modifications and their justifications can be used to prove the following statement of correctness. In the following lemma, we use $\mathsf{C}' = \mathsf{C} \cup \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$ to denote the set of counters used P''' (which is also the counters used by P' and P'').

Lemma 12.9. Let $V \subseteq \mathbb{N}^{\mathsf{C}}$ be the set of counter valuations that are $\emptyset$-computed by P from $\{\mathbf{0}\}$ using m zero tests. The following two set are equal.

(1) The extension of V to $\mathbb{N}^{\mathsf{C}'}$ with $\mathsf{b} = \mathsf{c} = \mathsf{d} = 0$, that is $\{(\mathbf{v}, 0, 0, 0) \in \mathbb{N}^{\mathsf{C}} \times \mathbb{N}^{\{\mathsf{b},\mathsf{c},\mathsf{d}\}} : \mathbf{v} \in V\}$.

(2) The set $\{\mathsf{d}\}$-computed by P''' from $\mathit{Ratio}(2(m+1), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C}')$.

Given that reachability in counter machines with F_k zero tests is F_k-hard for all $k \in \mathbb{N}$, if we are able to construct a program with counters C that computes, from $\mathbf{0}$, the set $\mathit{Ratio}(F_k(n), \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$,

then we deduce that reachability in counter machines is F_k-hard for all $k \in \mathbb{N}$. Such a gadget is called a *multiplier*.

Definition 12.10. A program M (without zero tests) with counter C that z-computes, from the zero valuations $\{\mathbf{0}\}$, the set $\mathit{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$, for some four of its counters $\mathsf{z}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in C$ is called a *B-multiplier*.

The following program is a B-multiplier.

```
1. b += B
2. loop:
3.     c += 1, d += B.
```

Unfortunately, the size of this B-multiplier is $\mathcal{O}(B)$. This means that it cannot be used to prove F_k-hardness of reachability. Instead, we need to construct $F_k(n)$-multipliers of size $\mathit{poly}(n)$; for this we will introduce *amplifiers* in the next lecture. We shall conclude this lecture by stating the correctness statement that can be used to show that multipliers, together with programs that use triples to simulate zero test can be composed.

Claim 12.11. Let M be a B-multiplier with counters C (including $\mathsf{z} \in \mathsf{C}$). Let P be a program with counters $\mathsf{C} \setminus \{\mathsf{z}\}$ and let $\mathsf{Y} \subseteq \mathsf{C} \setminus \mathsf{z}$. The set Y-computed by P from $\mathit{Ratio}(B, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{C})$ is equal to the set $(\{z\} \cup Y)$-computed by the composed program MP from $\{\mathbf{0}\}$.