A Pedagogical Proof of the Immerman-Szelepscényi Theorem

Richard E. Ladner and Julie Goldberg
University of Washington
Department of Computer Science and Engineering
Seattle, WA 98195-2350

November 21, 2000

The celebrated result of Immerman [1] and Szelepscényi [2] that nondeterministic space classes are closed under complement has a very subtle and beautiful proof. Making the proof accessible to a wide range of students is challenging. We have found that the concept of space bounded nondeterministic computable partial functions to be very useful. We assume our Turing machines are nondeterministic, have accept states and produce outputs. The output is only meaningful if the machine halts in an accept state. To be specific: a partial function $f(x)$ is *nondeterministically computable* in space $s(n)$ (or time $t(n)$) if there is a nondeterministic Turing Machine T with output such that for all x of length n:

- all computations run in allotted space $s(n)$ (or time $t(n)$),
- $f(x)$ is defined if and only if T accepts x,
- if $f(x)$ is defined then any output produced by T in the accept state is $f(x)$.

We see that a language L is in NSPACE($s(n)$) if and only if there is a partial function f, nondeterministically computable in space $s(n)$, with the property that for all inputs x, $x \in L$ implies $f(x) =$ true and $x \notin L$ implies $f(x)$ is undefined.

We now use this concept in the proof of the theorem.

Theorem (Immerman-Szelepscényi) If $s(n) \geq \log n$ and tape constructible, then

$$\text{NSPACE}(s(n)) = \text{CoNSPACE}(s(n)).$$

Proof. It suffices to show $L \in \text{NSPACE}(s(n))$ implies $\overline{L} \in \text{NSPACE}(s(n))$. Let T be a $s(n)$ space bounded nondeterministic Turing Machine. Fix x to be an input of length n and let I be the initial configuration of T on input x. Define the partial functions:

1. Count(k) = number of configurations reachable from I in $\leq k$ steps of T. Count is a total function.
2. Reach(C, k) = true if and only if configuration C is reachable from I in $\leq k$ steps of T. Reach is only defined when it is true.
3. NoReach(C, k) = true if and only if configuration C is not reachable from I in $\leq k$ steps of T. NoReach is only defined when it is true.

We will show that of these partial functions is nondeterministically computable in space $s(n)$. Since $s(n) \geq \log n$ then we can choose a such that the number of distinct configurations is bounded by $2^{a \cdot s(n)}$.

Using the last of these partial functions, an algorithm for $\overline{L}$ is:

- Algorithm for $\overline{L}$:

```
for all configurations C do:
    if C is accepting then
        if NoReach(C,2**(a*s(n)))) then continue
return true
```

Note that there may be computations of $\text{NoReach}(C, 2^{a \cdot s(n)})$ that halt without returning the value true. In such a case the algorithm for $\overline{L}$ halts without returning a value. The only way for the algorithm to return the value true is is for each configuration to be not accepting or to be accepting and not reachable from I in $2^{a \cdot s(n)}$ steps. The space constructibility of $s(n)$ is used to cycle through the configurations that use space $s(n)$.

The nondeterministic algorithms for the three partial functions are:

- Algorithm for $\text{Reach}(C, k)$:

```
run T nondeterministically from I for <= k steps
if C is reached then return true
```

- Algorithm for $\text{NoReach}(C, k)$:

```
if k = 0
   if C != I then return true
if k > 0
   m := 0
   for all configurations D do one of:
      if Reach(D, k-1) then
         if C is not reachable from D in <= 1 step then m := m+1
      continue
   if m = Count(k-1) then return true
```

- Algorithm for $\text{Count}(k)$:

```
m := n := 0
for all configurations C do one of:
     if Reach(C,k) then m := m+1
     if NoReach(C,k) then continue
return m
```

We would like to stress that a computation of $\text{NoReach}(k)$ terminates without without accepting if a call to $\text{Reach}(C, k-1)$ or $\text{Count}(k-1)$ halts without accepting. Similarly, a computation of $\text{Count}(k)$ terminates without without accepting if a call to $\text{Reach}(C, k)$ or $\text{NoReach}(C, k)$ halts without accepting.

The proof of correctness of the algorithm Reach is straightforward. The proof of correctness for NoReach and Count is by induction on k. It should be clear from the algorithms that Reach$(C, 0)$ returns true if and only if $C = I$ and NoReach$(C, 0)$ returns true if and only if $C \neq I$. Hence, Count$(0) = 1$ is computed correctly. For the inductive step, let $k > 0$ and assume that for all C, NoReach$(C, k-1)$ correctly determines if C is not reachable from I in $\leq k-1$ steps. Assume also that Count$(k-1)$ correctly outputs the number of configurations reachable from I in $\leq k-1$ steps. In the algorithm for NoReach(C, k) we examine each configuration D. Assume that C is not reachable from I in $\leq k$ steps. Nondeterministically, call Reach$(D, k-1)$ whenever D is reachable from I in $\leq k$ steps. This call returns true nondeterministically. Since C is not reachable from I in $\leq k$ steps then the second condition passes and we increment m. This m will eventually reach Count$(k-1)$ and the call NoReach(C, k) returns true. Now suppose that C is reachable from I in $\leq k$ steps. There is some D that is reachable from I in $\leq k-1$ steps and C is reachable from D in ≤ 1 step. Hence, m is not incremented for at least one D such that D is reachable from I in $\leq k-1$ steps. Hence m can never attain Count$(k-1)$. Thus, NoReach(C, k) is correct. To argue that Count(k) is correct we just note that for each configuration C exactly one of Reach(C, k) or NoReach(C, k) is true. The counter m must eventually reach the number of configurations reachable from I in $\leq k$ steps in any computation that returns a value.

To complete the proof we argue that the computations can be done in $O(s(n))$ space. The partial functions NoReach(k) and Count(k) are mutually recursive in the following way as described in Figure 1. Hence, we can compute NoReach(C, k) by iteratively computing Count(0), Count$(1), \ldots,$ Count$(k-1)$. The algorithm for $\overline{L}$ executes the computation of NoReach$(C, 2^{cs(n)})$ in this manner. Because $s(n) \geq \log n$, computing Reach(C, k) uses $O(s(n) + \log k)$ space. From Figure 1 we see that we can iterativley compute Count(k) in the space required to store the value of Count$(k-1)$ plus the storage needed to compute Reach(C, i). This is $O(s(n) + \log k)$ space. Hence, $\overline{L}$ can be computed in space $O(s(n) + \log 2^{cs(n)}) = O(s(n))$ space.

Acknowledgements

Thanks to Andrew Keller for helping with the preparation of this paper.

References

[1] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, Vol. 17, pp. 935-938, 1988.

[2] R. Szelepscényi. The method of forcing for nondeterministic automata. *Bulletin of the EATCS*, Vol. 33, pp. 96-100, 1987.

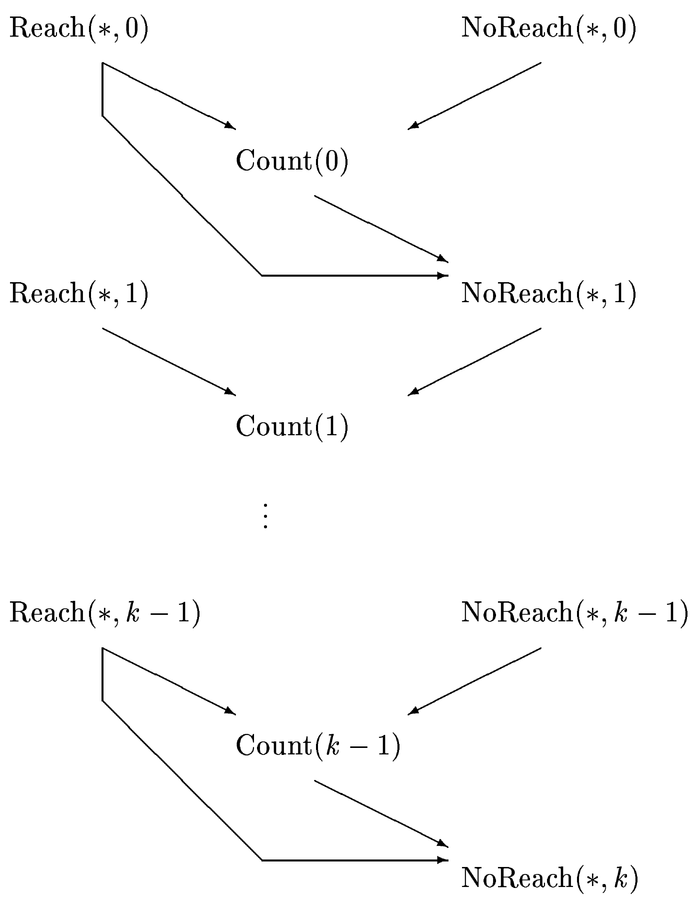

Figure 1: The dependencies between calls to Reach, NoReach, and Count.