MPRI 2.36.1 Project

Thiago F Cesar

1 Introduction

This report documents the process of proving the correctness of an algorithm for solving Takuzu puzzles. It contains explanation for all of the proofs as well as some personal notes on the difficulties faced.

2 Basic Functions on Arrays

2.1 Check for Consecutive Zeros

1) The predicate no_3_consecutive_zeros_sub looks at windows of length three whose indices ranges between $(i = 0, i + 1 = 1, i + 2 = 2)$ and $(i = l - 3, i + 1 = l - 2, i + 2 = l - 1)$, and thus the first index is given by $0 \leq i < l - 2$. The predicate no_3_consecutive_zeros is derived from the previous one, by looking trough the whole array — that is, with $l = a.length$.

2) For the same reason as explained in the previous question, the index i ranges through $0, ..., a.length-3$. The proposed invariant $\forall j. 0 \leq j < i \to \neg\, a[j] = a[j+1] = a[j+2] = 0$ clearly proves the direct implication of the post-condition. Indeed, if the result is true then the loop finished executing and thus with $i = a.length - 2$ we get the definition of no_3_consecutive_zeros. For the reverse implication, it results from the fact that if there are no consecutive zeros then we must finish the loop without an exception.

The invariant is trivially true for $i = 0$. Moreover, if we finish executing the loop without an exception, then $a[i] = a[i + 1] = a[i + 2] = 0$ must be false, and by combining with the previous invariant $\forall j. 0 \leq j < i - 1 \to \neg\, a[j] = a[j + 1] = a[j + 2] = 0$ we see that the invariant is preserved.

3) Differently from the first case, the index i no longer represents the first position of the window, but rather the last: now the window ranges from $(last2 = 0, last1 = 1, i = 2)$ to $(last2 = l - 3, last1 = l - 2, i = l - 1)$. Therefore, the index i ranges through $2, ..., l - 1$.

The first loop invariant $last1 = a[0] \wedge last2 = a[1]$ expresses the fact that the variables $last1$ and $last2$ store the other two positions on the window. It is clearly true for the initial iteration. Moreover, given that after an executing of the loop the current value of $a[i]$ is put in $last1$ whereas the previous value of $last1$ (which is equal to $a[i - 1]$ by the invariant) is put in $last2$, we can also clearly see that the invariant is preserved. This invariant is used in the second one to reason about the positions of the array rather then the values of $last1$ and $last2$, and thus allows us to write the second invariant in a clean way.

The second invariant $\forall j. 0 \leq j < i - 2 \to \neg\, a[j] = a[j + 1] = a[j + 2] = 0$ is virtually identical to the invariant of **2)**, the only difference being that the index i now represents the last position of the window. It is trivially true for the initial case, and if we reach the end of the loop then the exception was not raised and thus we must have $\neg\ a[i] = last1 = last2 = 0$. By the first invariant this means $\neg\ a[i] = a[i-1] = a[i-2] = 0$, and by combining with $\forall j. 0 \leq j < i - 2 \rightarrow \neg\ a[j] = a[j+1] = a[j+2] = 0$ we see the invariant also holds for $i+1$.