# Formal Verification of Neural Networks for Safety-Critical Tasks in Deep Reinforcement Learning - Supplementary Material

**Davide Corsi**[1] **Enrico Marchesini**[1] **Alessandro Farinelli**[1]

[1] Department of Computer Science , University of Verona , 37135, Verona, Italy

name.surname@univr.it

## PROPERTIES ENCODING

In this section we show the formal encoding of the safety properties used in our evaluation. Considering the network architecture, the input space, and the natural language description of the properties in the main paper, we provide the formal encoding of the properties for the *trajectory generation* problem ($\theta_{P,i}$) and the *mapless navigation* environment ($\theta_{T,i}$).

$\theta_{P,0L}$ : If $x_0 \in [1,1] \wedge x_1, ..., x_8 \in \mathcal{D} \Rightarrow y_0 < [y_1, ..., y_{11}], where \mathcal{D} = [0,1)$

$\theta_{P,0R}$ : If $x_0 \in [0,0] \wedge x_1, ..., x_8 \in \mathcal{D} \Rightarrow y_1 < [y_0, y_2, ..., y_{11}], where \mathcal{D} = [0,1)$.

$\theta_{P,1L}$ : If $x_1 \in [1,1] \wedge x_0, x_2, ..., x_8 \in \mathcal{D} \Rightarrow y_2 < [y_0, y_1, y_3, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,1R}$ : If $x_1 \in [0,0] \wedge x_0, x_2, ..., x_8 \in \mathcal{D} \Rightarrow y_3 < [y_0, y_1, y_2, y_4, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,2L}$ : If $x_2 \in [1,1] \wedge x_0, x_1, x_3, ..., x_8 \in \mathcal{D} \Rightarrow y_4 < [y_0, y_1, y_2, y_3, y_5, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,2R}$ : If $x_2 \in [0,0] \wedge x_0, x_1, x_3, ..., x_8 \in \mathcal{D} \Rightarrow y_5 < [y_0, y_1, y_2, y_3, y_4, y_6, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,3L}$ : If $x_3 \in [1,1] \wedge x_0, ..., x_2, x_4, ..., x_8 \in \mathcal{D} \Rightarrow y_6 < [y_0, ..., y_5, y_7, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,3R}$ : If $x_3 \in [0,0] \wedge x_0, ..., x_2, x_4, ..., x_8 \in \mathcal{D} \Rightarrow y_7 < [y_0, ..., y_6, y_8, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,4L}$ : If $x_4 \in [1,1] \wedge x_0, ..., x_3, x_5, ..., x_8 \in \mathcal{D} \Rightarrow y_8 < [y_0, ..., y_7, y_9, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,4R}$ : If $x_4 \in [0,0] \wedge x_0, ..., x_3, x_5, ..., x_8 \in \mathcal{D} \Rightarrow y_9 < [y_0, ..., y_8, y_{10}, ..., y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,5L}$ : If $x_5 \in [1,1] \wedge x_0, ..., x_4, x_6, ..., x_8 \in \mathcal{D} \Rightarrow y_{10} < [y_0, ..., y_9, y_{11}], \mathcal{D} = [0,1)$.

$\theta_{P,5R}$ : If $x_5 \in [0,0] \wedge x_0, ..., x_4, x_6, ..., x_8 \in \mathcal{D} \Rightarrow y_{11} < [y_0, ..., y_{10}], \mathcal{D} = [0,1)$.

*For the mapless navigation scenario we considered the normalized value of 0.1 for a near obstacle.*

$\theta_{T,0}$ : If $x_0, ..., x_{10} \in \mathcal{D}_1 \wedge x_{11} \in \mathcal{D}_2 \Rightarrow y_0 < y_1 \wedge y_2 < y_1, \mathcal{D}_1 = (0.1, 1] and \mathcal{D}_2 = [0,1]$.

$\theta_{T,1}$ : If $x_0, ..., x_3 \in [0, 0.1] \wedge x_4, ..., x_{10} \in \mathcal{D}_1 \wedge x_{11} \in \mathcal{D}_2 \Rightarrow y_0 < [y_1, y_2], \mathcal{D}_1 = (0.1, 1] and \mathcal{D}_2 = [0,1]$.

$\theta_{T,1}$ : If $x_0, ..., x_6 \in \mathcal{D}_1 \wedge x_7, ..., x_{10} \in [0, 0.1] \wedge x_{11} \in \mathcal{D}_2 \Rightarrow y_2 < [y_0, y_1], \mathcal{D}_1 = (0.1, 1] and \mathcal{D}_2 = [0,1]$.

*Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence* (UAI 2021).

## ALGORITHM ANALYSIS

In this section, we first provide a convergence proof, followed by the complexity in terms of time and space.

**Property 1.** *Given an input area and a discretization value $\epsilon$, ProVe always converges in a finite number of steps without loss of precision.*

*Proof.* To prove the convergence in a finite number of steps we demonstrate the following properties: (i) there are no infinite loops in the iterative splitting; (ii) splitting an area into two sub-areas covers all the discrete values of the range up to precision $\epsilon$. For the latter, we use $\epsilon$ in the computation of the mean value because the mean of an area could have a higher precision with respect to $\epsilon$, producing an infinite loop. This is due to the fact that the discrete values at a higher precision than $\epsilon$ are odd (it is ambiguous to divide it into equally sized groups).

Suppose an $\epsilon = 0.1$ and a split operation of the range [0.0,0.1]; we first compute the mean 0.05, splitting in [0.0,0.05] and [0.05,0.1]. We then round these values to the specified precision, obtaining [0.0,0.1] and [0.1,0.1]. The introduction of $\epsilon$ allows to rounding alternatively up and down in order to avoid the infinite recursion. For the former, it is trivially true that we cover all the discrete values of the range up to precision $\epsilon$, because if the mean between the extreme values of the range is at precision $\epsilon$, then the union of the sub-areas returns the original one. Moreover, if the mean has higher precision than $\epsilon$, we round the mean value once to the next bigger value at precision $\epsilon$ and once to the smaller ones. In particular, there is no value at precision $\epsilon$ that is between these two. □

**Time Complexity** Given the number of ranges in input $n$ and the discretization value $\epsilon$, we define the range size as $l$ (fixed to 1 by normalization) and the time cost of the control operation (propagation and property verification) as $op$. The time required for a matrix multiplication is $\mathcal{O}(mnp)$ where the input matrices are $m \times n$ and $n \times p$. The structures of the matrices used by ProVe are always in form of $m \times n$ and $n \times 2n$, where $m$ is dependent from the iterations of our algorithm; so the time required for a multiplication is $\mathcal{O}(n^2 m)$. For construction, we know that the number of rows $(m)$ double up at each iteration, i.e., $m = 2^i$ where $i$ is the number of iteration already performed. The time complexity of the algorithm becomes $\mathcal{O}(\sum_{k=1}^{I} n^2 2^{(k-1)})$ that can be written as $\mathcal{O}(n^2 \sum_{k=1}^{I} 2^k 2^{-1})$. Applying Gauss formula $\mathcal{O}(\frac{n^2}{2} 2^{\frac{I(I+1)}{2}})$ where $I$ is the total number of iterations of the algorithm. The total number of iteration required from the algorithm depends from the range size $l$ and the precision $\epsilon$; in fact, each iteration divides an area into two equally sized sub-areas. For a single input range that starts from 0 to 1 we need $\mathcal{O}(\log_2 \lceil \frac{l}{\epsilon} + 1 \rceil)$ and the total number of operation is $I = \mathcal{O}(n \log_2 \lceil \frac{l}{\epsilon} + 1 \rceil)$. Finally, the time complexity of the algorithm is $\mathcal{O}(op \frac{n^2}{2} 2^{\frac{(n \log_2 \lceil \frac{1}{\epsilon} + 1 \rceil)^2}{2}})$. Summarizing, the time complexity of ProVe is exponential both when $\epsilon$ decreases and $n$ increases.

**Space Complexity** Recalling the notation of the previous section and the fact that at each iteration we double up the number of rows of our matrix, ProVe requires $\mathcal{O}(2^{I^n})$ rows and always $n$ columns for the matrix. The total space required is then $\mathcal{O}(n 2^{I^n})$ that is $\mathcal{O}(n \lceil \frac{1}{\epsilon} \rceil^n)$.

This analysis further motivates the introduction of the $\epsilon$ discretization value to limit the depth of the recursion and the search of a heuristic, to limit the complexity of verification approaches.