coding feasible in practice. They used integer representations for probabilities with scaling techniques. The initial interval $[0, 1)$ was substituted by $[0, W)$, where $W = 2^p$, $p \geq 2$ being the bit size of the initial interval. The scaling is performed by doubling the size of the interval $I(s) = [Low, High)$ when one of the following conditions holds:

- **E1**: $0 \leq High < W/2$ : We double $Low$ and $High$ and we output 0 followed by $U_3$ ones, then $U_3$ is reset to 0.

- **E2**: $W/2 \leq Low < W$ : We double $Low$ and $High$ after substracting $W/2$ and we output 1 followed by $U_3$ zeros, then $U_3$ is reset to 0.

- **E3**: $W/4 \leq Low < W/2 \leq High < 3W/4$ : We double $Low$ and $High$ after substracting $W/4$ and we increase $U_3$ by 1 (no output).

Note that $U_3$ represents the number of the last E3 scalings done, and is initialized to 0.

The described arithmetic code is based on the binary source statistics, and it is essential that, for an encoded symbol index $i$, the encoder and the decoder use the same probabilities $p_0$ and $p_1$. Static arithmetic coding supposes that source statistics were transmitted to the decoder with no error, which results in additional bits and consequently a compression loss. Such situation is outperformed with adaptive arithmetic coding, where $p_0$ and $p_1$ are initialized to 0.5, then, for every symbol encoding step they are updated. Such scheme induces no remarkable compression loss when long source symbol sequences are