Let \( n \) be a natural variable. Add time according to the recursive measure, and find a finite upper bound on the execution time of

\[
P \iff \text{if } n \geq 2 \text{ then } n := n-2. \ P. \ n := n+1. \ P. \ n := n+1 \text{ else } \text{ok}\]

To ensure that every loop includes a time increment, it is sufficient to put \( t := t+1 \) just before the first call. (But the question isn't any harder, and the time bound isn't significantly different, if we put \( t := t+1 \) before both calls.) Because of the two calls, each at approximately the original value of \( n \), I guess the time might be exponential. Actually, it looks just like Fibonacci: the first call is at \( n-2 \), the second is at \( n-1 \). Let's try

\[
P = t' = t + 2^n\]

The proof of the refinement will be by cases. First case:

\[
n \geq 2 \land (n := n-2. \ t := t+1. \ P. \ n := n+1. \ P. \ n := n+1)\]

\[
\implies n \geq 2 \land (t' \leq t + 1 + 2^{n-2}. \ t' \leq t + 2^{n+1}. \ n' := n+1 \land t' = t)\]

\[
\implies n \geq 2 \land \exists n'', t', n''', t'''. \ t''' \leq t + 1 + 2^{n-2} \land t'''' \leq t'' + 2^{n+1} \land n'' = n'''' + 1 \land t' = t''''\]

\[
\implies n \geq 2\]

Oops. The final time seems to be completely arbitrary. The problem is that the first call of \( P \) allows \( n \) to change arbitrarily, so the last call of \( P \) allows \( t \) to change arbitrarily. Let's try again.

\[
P = n' = n \land t' \leq t + 2^n\]

The proof of the refinement will be by cases. First case:

\[
n \geq 2 \land (n := n-2. \ t := t+1. \ P. \ n := n+1. \ P. \ n := n+1)\]

\[
\implies n \geq 2 \land n' = n \land t' \leq t + 1 + 2^{n-2} + 2^{n-1}\]

\[
\implies n \geq 2 \land n' = n \land t' \leq t + 1 + 3 \times 2^{n-2}\]

\[
\implies n' = n \land t' \leq t + 2^n\]

Last case:

\[
n < 2 \land \text{ok}\]

\[
\implies n < 2 \land n' = n \land t' = t\]

\[
\implies n' = n \land t' \leq t + 2^n\]

and \( 0 \leq 2^n \)