
Rule 0.20 *⟨Limit Number of Modifiers⟩*

<pre> [...] contract A { [...] modifier m₁() { checks₁; _; } modifier m₂() { checks₂; _; } ... function f() m₁ m₂ ... { stmts } [...] }</pre>	=	<pre> [...] contract A' { [...] modifier m() { checks₁; checks₂; ... _; } function validate() { checks' } function f() m { validate(); stmts } [...] }</pre>
--	---	---

where

- m_1, m_2, \dots, m_k are modifiers in contract A applied to function f ;
- m is a consolidated modifier in contract A' that combines multiple checks;
- $checks_i$ represents the validation logic in modifier m_i ;
- $checks'$ represents additional validation logic moved to a function;
- $validate()$ is a helper function that performs some of the validation checks;
- $stmts$ represents the function body statements;
- $_$ denotes the placeholder in modifiers where the function body is executed.

provided

- The modifiers m_1, \dots, m_k perform related validation checks;
- Consolidating modifiers into m or moving checks to $validate()$ maintains the same validation logic;
- The order of checks is preserved to maintain security properties;
- All validation conditions remain equivalent before and after transformation;
- The consolidated approach reduces function call stack depth;
- Access control and validation semantics are preserved.

Invariant:

- Let s_i and s'_i be the initial state of A and A' , respectively.
- Let s_f and s'_f be the state reached by A and A' , respectively, after $A.f()$ and $A'.f()$ are executed from s_i and s'_i , respectively.
- Then, the coupling invariant is

$$\forall s_i, s'_i . (s_i = s'_i) \rightarrow (s_f = s'_f)$$
