**Testarea unor smartcontracte Ethereum**

- Testare structurală –

**Introducere**

Aplicația pe care o avem de testat implementează prin intermediul mai multor smartcontracte o piață de tokeni ERC20. Utilizatorii pot creea tokeni ERC20, dar și Liquidity Pool-uri (LP) ce funcționează drept o stație de schimb valutar. Practic, un LP conține doi tokeni în anumite cantități, iar utilizatorii pot să depună lichiditate (fonduri) în aceste LP-uri, dar și să le folosească pentru a schimba o anumită cantitate din tokenA în tokenB. Pentru ca acest schimb să se realizeze, trebuie să existe, inițial, un LP care schimbă din tokenA în tokenB. Cum acest lucru nu este mereu posibil, am ales să implementăm contractul LPRouter, ce calculează drumul minim (ca număr de LP-uri) pentru schimbul între tokenA și tokenB. Astfel, nu mai este nevoie ca cei doi tokeni să fie “conectați” direct printr-un LP, ci doar să existe un drum între cei doi tokeni în graful descris.

**Testare**

Din contractul LPRouter am ales să realizăm testarea structurală pentru funcția ce calculează drumul minim între cei doi tokeni. Codul aceste funcții se află TODO.

Pentru început, am construit Control Flow Graph-ul (CFG) asociat funcției pe care o testăm din smartcontract-ul LPRouter. Graful respectiv se află pe pagina imediat următoarei celei ce conține codul funcției.

Abordarea pe care am ales-o pentru a testa structural funcția respectivă folosește tehnica **acoperirii la nivel de condiție/decizie (condition/decision coverage)**. În acest sens, am construit și tabelul de mai jos, ce conține fiecare decizie pe care algoritmul le conține, dar și condițiile individuale din aceste decizii.

|  |  |
| --- | --- |
| **Decizii** | **Condiții individuale** |
| **1D** while (!queue.empty()) | **1C** !queue.empty() |
| **2D** if (current == address(\_tokenB) | **2C** current == address(\_tokenB) |
| **3D** for (uint i = 0; i < allTokens.length; i++) | **3C** i < allTokens.length |
| **4D** if (pools[current][neighbor] != address(0) && !visited[neighbor]) | **4C** pools[current][neighbor] != address[0];  **5C** !visited[neighbor] |
| **5D** if (!foundPath) | **6C** !foundPath (true sau false dacă nu s-a găsit nici un drum, respectiv dacă a fost găsit) |

Acum că avem deciziile și condițiile separate într-un tabel, putem să trecem la scrierea de teste. În tabelul de mai jos pot fi observate testele pe care le-am creat, rezultatele acestora (valoarea returnată de către funcție), dar și deciziile și condițiile individuale pe care fiecare dintre acestea le acoperă.

(Notiță: Mapping-ul cu valori booleene **visited** face parte din starea contractului, dar înainte de fiecare apel al funcției minPath va conține doar valori false, pentru ca la finalul acestei funcții se apelează clearVisited, iar nici o altă funcție nu manipulează această variabilă. De asemenea, **queue** este o coadă pre-implementată în alt contract, iar aceasta va fi întotdeauna goală la începutul executării funcției, datorită apelului queue.clear(). Mapping-ul cu adrese **previous** este folosit pentru construirea drumului, valorile anterioare ale acestuia fiind neimportante, deoarece va exista un unic drum de la tokenA la tokenB, dat prin acest mapping ce se va actualiza la fiecare iterație, pentru ca în final valorile din acest mapping să fie cele corecte pentru nodurile de pe drumul respectiv).

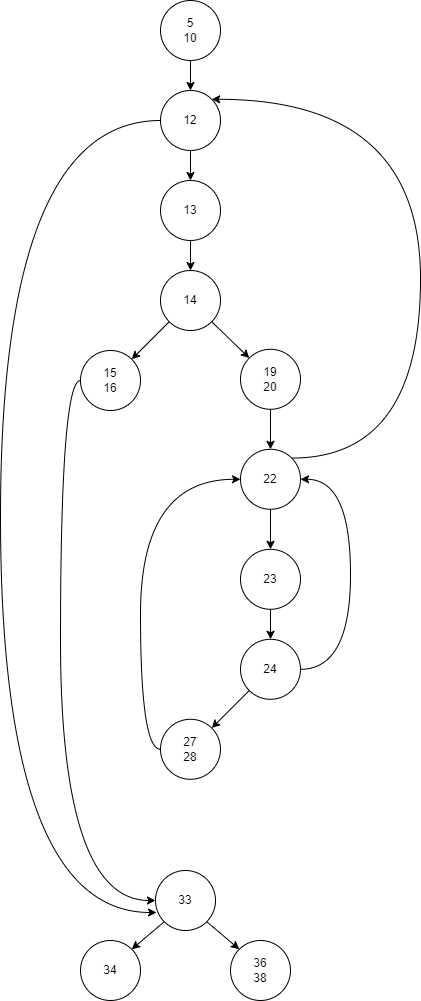
*allTokens* reprezintă un array ce conține toți tokenii pentru care au fost create LP-uri.

*pools* reprezintă un mapping ce tine minte pentru fiecare pereche de tokeni LP-ul care îi “leagă” (un fel de matrice de adiacență pentru un graf neorientat). Acest mapping va fi reprezentat în tabelul de mai jos drept o listă de perechi (neorientate) de tokeni între care există un LP.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Starea contractului (storage)** | | **Intrări** | | **Rezultat** | **Condiții individuale acoperite** | **Decizii acoperite** |
| *pools* | *allTokens* | *\_tokenA* | *\_tokenB* | ---------- | ---------------- | ----------- |
| [(tokenA, tokenB), (tokenB, tokenC)] | [tokenA, tokenB, tokenC] | tokenA | tokenC | [tokenA, tokenB, tokenC] | 1C, 2C, 3C, 4C, 5C;  6C – doar false; | 1D, 2D, 3D, 4D;  5D – doar false; |
| [(tokenA, tokenB)] | [tokenA, tokenB] | tokenA | tokenB | [] | 1C, 3C, 4C, 5C;  2C – doar false;  6C – doar true; | 1D, 3D, 4D;  2D – doar false;  5D – doar true; |

După cum se poate observa și din tabel, aceste două teste sunt suficiente pentru a acoperi atât deciziile, cât și condițiile individuale ce apar în funcția minPath.

**Codul funcției de testat:**

**CFG:**