Calcolo del Cost Model Esatto di un Programma in Solidity

Mattia Guazzaloca

Alma Mater Studiorum - Università di Bologna

13 Ottobre 2021

Sommario

Background

Ethereum

Ethereum è una **piattaforma** open-source basata su blockchain per lo sviluppo di applicazioni distribuite denominate **smart contracts**.

L'Ether e il Gas

L'**Ether** (ETH) è la criptovaluta di Ethereum. Viene usata specialmente per l'acquisto del **gas**, l'unità di misura dello sforzo computazionale richesto per l'esecuzione di uno smart contract.

Solidity

Solidity è il linguaggio di alto livello per lo sviluppo di smart contract più utilizzato. È object-oriented e la sua sintassi è largamente ispirata a quella del linguaggio JavaScript.

CerCo

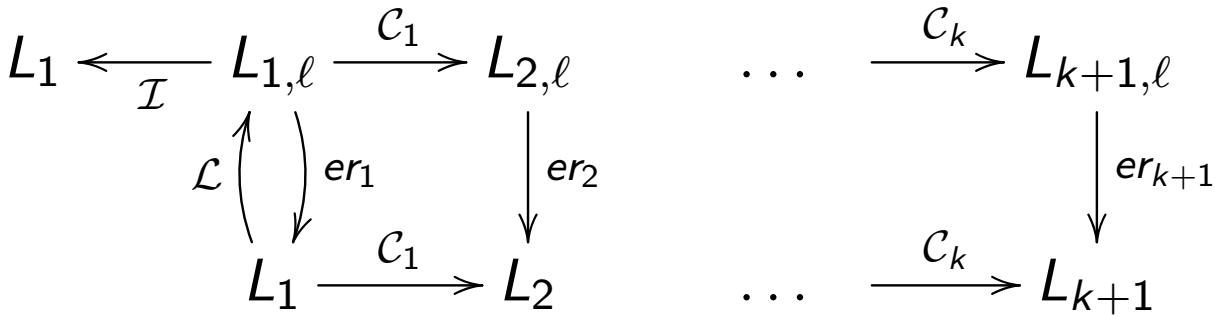
CerCo è un progetto europeo di ricerca nato con lo scopo di sviluppare e provare formalmente un **cost annotating compiler**.

Cost Annotating Compiler

Un *cost annotating compiler* C è un compilatore che preso in input un programma P produce in output un programma $An(P)$ "annotato", funzionalmente equivalente a P, ma capace di tracciare il costo della sua esecuzione.

Le informazioni che il compilatore C inserisce nel programma annotato $An(P)$ per tracciare l'esecuzione si chiamano **annotazioni di costo**.

Il Metodo del Labelling



Labelling

Un *labelling* L di un linguaggio sorgente L_i è una funzione tale che $er_{L_i} \circ L$ è la funzione identità

Instrumentazione

Una *instrumentazione* I di un linguaggio etichettato $L_{1,\ell}$ è una funzione che rimpiazza le label di $L_{1,\ell}$ con, per esempio, incrementi di una variabile di costo

Implementazione

Scopo della Tesi

Realizzare un **cost annotating compiler** per il linguaggio Solidity applicando l'approccio sviluppato da CerCo.

Risultato

La complessità di Solidity e le peculiarità degli smart contract hanno generato problemi nuovi rispetto a CerCo e sono causa delle limitazioni dell'attuale implementazione.

Labelling

- L'inizio di **ciascun blocco** è etichettato con una label
- Tutte le sequenze di istruzioni precedute da un qualche costrutto per il **controllo di flusso** sono etichettate con una label posta subito dopo la fine del costrutto

```
1  contract Fibonacci {
2    function fibonacci(uint256 n)
       public pure returns (uint256 b)
       {
3      /* __cost0 */
4      if (n == 0) {
5        /* __cost1 */
6        return 0;
7      }
8      /* __cost2 */
9      uint256 a = 1;
10     b = 1;
11     for (uint256 i = 2; i < n; i++) {
12       /* __cost3 */
13       uint256 c = a + b;
14       a = b;
15       b = c;
16     }
17     /* __cost4 */
18     return b;
19   }
20 }
```

Calcolo dei Costi

Funzione di Costo del Gas

La funzione di costo del gas $C(\sigma, \mu, w)$ associa all'istruzione w il consumo di gas dato lo stato della blockchain σ e quello della EVM μ:

$$C(\sigma, \mu, w) \equiv C_{\text{mem}}(\mu_i') - C_{\text{mem}}(\mu_i) + \begin{cases} C_{\text{SSTORE}}(\sigma, \mu) & \text{if } w = \text{SSTORE} \\ G_{\text{verylow}} + G_{\text{copy}} \times \lceil \mu_s[2] \nabla \cdot 32 \rceil & \text{if } w \in W_{\text{copy}} \\ C_{\text{CALL}}(\sigma, \mu) & \text{if } w \in W_{\text{call}} \\ C_{\text{SELFDESTRUCT}}(\sigma, \mu) & \text{if } w = \text{SELFDESTRUCT} \\ G_{\text{create}} & \text{if } w = \text{CREATE} \\ G_{\text{zero}} & \text{if } w \in W_{\text{zero}} \\ G_{\text{base}} & \text{if } w \in W_{\text{base}} \end{cases}$$

dove:

$$C_{\text{mem}}(a) \equiv G_{\text{memory}} \cdot a + \left\lfloor \frac{a^2}{512} \right\rfloor$$

Soluzione adottata:

- Simulazione **parziale** dell'esecuzione
- Assegnamento diretto dei consumi per le istruzioni con costo costante
- Rapprensetazione di $C(\sigma, \mu, w)$ per le istruzioni più complesse

Attribuzione dei Costi

1. Si costruisce il **Control Flow Graph** del sorgente assembly

Control Flow Graph

Dato un qualche programma P, si definisce *Control Flow Graph* il grafo orientanto $G = (N, E)$ dove ciascun nodo $n \in N$ è un blocco del programma P e ogni arco $(n_i, n_j) \in E$ rappresenta il passaggio del flusso di esecuzione dal nodo n_i al nodo n_j.

2. Si visita in profondità il CFG mantenendo uno stack μ delle label raggiunte. Durante la visita:
   - i nodi **etichetta**, non precedentemente visitati, vengono aggiunti a μ
   - il consumo di gas dei nodi **istruzione** viene sommato all'etichetta che si trova in cima a μ

Costo dell'Inizializzazione

Ogni smart contract necessita di codice aggiuntivo per:

- Il deploy, i.e. il caricamento del contratto sulla blockchain
- La corretta invocazione dei metodi parte dell'API pubblica del contratto

Il codice generato non è coperto da label e genera quindi annotazioni di costo *unsound*.

`startupCost`

L'algoritmo per l'attribuzione dei costi definisce una variabile ausiliaria `startupCost` che tiene traccia del costo dell'inizializzazione. Il suo valore viene assegnato alla prima label incontrata durante la visita.

Instrumentazione

1. Aggiunta del codice per definire ed accedere alla **variabile di costo**
2. Sostituzione delle label con i corrispettivi **incrementi** della variabile di costo

```
1  contract SimpleStorageInstrumented {
2      uint256 storedData;
3
4      // NOTE: was a view function
5      function get() internal returns (uint256) {
6          /* __cost1 */
7          __costAcc = __costAcc + 874; /* +
               memaccess(128, add(not(127),
               abi_encode_uint256(sload(0)))) */
8          return storedData;
9      }
10
11     function get_external() external returns (
           uint256) {
12         /* __cost1.startup */
13         __costAcc = __costAcc + 155;
14         return get();
15     }
16
17     // Instrumentation Machinery
18     uint256 private __costAcc = 0;
19     function __SimpleStorage_getCost() public
           view returns(uint256) {
20         return __costAcc;
21     }
22     function __SimpleStorage_resetCost() public
           {
23         __costAcc = 0;
24     }
25 }
```

Conclusioni

Stimare in maniera chiara e precisa i consumi di gas degli smart contract è un compito non triviale, specialmente quando questi sono scritti in un linguaggio di alto livello come Solidity. L'implementazione presentata risolve in parte il problema grazie al lavoro svolto dal progetto CerCo.

Attuali limitazioni:

1. L'instrumentazione è **incompleta**: manca ad esempio il supporto per le `library`
2. Il calcolo dei consumi di gas rappresenta solo un'indicazione allo sviluppatore per via di quelle istruzioni il cui costo dipende dallo stato della EVM

Inoltre, principalmente a causa del punto 2, il testing dell'implementazione è avvenuto solo in maniera parziale.

Metriche sul Lavoro Svolto

In totale lo sviluppo dell'elaborato ha richiesto all'incirca **5 mesi**, da maggio a settembre compresi, per quanto le ore effettive di lavoro siano chiaramente in numero inferiore, ma difficilmente quantificabili.

La scrittura del codice ha riguardato due aree distinte:

- Il **compilatore di Solidity** (solc): modifiche minime al sorgente in C++ per un totale di circa **100-200 LoC**.
- **Cerco2**: tool esterno al compilatore principalmente legato alla fase di **calcolo e attribuzione dei costi**. Questo tool è stato scritto in JavaScript ed ha una dimensione pari a circa **2000 LoC**.

Grazie!