Capitolo 1

Generazione di WebAssembly

La ricerca di soluzioni efficienti e portabili per eseguire codice in ambienti diversi è diventata una priorità fondamentale. In questo contesto, WebAssembly (Wasm) emerge come una tecnologia chiave, fornendo un formato binario sicuro, veloce e indipendente dalla piattaforma. Nel corso di questo capitolo, esploreremo il processo di generazione di codice WebAssembly attraverso un compilatore dedicato a un linguaggio personalizzato. Il nostro linguaggio, creato per soddisfare specifiche esigenze o paradigmi di programmazione unici, si propone di offrire una flessibilità senza precedenti agli sviluppatori. Attraverso un compilatore appositamente progettato, saremo in grado di tradurre il codice sorgente del nostro linguaggio in istruzioni Wasm, consentendo così l'esecuzione di programmi in un ambiente virtuale altamente performante e sicuro. Nel corso di questo capitolo, esamineremo in dettaglio il processo di compilazione, passando attraverso le fasi cruciali che trasformano il nostro codice sorgente in un modulo WebAssembly. Dalla rappresentazione intermedia alla gestione delle dipendenze, esploreremo come il compilatore si adatta alle specificità del nostro linguaggio per garantire una corretta esecuzione e ottimizzazione delle risorse. Il capitolo si propone inoltre di approfondire le sfide e le opportunità che emergono durante il processo di generazione di WebAssembly. Analizzeremo le scelte di progettazione del compilatore, l'ottimizzazione del codice e la gestione delle risorse, fornendo un quadro completo delle considerazioni che guidano il nostro approccio alla generazione di codice Wasm.

1.1 Introduzione WebAssembly

WebAssembly(Wasm) [2] è un formato di istruzioni portabile, sicuro e ad alte prestazioni, progettato per essere eseguito in ambienti virtuali. Il formato

è stato sviluppato da un gruppo di lavoro congiunto tra Google, Mozilla, Microsoft e Apple, con l'obiettivo di fornire un formato binario sicuro, veloce e indipendente dalla piattaforma. Il formato è stato progettato per essere eseguito in ambienti virtuali, come browser web, ma può essere utilizzato anche in altri contesti, come ad esempio server, dispositivi IoT e applicazioni desktop. Le istruzioni Wasm si differiscono dalle istruzioni di un processore reale, in quanto sono progettate per essere eseguite in un ambiente virtuale. Questo significa che le istruzioni Wasm non sono direttamente eseguibili da un processore fisico, ma devono essere prima tradotte in istruzioni native. Questo processo di traduzione è gestito da un motore di runtime, che si occupa di interpretare le istruzioni Wasm e di tradurle in istruzioni native. Il motore di runtime è responsabile anche di gestire la memoria e le risorse del sistema, fornendo un'astrazione sicura e indipendente dalla piattaforma.

1.2 WebAssembly Text Format

Il formato di testo WebAssembly (WAT) [**WebAssemblyTextFormat**] è un formato di testo leggibile dall'uomo per la rappresentazione di moduli WebAssembly. Il formato è stato progettato per essere utilizzato come rappresentazione intermedia durante il processo di compilazione, fornendo un'astrazione leggibile dall'uomo per il codice Wasm. Il compilatore che abbiamo sviluppato generiamo un file .wat ed in seguito il tool wat2wasm [1] genera il file .wasm, che a sua volta potrá essere eseguito da un motore di runtime.

1.2.1 WebAssembly Text Format

I tipi di dati che troviamo in wat sono:

- **i32** 32-bit integer
- **i64** 64-bit integer
- **f32** 32-bit float
- **f64** 64-bit float

Un singolo parametro *(param i32)* e il tipo di ritorno *(result i32)*.

```
1    (func (param i32) (param i32) (result f64) ...)
```

Listing 1.1: Esempio di funzione in wat

I parametri locali possono essere dichiarati all'interno di una funzione, e sono accessibili solo all'interno della funzione stessa. I comandi *local.get* e *local.set* vengono utilizzati per accedere agli indici dei parametri locali. Possiamo usare anche l'operatore $ per accedere ai parametri locali, in maniera più human-readable.

```
1    (func $fun (param i32) (param i32) (result f64)
2        (local $par1 i32)
3        (local $par2 i32)
4        (local $par3 f64)
5        ...
6        (local.get $par1)
7        (local.get $par2)
8        (local.set $par3)
9        ...
10   )
```

Listing 1.2: Esempio di funzione in wat

Come vediamo in questo esempio la funzione $fun prende in input due parametri di tipo intero e ritorna un valore di tipo float. Inoltre all'interno della funzione vengono dichiarati tre parametri locali, due di tipo intero e uno di tipo float.