05.1-LLVM-IR

Struttura di una file di bitcode LLVM

> LLVM (Low-Level Virtual Machine) è un infrastruttura di compilazione modulare e riutilizzabile. In sostanza, è un insieme di strumenti e librerie che consentono di costruire compilatori, ottimizzatori e altri strumenti legati alla programmazione. È progettato per essere altamente flessibile e supportare un'ampia gamma di linguaggi di programmazione.

Un file di **bitcode LLVM** è la rappresentazione binaria del codice IR, serve come formato intermedio che può essere usato per:

- **Memorizzare il programma** durante le fasi di compilazione;
- **Riutilizzare** il codice ottimizzato;
- **Eseguire analisi o trasformazioni** su moduli già generati;
  Un file di bitcode rappresenta un **modulo LLVM**, che è una collezione di oggetti del programma (funzioni, variabili, metadati, ecc.). La struttura è composta da:
- Variabili globali: possono avere diverse modalità di collegamento (linkage), come:
  - `external`: accessibile ad altri moduli;
  - `internal`: visibile solo all'interno del modulo;
  - `weak`: permette risoluzioni multiple (utile per le librerie condivise);
- Funzioni globali;
- Metadata;

Identificatori

Gli **identificatori** sono utilizzati per fare riferimento a **tipi**, **valori** e altre entità (come funzioni o variabili globali) presenti nel codice intermedio.

- entità visibili a livello **globale**: il nome inizia con il carattere @;
- entità **locali**, che esistono solo all'interno di una funzione: il nome inizia con il carattere %;
- Identificatori "anonimi": il nome è un intero non negativo
  - @123, %0, %1, %2;
- Identificatori con nome usano la RE: {[%@] [-a-zA-Z$._] [-a-zA-Z$._0-9]*}
  - @x, @main, @bond.james.bond.007;

Tipi di dato semplici

I tipi di dato semplici rappresentano i blocchi fondamentali per definire variabili, valori e operazioni.

- void (nessun valore): void;
- interi (con dimensione *N* in bits): i*N*
  - i1, i8, i23, i32, i123456;
- floating point (dimensione standard):
  - half, float, double, fp128;
  - anche specifici: x86_fp80, ppc_fp128;
- puntatori: type *
  - i32*
- label, token;

Tipi di dato composti

- array: [n x type]
  - [10 x i32]
  - [10 x [20 x i8]]
- vector: <n x type>
- struct: {typelist}
  - i32, i8*, float, [5 x i32]
  - <{i16, i8, i32}>; packed
- funzioni: ret-type (typelist)
  - i32 (int32, i1)
  - i32 (int8*, ...)
- metadata, ...

Target Triple e Data Layout

Sono configurazioni fondamentali che definiscono le caratteristiche del sistema target su cui il codice generato verrà eseguito. Essi guidano il compilatore nella generazione di codice ottimizzato per una specifica piattaforma.

Target Triple

La **target triple** è una stringa che identifica il sistema per cui il codice deve essere compilato. La stringa è composta da tre (o più) componenti separati da trattini (`-`), che specificano l'architettura, il produttore, il sistema operativo e, opzionalmente, l'ambiente. Ecco un qesempio: `x86_64-pc-linux-gnu`.

Data Layout

Il **data layout** è una stringa che descrive il formato di memoria e le regole di allineamento per i tipi di dati sulla piattaforma target. Definisce:

- L'endianness (ordine dei byte).
- La dimensione e l'allineamento dei tipi di dato.
- La granularità delle operazioni di memoria.
  Vediamo un esempio: `e-m:e-i64:64-f80:128-n8:16:32:64-S128`

Costanti

- booleane (tipo i1): true, false
- intere: 4, 56, -1234 (anche negative)
- floating point: 123.0, 1.5e12
- puntatore: null, @global
- array: [i32 3023, i32 -12, i32 18]
- vector: <i32 3, i32 -12, i32 18, i32 4096>
- struct: {i32 3, float 2.5, i32* @global}
- zeroinitializer
- undef

La struttura di una funzione

Una **funzione** è una sequenza di basic block (BB), quindi una sequenza di istruzioni, solamente l'ultima istruzione del BB (terminatore) cambia il flusso di esecuzione (determina il BB successivo o l'uscita dalla funzione). Il codice è in forma SSA (Static Signle Assignment),

per generare i nomi locali al BB (tipi, valori, etichette) si usa un unico contatore (poco leggibile), l'opzione `-fno-discard-value-names` cerca di mantenere i nomi locali (più leggibile).

Istruzioni LLVM

Consiglio la visione del pdf fornito dal prof. Zaffanella per la carrellata di istruzioni fornite da LLVM, il pdf in riferimento è chiamato `21-LLVM-IR.pdf`.