MI-SP-SP-11 (MI-GEN) (přednášky 5 a 6)

Generování cílového kódu pokrýváním syntaktického stromu a syntaxí řízeným překladem.

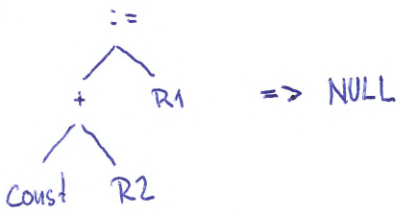
- generování optimálního kódu je NP-hard problém
  + závislé na architektuře a jejích vlastnostech

Naive approach - 1:N - N instrukcí je vygenerováno pro každou instrukci IR
→ simple and inefficient

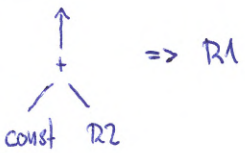
Conventional approach M:N → must use heuristic

- IR is an AST - each target instruction has a corresponding tree pattern
→ we match parts of the tree until it is removed (reduced to Null)

ST const (R2), R1



LD const (R2), R1



ADD R2, R1

→ sometimes we can match such that it is not possible to reduce the tree
→ backtracking and new try
- try to use the cheapest patterns to cover the tree

Graham - Glanville

- grammar rule for each machine instruction → LR(0) parser is constructed
→ reductions by rule → given instruction is selected
- grammar is generally ambiguous and contains conflicts → heuristic

grammar is constructed: (prefix notation)

R2 → := R1 {LD R2, R1}

R1 → + const R1 {ADD #const R1}

Null → := const R2 R1 {ST const (R2), R1}

→ construct LR(0) parser

→ handle conflicts such that minimum # of instructions will be generated

- shift-reduce conflict → shift is preffered
- reduce-reduce conflict → biggest right-hand side is preffered

Dynamic programming - total cost of all patterns is to be minimal one

- produces optimal tiling (good for CISC CPUs)

→ bottom-up traversal - calculate prices of each subtree that can be matched

→ 2-traversal - generate target code → go all the way to root

Maximal munch method

- tree top-down method
- tree pattern for each machine instruction → sort largest first

→ from root, cover whole tree (subtrees)

- good for RISC CPUs, not optimal tiling

SSA & Phi function

Single static assignment - each variable is always assigned just once

- Phi function - merger of values from different branches