Compilers: SCIL Code Generation Invariants

a topic in

DM565 – Formal Languages and Data Processing

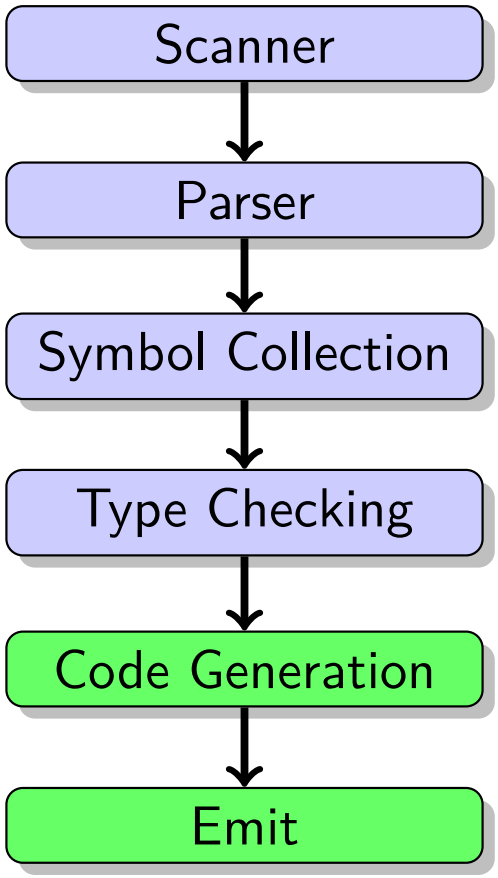
Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk

October, 2019

Compiler Phases



Invariants

- Without formulating this very formally, *invariants* are conditions one can rely one throughout the compiler.
- We establish/maintain invariants by following protocols in various situations.
- Work has to be done to ensure that all constructs adhere to the protocols.
- The benefit is that it is much easier to implement constructions when we know some invariants we can rely on.

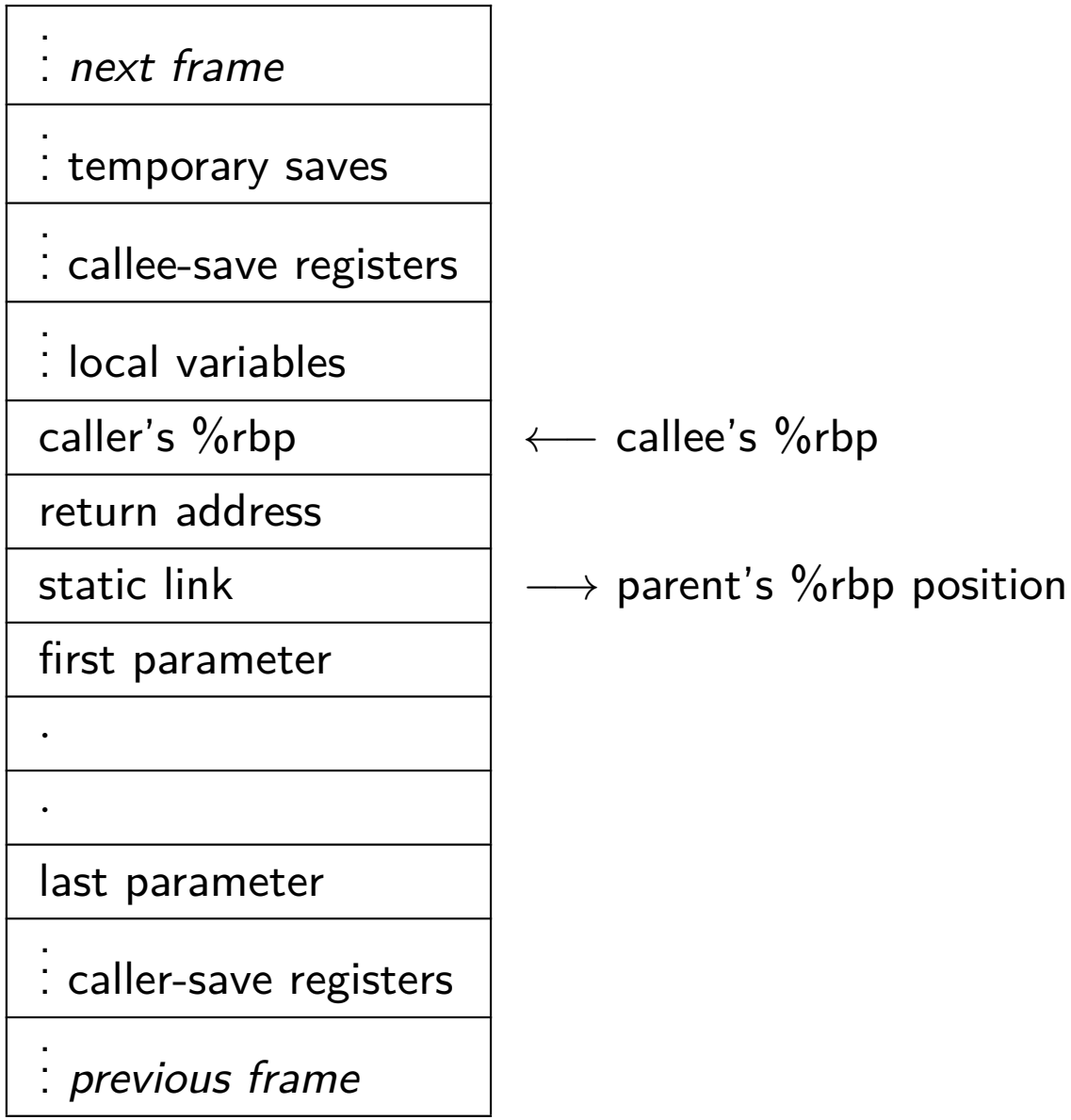
Concrete Invariants in SCIL

- The stack organization in the form of stack frames.
- Constructions "consuming" a number of operands pop them from the stack; the stack top is the last argument.
- Constructions "producing" a result push this onto the stack.
- For constructions with multiple operands, the operands are processed left-to-right.
- Function calls use the stack for all parameters.
- A function's return value is stored in `%rax`.
- We use 64 bits (8 bytes) for all types of values.

Some of these invariants were chosen for simplicity at some (fairly limited) cost in efficiency.

One conclusion is that no registers contain values that will be used again after a function call.

The Stack Frame (Activation Record)



Stack Frame Protocols: Caller

SDU

1. (push caller-save registers)
2. push parameters in reverse order
3. compute and push static link
4. call function [return value in `%rax`]
5. pop/deallocate static link and parameters
6. (pop/deallocate caller-save registers)

Stack Frame Protocols: Callee

SDU

1. push %rbp
2. movq %rsp, %rbp
3. allocate stack space for local variables
4. (push callee-save registers)
5. perform computation:
   - parameter 1 in 24(%rbp)
   - return value in %rax
6. (pop callee-save registers)
7. deallocate local variables
8. pop %rbp
9. ret

Understanding SCIL Generated Code

SDU

- `./compiler < scil_program.src > file.s` writes assembler code to `file.s` (unless the program was bugged), so you can look at it. Attempts have been made to produce also well-commented assembler code to the extent possible for generated code.
- If you want to see how other constructions are handled, you can write small C-programs: `gcc -S program.c` produces a file `program.s` you can inspect (but simplicity has been traded for efficiency). You sometimes get a few comments with `gcc -S -fverbose-asm program.c`.
- When writing/understanding assembler code, draw the stack and register content (or use a debugger for the same purpose).