Low-Level Virtual Machine

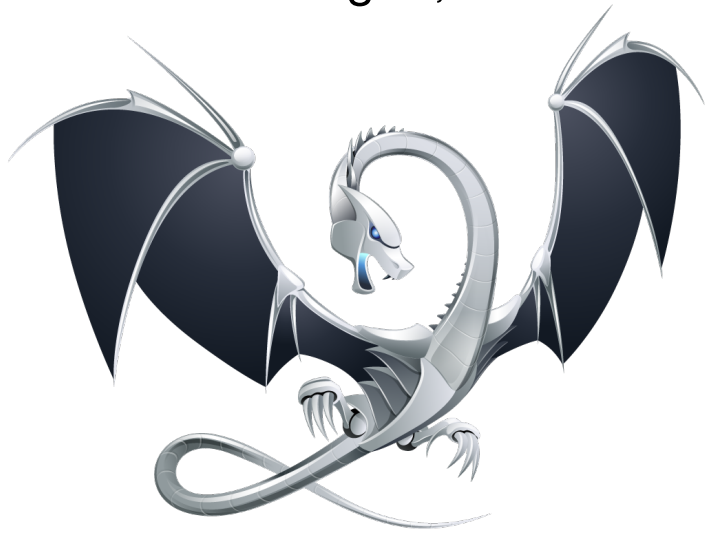
Ein Überblick

TECHNISCHE UNIVERSITÄT DARMSTADT

Fortgeschrittener Compilerbau
26. Juni 2018

Julian Oppermann, Lukas Sommer

Eingebettete Systeme und Anwendungen, Technische Universität Darmstadt



Agenda

TECHNISCHE UNIVERSITÄT DARMSTADT

- Was ist LLVM überhaupt genau?
- Entstehungsgeschichte
- Grober Überblick für die Tools
- C-Frontend
- **Einführung in die Zwischendarstellung**
- Fallstudie: Wie programmiert man eine Optimierung?
- Ein paar interessante Eigenschaften der Codeerzeugung

$\rightarrow$ Exkurs in die echte Welt

Wo wird LLVM verwendet?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Grundlage für die Apple-Entwicklungswerkzeuge
  - Wenn Sie einen Mac oder ein iOS-Gerät dabei haben, läuft darauf Code, der mit LLVM übersetzt wurde.
- WebKit nutzt einen LLVM-basierten Just-in-Time-Compiler für JavaScript
- NVIDIA nutzt LLVM als Basis für den CUDA-Compiler
- Erste Wahl in der Forschung
  - Lattner, Chris, and Vikram Adve. *LLVM: A compilation framework for lifelong program analysis & transformation.* CGO 2004.
    ... wurde bislang über 3500x zitiert (lt. Google Scholar))

Was ist LLVM (nicht)?

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM ist **kein** Compiler!
- Die LLVM-IR ist eine Zwischendarstellung, auf die man eine Vielzahl von Quellsprachen abbilden kann.
- “The LLVM Compiler Infrastructure Project” koordiniert die Entwicklung der IR und vieler weiterer Unterprojekte.
- Typischer Einsatz als Compiler:
  `clang` liest Quelltext und *benutzt* die LLVM-Bibliotheken zum Aufbau der IR, Optimierung und Codeerzeugung.

LLVM-Projekt
Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

LLVM Core	LLVM-IR, Analysen, Optimierungen, Codeerzeugung
clang	C/C++/Objective-C Frontend
dragonegg	Schnittstelle zu den Frontends der GCC
LLDB	Debugger
libc++	C++-Standardbibliothek
compiler-rt	Laufzeitumgebung für Architekturen, denen bestimmte Instruktionen fehlen
vmkit	LLVM-basierte virtuelle Maschinen für Java und .NET
polly	Schleifentransformationen zur Verbesserung der Cachelokalität und zur automatischen Parallelisierung
	⋮

LLVM-Projekt
Geschichte

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM stand ursprünglich für **L**ow **L**evel **V**irtual **M**achine.
  - Aktuell wird nur noch das Akronym verwendet, da das Projekt mittlerweile viel umfassender geworden ist.
- Begonnen Dezember 2000 von Chris Lattner und Vikram Adve an der University of Illinois.
- Heute ein erfolgreiches Open Source-Projekt unter BSD-kompatibler Lizenz.
- Gewinner des ACM Software System Award 2012.
  - In der Gesellschaft von Eclipse (2011), Java (2002), Apache (1999), WWW (1995), TCP/IP (1991), TeX (1986), UNIX (1983), ...
- Aktuelle Version: LLVM 6.0 (März 2018)

LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

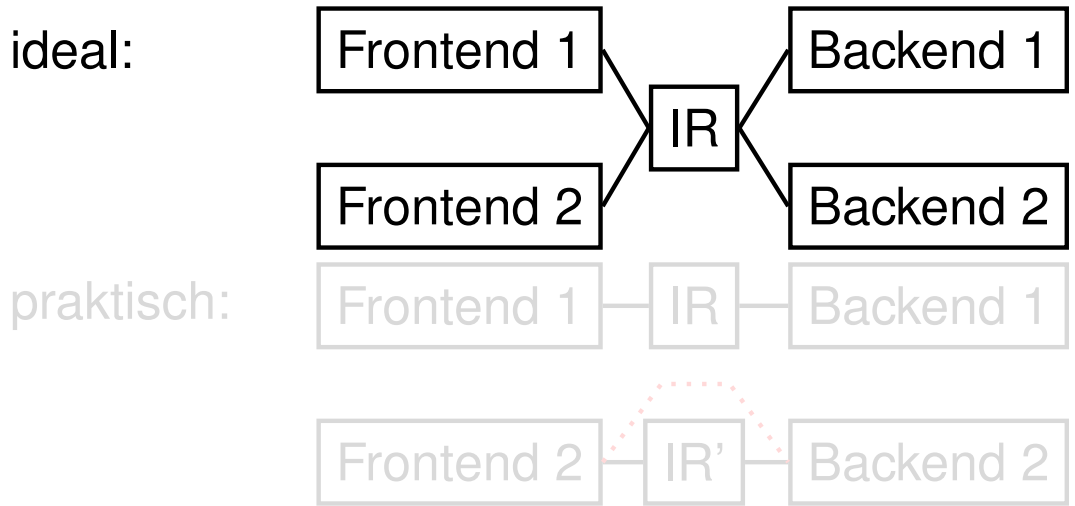
- LLVM-IR ist eigenständige Sprache, enthält alle Programminformationen.
- Ermöglicht echte Entkopplung von Frontend, Optimierungen und Codeerzeugung.
  - Im Gegensatz zu allen anderen Sprachimplementierungen zu Beginn der Entwicklung!

LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM-IR ist eigenständige Sprache, enthält alle Programminformationen.
- Ermöglicht echte Entkopplung von Frontend, Optimierungen und Codeerzeugung.
  - Im Gegensatz zu allen anderen Sprachimplementierungen zu Beginn der Entwicklung!

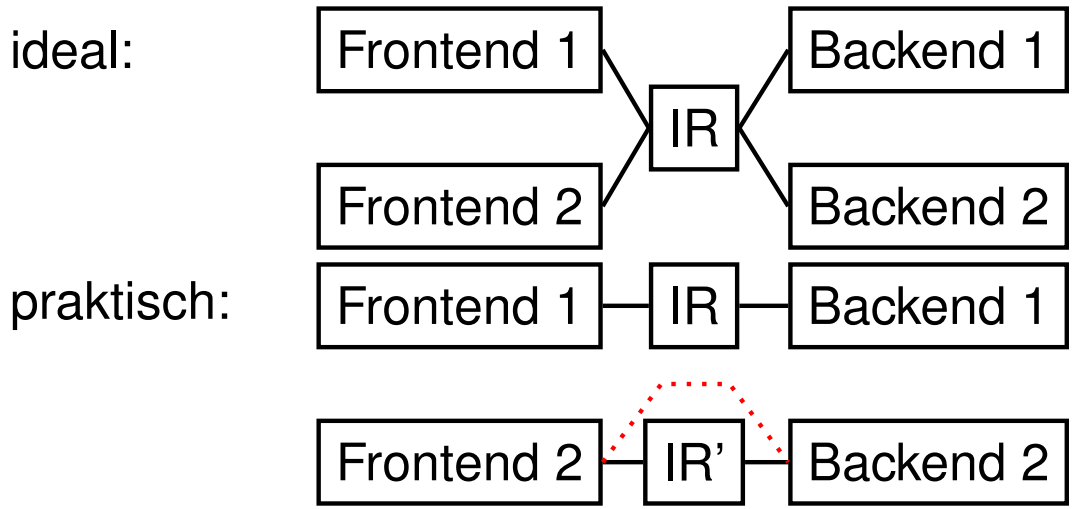


LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM-IR ist eigenständige Sprache, enthält alle Programminformationen.
- Ermöglicht echte Entkopplung von Frontend, Optimierungen und Codeerzeugung.
  - Im Gegensatz zu allen anderen Sprachimplementierungen zu Beginn der Entwicklung!

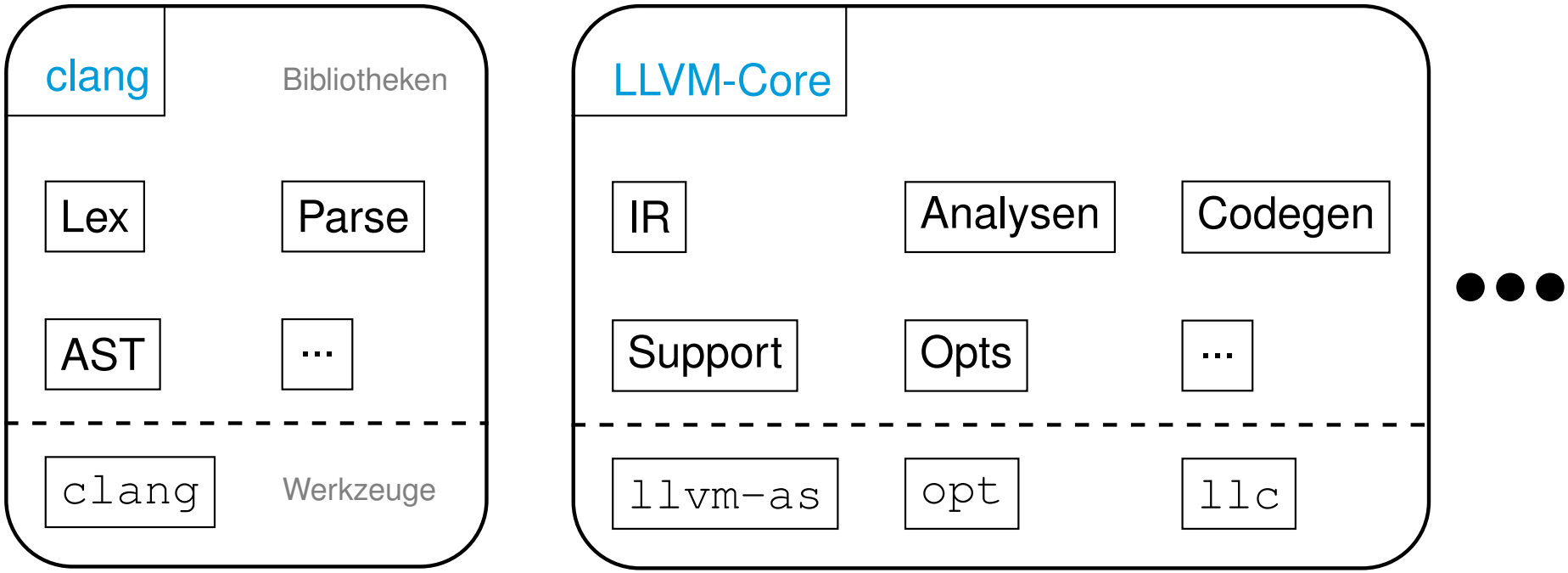


LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Modulare Architektur, bestehend aus wiederverwendbaren Bibliotheken



LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Modulare Architektur, bestehend aus wiederverwendbaren Bibliotheken

```
[julian@Julians-MBP] ~/Src/llvm-lecture/llvm/build/lib > ls *.dylib
libLLVMAggressiveInstCombine.dylib      libLLVMLinker.dylib                 libLLVMTarget.dylib                 libclangEdit.dylib
libLLVMAnalysis.dylib                   libLLVMMC.dylib                     libLLVMTransformUtils.dylib         libclangFormat.dylib
libLLVMAsmParser.dylib                  libLLVMMCDisassembler.dylib         libLLVMVectorize.dylib              libclangFrontend.dylib
libLLVMAsmPrinter.dylib                 libLLVMMCParser.dylib               libLLVMX86AsmParser.dylib           libclangFrontendTool.dylib
libLLVMBinaryFormat.dylib               libLLVMObjCARCOpts.dylib            libLLVMX86AsmPrinter.dylib          libclangIndex.dylib
libLLVMBitReader.dylib                  libLLVMObject.dylib                 libLLVMX86CodeGen.dylib             libclangLex.dylib
libLLVMBitWriter.dylib                  libLLVMOption.dylib                 libLLVMX86Desc.dylib                libclangParse.dylib
libLLVMCodeGen.dylib                    libLLVMPasses.dylib                 libLLVMX86Disassembler.dylib        libclangRewrite.dylib
libLLVMCore.dylib                       libLLVMProfileData.dylib            libLLVMX86Info.dylib                libclangRewriteFrontend.dylib
libLLVMCoroutines.dylib                 libLLVMRISCVAsmParser.dylib         libLLVMX86Utils.dylib               libclangSema.dylib
libLLVMCoverage.dylib                   libLLVMRISCVAsmPrinter.dylib        libLLVMipo.dylib                    libclangSerialization.dylib
libLLVMDebugInfoCodeView.dylib          libLLVMRISCVCodeGen.dylib           libclangARCMigrate.dylib            libclangStaticAnalyzerCheckers.dylib
libLLVMDebugInfoMSF.dylib               libLLVMRISCVDesc.dylib              libclangAST.dylib                   libclangStaticAnalyzerCore.dylib
libLLVMDemangle.dylib                   libLLVMRISCVDisassembler.dylib      libclangASTMatchers.dylib           libclangStaticAnalyzerFrontend.dylib
libLLVMGlobalISel.dylib                 libLLVMRISCVInfo.dylib              libclangAnalysis.dylib              libclangToolingCore.dylib
libLLVMIRReader.dylib                   libLLVMScalarOpts.dylib             libclangBasic.dylib                 libclangToolingInclusions.dylib
libLLVMInstCombine.dylib                libLLVMSelectionDAG.dylib           libclangCodeGen.dylib
libLLVMInstrumentation.dylib            libLLVMSupport.dylib                libclangCrossTU.dylib
libLLVMLTO.dylib                        libLLVMTableGen.dylib               libclangDriver.dylib
```

LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Es gibt drei **äquivalente** Darstellungsformen:
  - textuell (Assembler-Format, siehe Beispiel) `prog.ll`
  - binär (Bitcode-Format) `prog.bc`
  - im Speicher (C++-Objekte)
- Jede Darstellungsform enthält stets alle Details des Programms
  $\rightarrow$ klare Schnittstelle für Analysen und Transformationen.
- Für alle Analysen und Transformationen wird ausschließlich diese IR verwendet.

LLVM-Projekt

Was macht LLVM besonders?

TECHNISCHE UNIVERSITÄT DARMSTADT

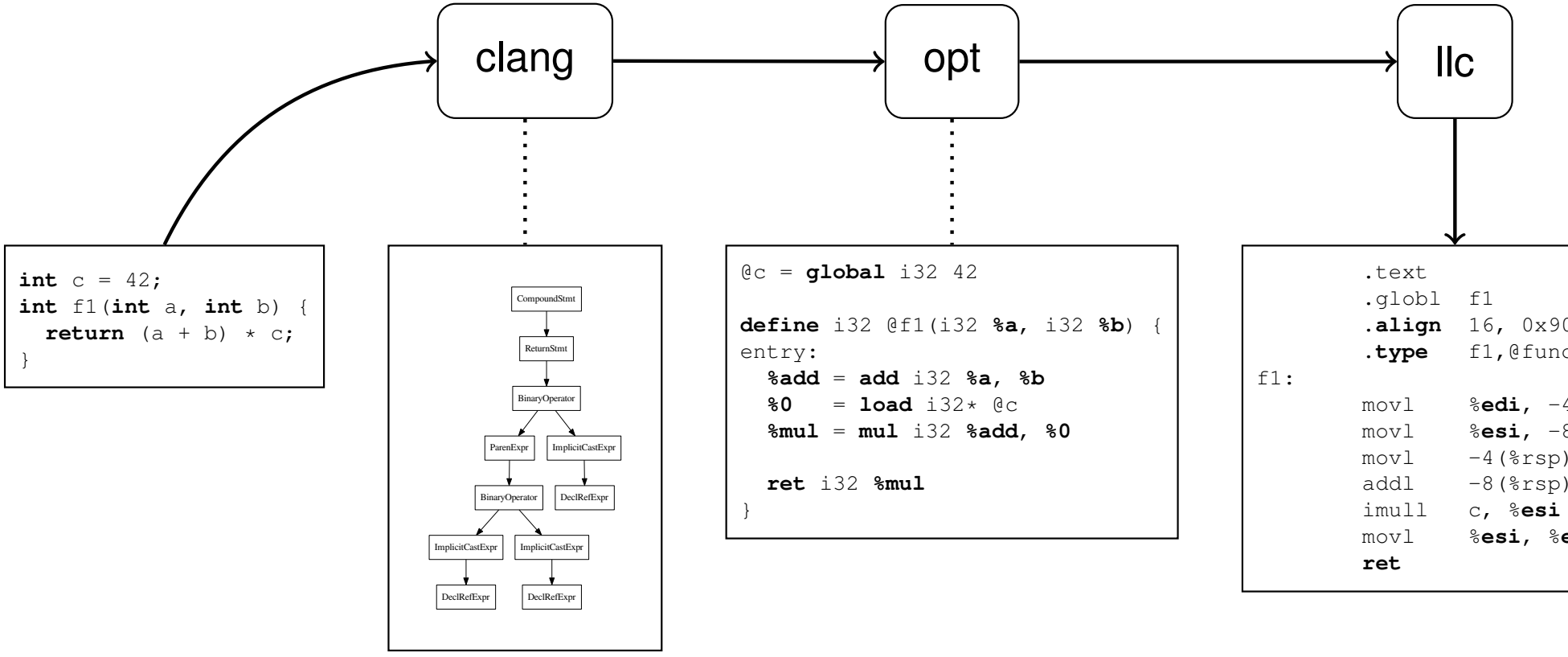
- clang
  - Schneller als gcc (compile time)
  - Bessere Fehlermeldungen

```
$ gcc-4.2 -fsyntax-only t.c
t.c:7: error: invalid operands to binary + (have 'int' and 'struct A')
$ clang -fsyntax-only t.c
t.c:7:39: error: invalid operands to binary expression ('int' and 'struct A')
  return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
                      ~~~~~~~~~~~~~~ ^ ~~~~~
```

- LLVM Core
  - Aggressive skalare Optimierungen
  - Link-time optimization
  - “easily hackable”

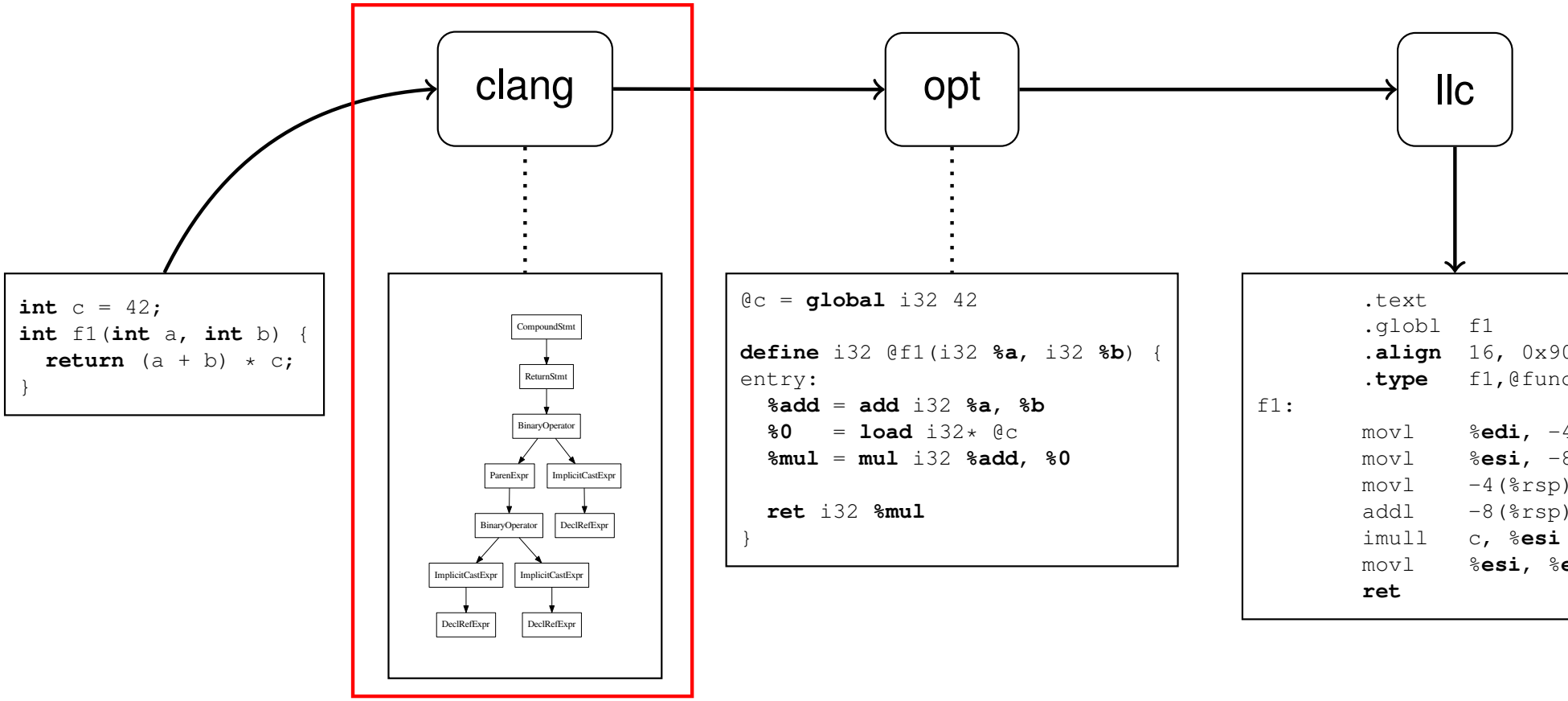
Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT



Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT



clang
Parser

TECHNISCHE UNIVERSITÄT DARMSTADT

- Benutzt einen handgeschriebenen Parser nach dem Prinzip des rekursiven Abstiegs.
- Beispiel: if-Statement (stark vereinfacht)

```
StmtResult Parser::ParseIfStatement(SourceLocation *TrailingElseLoc) {
  SourceLocation IfLoc = ConsumeToken();  // eat the 'if'.

  if (Tok.isNot(tok::l_paren)) return StmtError();

  if (ParseParenExprOrCondition(CondExp, CondVar, IfLoc, true)) return StmtError();

  StmtResult ThenStmt(ParseStatement(&InnerStatementTrailingElseLoc));

  StmtResult ElseStmt;
  if (Tok.is(tok::kw_else))
    ElseStmt = ParseStatement();

  return Actions.ActOnIfStmt(...);
}
```

clang
AST

TECHNISCHE UNIVERSITÄT DARMSTADT

- Klassenhierarchien für Deklarationen (`Decl`), Anweisungen (`Stmt`) und Typen (`Type`).
  - Ausdrücke (`Expr`) sind Unterklassen von `Stmt`.
- Wurzelknoten ist `TranslationUnitDecl`.
- Keine gemeinsame Oberklasse, jeder Knotentyp spezifiziert seine eigenen Zugriffsmethoden:

```
class IfStmt : public Stmt {
  ...
  Expr *getCond() { return reinterpret_cast<Expr*>(SubExprs[CO
  Stmt *getThen() { return SubExprs[THEN]; }
  Stmt *getElse() { return SubExprs[ELSE]; }
  ...
}
```

- Traversierung mittels RecursiveASTVisitor (“Makromonster”).

clang
AST (Beispiel)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int
addabs(int a,
       int b)
{
  int x;
  if (a*b >= 0)
    x = a+b;
  else
    x = a-b;
  return x;
}
```

```
TranslationUnitDecl 0x5ff6ba0 <<invalid sloc>>
`-FunctionDecl 0x5ff75d0 <../llvm-vortrag/cfg.c:1:1, line:8:1> addabs 'int (int, int)'
  |-ParmVarDecl 0x5ff7490 <line:1:12, col:16> a 'int'
  |-ParmVarDecl 0x5ff7500 <col:19, col:23> b 'int'
  `-CompoundStmt 0x6023f10 <col:26, line:8:1>
    |-DeclStmt 0x5ff76e8 <line:2:2, col:7>
    | `-VarDecl 0x5ff7690 <col:2, col:6> x 'int'
    |-IfStmt 0x6023e80 <line:3:2, line:6:9>
    | |-<<<NULL>>>
    | |-BinaryOperator 0x5ff77c8 <line:3:6, col:13> 'int' '>='
    | | |-BinaryOperator 0x5ff7780 <col:6, col:8> 'int' '*'
    | | | |-DeclRefExpr 0x5ff7700 <col:6> 'int' lvalue ParmVar 0x5ff7490 'a' 'int'
    | | | `-DeclRefExpr 0x5ff7728 <col:8> 'int' lvalue ParmVar 0x5ff7500 'b' 'int'
    | | `-IntegerLiteral 0x5ff77a8 <col:13> 'int' 0
    | |-BinaryOperator 0x6023d60 <line:4:3, col:9> 'int' '='
    | | |-DeclRefExpr 0x5ff77f0 <col:3> 'int' lvalue Var 0x5ff7690 'x' 'int'
    | | `-BinaryOperator 0x5ff7898 <col:7, col:9> 'int' '+'
    | |   |-DeclRefExpr 0x5ff7818 <col:7> 'int' lvalue ParmVar 0x5ff7490 'a' 'int'
    | |   `-DeclRefExpr 0x5ff7840 <col:9> 'int' lvalue ParmVar 0x5ff7500 'b' 'int'
    | `-BinaryOperator 0x6023e58 <line:6:3, col:9> 'int' '='
    |   |-DeclRefExpr 0x6023d88 <col:3> 'int' lvalue Var 0x5ff7690 'x' 'int'
    |   `-BinaryOperator 0x6023e30 <col:7, col:9> 'int' '-'
    |     |-DeclRefExpr 0x6023db0 <col:7> 'int' lvalue ParmVar 0x5ff7490 'a' 'int'
    |     `-DeclRefExpr 0x6023dd8 <col:9> 'int' lvalue ParmVar 0x5ff7500 'b' 'int'
    `-ReturnStmt 0x6023ef0 <line:7:2, col:9>
      `-DeclRefExpr 0x6023eb0 <col:9> 'int' lvalue Var 0x5ff7690 'x' 'int'
```

clang
Codeerzeugung (LLVM-IR)

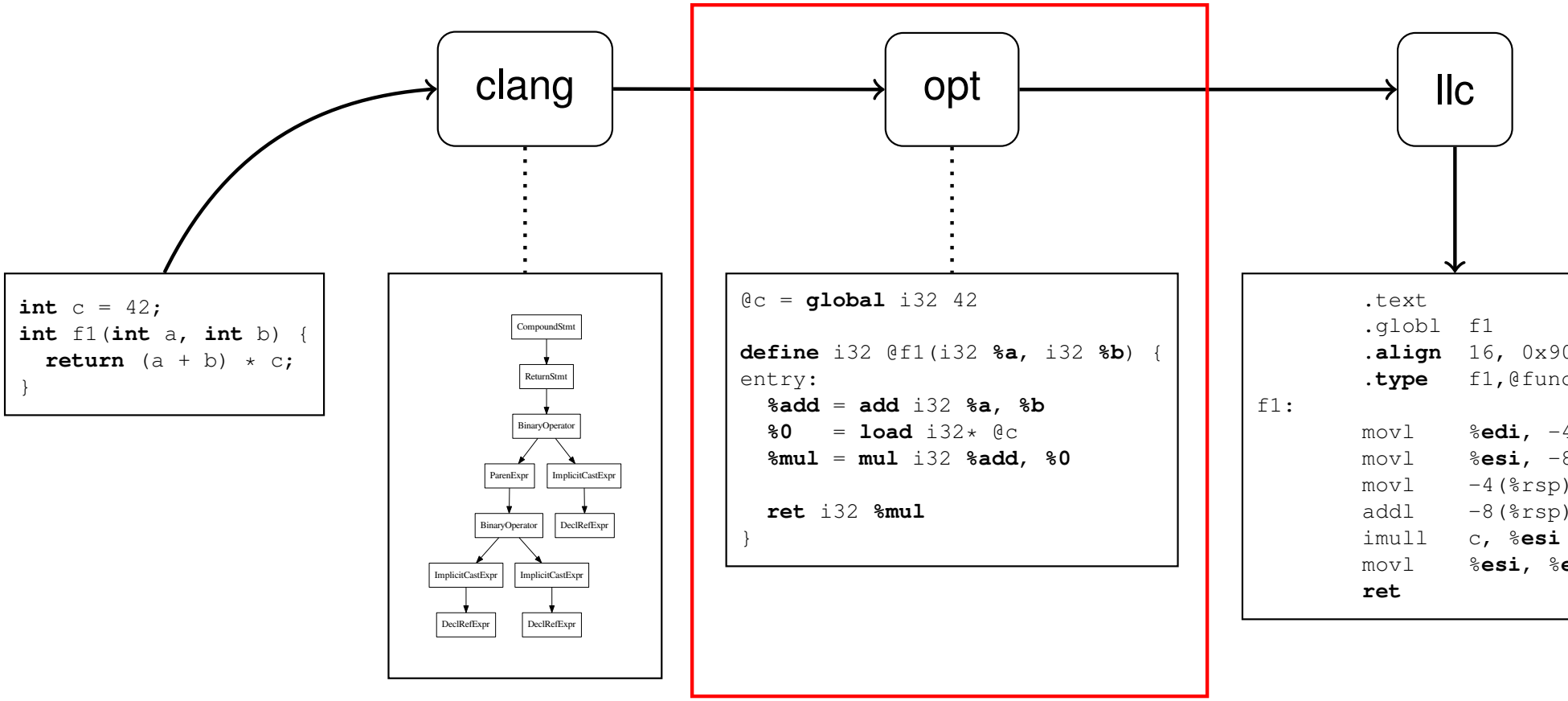
TECHNISCHE UNIVERSITÄT DARMSTADT

- (Gekürzte) LLVM-IR-Generierung für ein If-Statement

```
void CodeGenFunction::EmitIfStmt(const IfStmt &S) {
  llvm::BasicBlock *ThenBlock = createBasicBlock("if.then");
  llvm::BasicBlock *ContBlock = createBasicBlock("if.end");
  llvm::BasicBlock *ElseBlock = ContBlock;
  if (S.getElse())
    ElseBlock = createBasicBlock("if.else");
  EmitBranchOnBoolExpr(S.getCond(), ThenBlock, ElseBlock);

  EmitBlock(ThenBlock);
  EmitStmt(S.getThen());
  EmitBranch(ContBlock);

  if (const Stmt *Else = S.getElse()) {
    EmitBlock(ElseBlock);
    EmitStmt(Else);
    EmitBranch(ContBlock);
  }

  EmitBlock(ContBlock, true);
}
```

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT



LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

```
@c = global i32 42

define i32 @f1(i32 %a, i32 %b) {
entry:
  %add = add i32 %a, %b
  %0   = load i32* @c
  %mul = mul i32 %add, %0

  ret i32 %mul
}
```

LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

```
@c = global i32 42
```

Globale Variable

```
define i32 @f1(i32 %a, i32 %b) {
entry:
  %add = add i32 %a, %b
  %0   = load i32* @c
  %mul = mul i32 %add, %0

  ret i32 %mul
}
```

Funktionsdefinition

LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

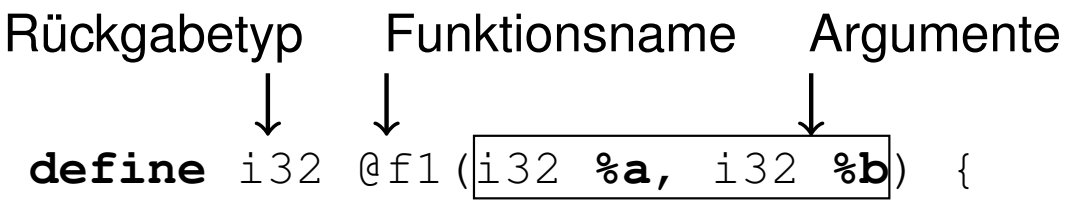
```
@c = global i32 42
```

↑ Name ↑ Typ ↑ Initialer Wert

LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```



LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

```
define i32 @f1(i32 %a, i32 %b) {
entry:
  %add = add i32 %a, %b
```

← Operanden

Ergebnistyp

Opcode

Zielregister

LLVM-IR
Ein Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int c = 42;
int f1(int a, int b) { return (a + b) * c; }
```

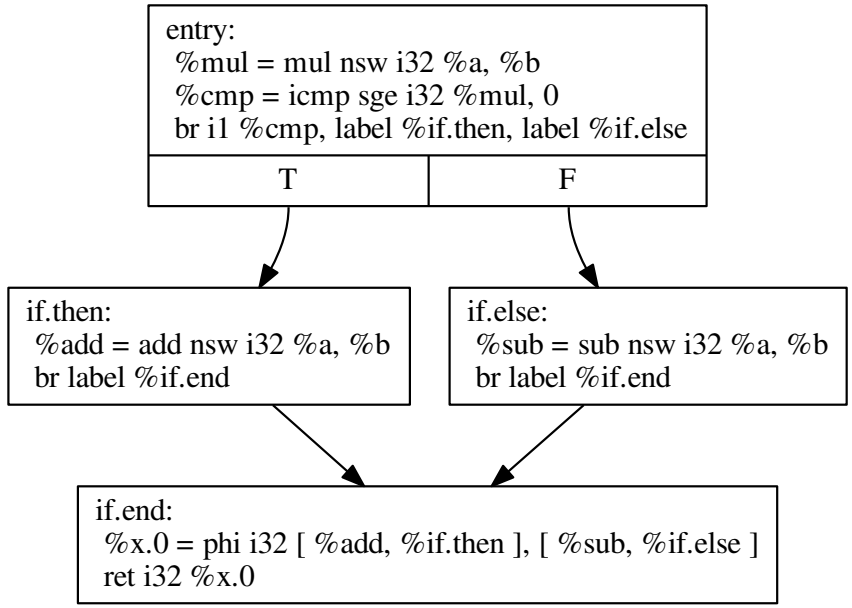
```
@c = global i32 42

define i32 @f1(i32 %a, i32 %b) {
entry:
  %add = add i32 %a, %b
  %0   = load i32* @c
  %mul = mul i32 %add, %0

  ret i32 %mul
}
```

LLVM-IR

Beispiel mit Steuerfluss

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int addabs(int a, int b) {
  int x;
  if (a*b >= 0)
    x = a+b;
  else
    x = a-b;
  return x;
}
```



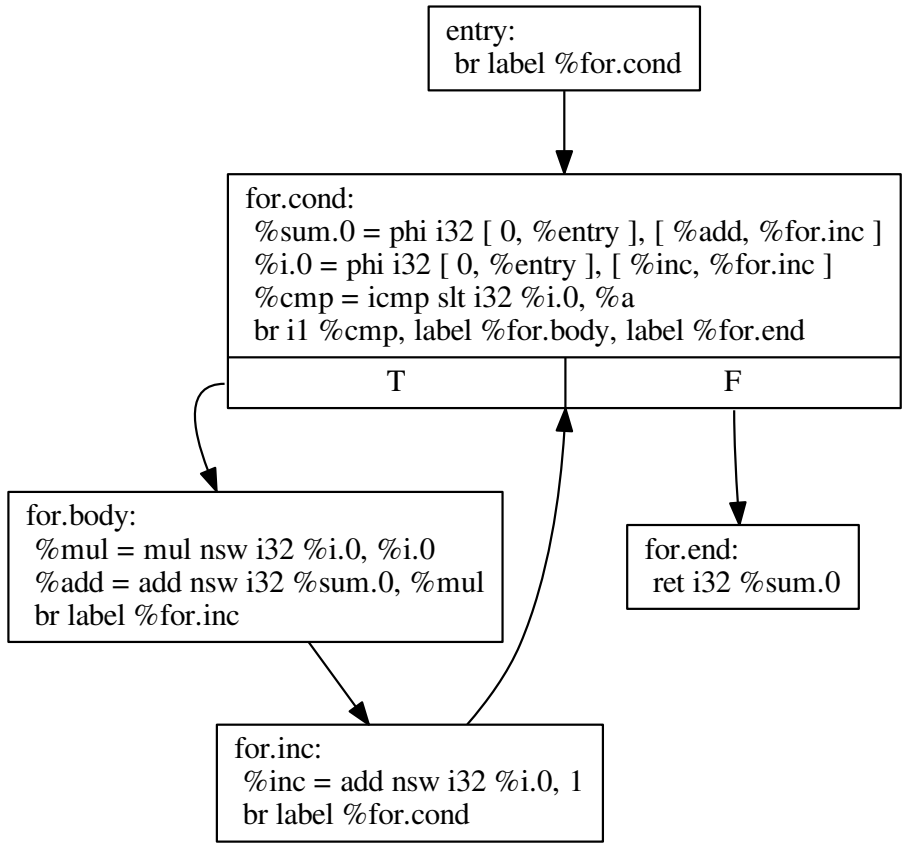
CFG for 'addabs' function

LLVM-IR

Beispiel mit Schleife

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int sumsqares(int a) {
  int sum = 0, i;
  for (i=0; i<a; i++) {
    sum += i*i;
  }
  return sum;
}
```



CFG for 'sumsqares' function

LLVM-IR

“Virtuelle Maschine”

TECHNISCHE UNIVERSITÄT DARMSTADT

- Erkenntnis: Sieht aus wie eine Assembler-Darstellung für einen RISC-Prozessor.
  - unendlich viele Register
  - Jedes Register kann nur einmal von einer eindeutig bestimmten Instruktion beschrieben werden ($\rightarrow$ **SSA-Form**).
  - typisiert
- “low level”: im Kontrast zu Java / .NET VMs
  - keine Klassen/Objekte
  - keine Vererbung
  - keine Polymorphie
  - kein Exception Handling
  - ...
  - **Aber:** alle diese Konstrukte lassen sich auf LLVM-IR abbilden!

LLVM-IR

Befehlssatz (Auszug)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerfluss: `ret` `br` `switch` `indirectbr` `invoke` `resume` `unreachable`
- Arithmetisch: `add` `fadd` `sub` `fsub` `mul` `fmul` `udiv` `sdiv` `fdiv` `urem` `srem` `frem` `shl` `lshr` `ashr` `and` `or` `xor`
- Elementzugriff: `extractelement` `insertelement` `shufflevector` `extractvalue` `insertvalue`
- Speicher und Adressierung: `alloca` `load` `store` `fence` `cmpxchg` `atomicrmw` `getelementptr`
- Konversionen: `trunc` `zext` `sext` `fptrunc` `fpext` `fptoui` `fptosi` `uitofp` `sitofp` `ptrtoint` `inttoptr` `bitcast`
- Andere: `icmp` `fcmp` `phi` `select` `call` `va_arg` `landingpad`

Auszug aus der Language Reference

TECHNISCHE UNIVERSITÄT DARMSTADT

'add' Instruction

Syntax:

```
<result> = add <ty> <op1>, <op2>          ; yields ty:result
<result> = add nuw <ty> <op1>, <op2>      ; yields ty:result
<result> = add nsw <ty> <op1>, <op2>      ; yields ty:result
<result> = add nuw nsw <ty> <op1>, <op2>  ; yields ty:result
```

Overview:

The '`add`' instruction returns the sum of its two operands.

Arguments:

The two arguments to the '`add`' instruction must be *integer* or *vector* of integer values. Both arguments must have identical types.

Semantics:

The value produced is the integer sum of the two operands.

If the sum has unsigned overflow, the result returned is the mathematical result modulo 2^n, where n is the bit width of the result.

Because LLVM integers use a two's complement representation, this instruction is appropriate for both signed and unsigned integers.

`nuw` and `nsw` stand for "No Unsigned Wrap" and "No Signed Wrap", respectively. If the `nuw` and/or `nsw` keywords are present, the result value of the `add` is a *poison value* if unsigned and/or signed overflow, respectively, occurs.

Example:

```
<result> = add i32 4, %var          ; yields i32:result = 4 + %var
```

LLVM-IR

Befehlssatz (Besonderheiten)

TECHNISCHE UNIVERSITÄT DARMSTADT

phi Die Φ-Funktion der SSA-Form

- explizite Instruktion, Tupel von Wert und Label als Argumente
- Beispiel:

```
%x = phi i32 [ %add, %then ], [ %sub, %else ]
```

select Bedingter Datenfluss

- Beispiel:

```
%sel = select i1 %cmp, i32 %val.1, i32 %val.0
```

call Funktionsaufruf

- abstrahiert Aufrufkonventionen, erhält Funktionsname und -argumente als Parameter
- Beispiel: `%y = call i32 @Get_Bits(i32 1)`

switch Switch-Instruktion

- Beispiel:

```
switch i32 %val, label %def [ i32 0, label %l0,
                              i32 1, label %l1,
                              i32 2, label %l2 ]
```

LLVM-IR
Typsystem

TECHNISCHE UNIVERSITÄT DARMSTADT

Jeder Wert hat einen Typ (unabhängig von der Quellsprache)!

- Integerwerte: `i1, i8, i16, i32, ...`
  alle Bitbreiten möglich, keine signed/unsigned-Unterscheidung
- Fließkommazahlen: `half, float, double, ...`
- Zeiger: `i64*`
- Arrays: `[10 x i32]`, `[2 x [2 x float]]`
- Strukturen: `{i32, float, i32}`
- Vektoren (SIMD): `<i8, i8, i8, i8>`
- ...

LLVM-IR

GetElementPtr (GEP)

TECHNISCHE UNIVERSITÄT DARMSTADT

- *Typsichere* Adressrechnung für Array- oder Struktur-Elemente (wichtig für Optimierungen!)
- Erhält einen Basiszeiger und eine Folge von Indizes; liefert einen Zeiger. Macht keine Speicherzugriffe!
- Achtung, entspricht semantatisch nicht 1:1 dem `[]`-Operator in C!
- Syntax:

```
<result> = getelementptr <ty>, <ty>* <ptrval>{, <ty> <idx>}*
```

GetElementPtr - Beispiele (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void gep_arr() {
    int A[10];
    A[4] = …;
}
```

```
%A   = alloca [10 x i32], align 16
%adr = getelementptr inbounds [10 x i32],
       [10 x i32]* %A, i64 0, i64 4
```

GetElementPtr - Beispiele (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void gep_arr() {
    int A[10];
    A[4] = …;
}
void get_mdim() {
    int B[10][10];
    B[3][5] = …;
}
```

```
%A   = alloca [10 x i32], align 16
%adr = getelementptr inbounds [10 x i32],
       [10 x i32]* %A, i64 0, i64 4

%B   = alloca [10 x [10 x i32]], align 16
%adr = getelementptr inbounds
       [10 x [10 x i32]], [10 x [10 x i32]]*
       %B, i64 0, i64 3, i64 5
```

GetElementPtr - Beispiele (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void gep_arr() {
    int A[10];
    A[4] = …;
}
```

```
%A   = alloca [10 x i32], align 16
%adr = getelementptr inbounds [10 x i32],
       [10 x i32]* %A, i64 0, i64 4
```

```
void get_mdim() {
    int B[10][10];
    B[3][5] = …;
}
```

```
%B   = alloca [10 x [10 x i32]], align 16
%adr = getelementptr inbounds
       [10 x [10 x i32]], [10 x [10 x i32]]*
       %B, i64 0, i64 3, i64 5
```

```
struct astruct {
  int f1;
  int f2;
};
void gep_struct() {
    struct astruct S;
    S.f2 = …;
}
```

```
%S  = alloca %struct.astruct, align 4
%f2 = getelementptr inbounds %struct.astruct,
      %struct.astruct* %S, i64 0, i32 1
```

GetElementPtr - Beispiele (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void gep_aptr(int *P) {
    P[1] = …;
}
```

```
define void @gep_aptr(i32* %P) {
entry:
  %adr = getelementptr inbounds i32,
         i32* %P, i64 1
  …
```

GetElementPtr - Beispiele (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void gep_aptr(int *P) {
    P[1] = …;
}
```

```
define void @gep_aptr(i32* %P) {
entry:
  %adr = getelementptr inbounds i32,
         i32* %P, i64 1
  …
```

```
void gep_sptr(
  struct astruct *P) {
    P[3].f1 = …;

    …
    P->f2 = …;
}
```

```
define void @gep_sptr(%struct.astruct* %P) {
entry:
  %f1 = getelementptr inbounds
        %struct.astruct,
        %struct.astruct* %P, i64 3, i32 0
  …
  %f2 = getelementptr inbounds
        %struct.astruct,
        %struct.astruct* %P, i32 0, i32 1
  …
```

GetElementPtr - Aufgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
%struct.task_t = type { i32, f32, [4 x [4 x double]] }
…
%arrayidx2 = getelementptr inbounds %struct.task_t,
             %struct.task_t* %t, i64 1, i32 2, i64 3, i64 0
```

- Welchen Typ hat `%arrayidx2`?

- Welches Offset (in Bytes) hat `%arrayidx2` zu `%t`?

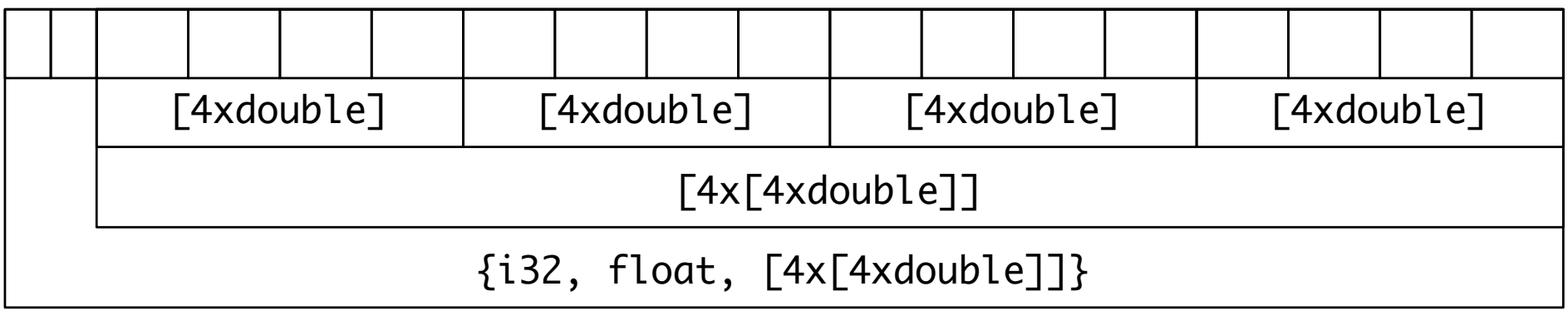
GetElementPtr - Aufgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
%struct.task_t = type { i32, f32, [4 x [4 x double]] }
…
%arrayidx2 = getelementptr inbounds %struct.task_t,
             %struct.task_t* %t, i64 1, i32 2, i64 3, i64 0
```

- Welchen Typ hat `%arrayidx2`?

- Welches Offset (in Bytes) hat `%arrayidx2` zu `%t`?



Demo-Time!

LLVM-IR
C++ Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include <iostream>
using namespace std;

class A {
int x;
public:
  virtual void foo() { cout << "A::foo" << endl; }
};

class B : public A {
public:
  virtual void foo() { cout << "B::foo" << endl; }
};

int main() {
  A *ab = new B();
  ab->foo();
  return 0;
}
```

LLVM-IR

C++ Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
; ModuleID = 'Klass.ll'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-a0:0:64-s0:64:64-f80:128:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.9.0"

%"class.std::ios_base::Init" = type { i8 }
%"class.std::basic_ostream" = type { i32 (...)**, %"class.std::basic_ios" }
%"class.std::basic_ios" = type { %"class.std::ios_base", %"class.std::basic_ostream"*, i8, i8, %"class.std::basic_streambuf"*, %"class.std::ctype"*, %"class.std::num_put"*, %"class.std::num_get"* }
%"class.std::ios_base" = type { i32 (...)**, i64, i64, i32, i32, i32, %"struct.std::ios_base::_Callback_list"*, %"struct.std::ios_base::_Words"[8 x %"struct.std::ios_base::_Words"*], i32, %"struct.std::ios_base::_Words"*, %"class.std::locale" }
%"struct.std::ios_base::_Callback_list" = type { %"struct.std::ios_base::_Callback_list"*, void (i32, %"class.std::ios_base"*, i32)*, i32, i32 }
%"struct.std::ios_base::_Words" = type { i8*, i64 }
%"class.std::locale" = type { %"class.std::locale::_Impl"* }
%"class.std::locale::_Impl" = type { i32, %"class.std::locale::facet"**, i64, %"class.std::locale::facet"**, i8** }
%"class.std::locale::facet" = type { i32 (...)**, i32 }
%"class.std::basic_streambuf" = type { i32 (...)**, i8*, i8*, i8*, i8*, i8*, i8*, i8*, %"class.std::locale" }
%"class.std::ctype" = type { %"class.std::locale::facet", i32*, i8, i32*, i32*, [64* x i8], [256 x i8], [256 x i8], i8 }
%"class.std::num_put" = type { %"class.std::locale::facet" }
%"class.std::num_get" = type { %"class.std::locale::facet" }
%class.B = type { %class.A }
%class.A = type { i32 (...)**, i32 }

@_ZStL8__ioinit = internal global %"class.std::ios_base::Init" zeroinitializer, align 1
@__dso_handle = external global i8
@_ZTV1B = linkonce_odr unnamed_addr constant [3 x i8*] [i8* null, i8* bitcast ({ i8*, i8*, i8* }* @_ZTI1B to i8*), i8* bitcast (void (%class.B*)* @_ZN1B6fooEv to i8*)]
@_ZTVN10__cxxabiv120__si_class_type_infoE = external global i8*
@_ZTS1B = linkonce_odr constant [3 x i8] c"1B\00"
@_ZTVN10__cxxabiv117__class_type_infoE = external global i8*
@_ZTS1A = linkonce_odr constant [3 x i8] c"1A\00"
@_ZTI1A = linkonce_odr unnamed_addr constant { i8*, i8* } { i8* bitcast (i8** getelementptr inbounds (i8** @_ZTVN10__cxxabiv117__class_type_infoE, i64 2) to i8*), i8* getelementptr inbounds ([3 x i8]* @_ZTS1A, i32 0, i32 0) }
@_ZTI1B = linkonce_odr unnamed_addr constant { i8*, i8*, i8* } { i8* bitcast (i8** getelementptr inbounds (i8** @_ZTVN10__cxxabiv120__si_class_type_infoE, i64 2) to i8*), i8* getelementptr inbounds ([3 x i8]* @_ZTS1B, i32 0, i32 0), i8* bitcast ({ i8*, i8* }* @_ZTI1A to i8*) }
@_ZSt4cout = external global %"class.std::basic_ostream"
@.str = private unnamed_addr constant [7 x i8] c"Hi foo\00", align 1
@_ZTV1A = linkonce_odr unnamed_addr constant [3 x i8*] [i8* null, i8* bitcast ({ i8*, i8* }* @_ZTI1A to i8*), i8* bitcast (void (%class.A*)* @_ZN1A3fooEv to i8*)]
@.str1 = private unnamed_addr constant [7 x i8] c"A: foo\0A", align 1
@llvm.global_ctors = appending global [1 x { i32, void ()* }] [{ i32, void ()* } { i32 65535, void ()* @_GLOBAL__I_a }]

define internal void @__cxx_global_var_init() section "__TEXT,__StaticInit,regular,pure_instructions" {
entry:
  call void @_ZNSt8ios_base4InitC1Ev(%"class.std::ios_base::Init"* @_ZStL8__ioinit)
  %0 = call i32 @__cxa_atexit(void (i8*)* bitcast (void (%"class.std::ios_base::Init"*)* @_ZNSt8ios_base4InitD1Ev to void (i8*)*), i8* getelementptr inbounds (%"class.std::ios_base::Init"* @_ZStL8__ioinit, i32 0, i32 0), i8* @__dso_handle) #1
  ret void
}

declare void @_ZNSt8ios_base4InitC1Ev(%"class.std::ios_base::Init"*) #0

declare void @_ZNSt8ios_base4InitD1Ev(%"class.std::ios_base::Init"*) #0

; Function Attrs: nounwind
declare i32 @__cxa_atexit(void (i8*)*, i8*, i8*) #1

; Function Attrs: uwtable
define i32 @main() #2 {
entry:
  %call = call noalias i8* @_Znwm(i64 16)
  %0 = bitcast i8* %call to %class.B*
  %1 = bitcast %class.B* %0 to i8*
  call void @llvm.memset.p0i8.i64(i8* %1, i8 0, i64 16, i32 8, i1 false)
  call void @_ZN1BC1Ev(%class.B* %0) #1
  %2 = bitcast %class.B* %0 to %class.A*
  %3 = bitcast %class.A* %2 to void (%class.A*)***
  %vtable = load void (%class.A*)**** %3
  %vfn = getelementptr inbounds void (%class.A*)** %vtable, i64 0
  %4 = load void (%class.A*)** %vfn
  call void %4(%class.A* %2)
  ret i32 0
}

declare noalias i8* @_Znwm(i64) #0

; Function Attrs: nounwind
declare void @llvm.memset.p0i8.i64(i8* nocapture, i8, i64, i32, i1) #1

; Function Attrs: inlinehint nounwind uwtable
define linkonce_odr void @_ZN1BC1Ev(%class.B* %this) unnamed_addr #3 align 2 {
entry:
  call void @_ZN1BC2Ev(%class.B* %this) #1
  ret void
}

; Function Attrs: inlinehint nounwind uwtable
define linkonce_odr void @_ZN1BC2Ev(%class.B* %this) unnamed_addr #3 align 2 {
entry:
  %0 = bitcast %class.B* %this to %class.A*
  call void @_ZN1AC2Ev(%class.A* %0) #1
  %1 = bitcast %class.B* %this to i8***
  store i8** getelementptr inbounds ([3 x i8*]* @_ZTV1B, i64 0, i64 2), i8*** %1
  ret void
}

; Function Attrs: inlinehint nounwind uwtable
define linkonce_odr void @_ZN1AC2Ev(%class.A* %this) unnamed_addr #3 align 2 {
entry:
  %0 = bitcast %class.A* %this to i8***
  store i8** getelementptr inbounds ([3 x i8*]* @_ZTV1A, i64 0, i64 2), i8*** %0
  ret void
}

; Function Attrs: uwtable
define linkonce_odr void @_ZN1B6fooEv(%class.B* %this) unnamed_addr #2 align 2 {
entry:
  %call = call %"class.std::basic_ostream"* @_ZSt4lsI61char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc(%"class.std::basic_ostream"* @_ZSt4cout, i8* getelementptr inbounds ([7 x i8]* @.str, i32 0, i32 0))
  %call2 = call %"class.std::basic_ostream"* @_ZNSolsEPFRSoS_E(%"class.std::basic_ostream"* %call, %"class.std::basic_ostream"* (%"class.std::basic_ostream"*)* @_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_)
  ret void
}

declare %"class.std::basic_ostream"* @_ZSt4lsI61char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc(%"class.std::basic_ostream"*, i8*) #0

declare %"class.std::basic_ostream"* @_ZNSolsEPFRSoS_E(%"class.std::basic_ostream"*, %"class.std::basic_ostream"* (%"class.std::basic_ostream"*)*) #0

declare %"class.std::basic_ostream"* @_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(%"class.std::basic_ostream"*) #0

; Function Attrs: uwtable
define linkonce_odr void @_ZN1A3fooEv(%class.A* %this) unnamed_addr #2 align 2 {
entry:
  %call = call %"class.std::basic_ostream"* @_ZSt4lsI61char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc(%"class.std::basic_ostream"* @_ZSt4cout, i8* getelementptr inbounds ([7 x i8]* @.str1, i32 0, i32 0))
  %call2 = call %"class.std::basic_ostream"* @_ZNSolsEPFRSoS_E(%"class.std::basic_ostream"* %call, %"class.std::basic_ostream"* (%"class.std::basic_ostream"*)* @_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_)
  ret void
}

define internal void @_GLOBAL__I_a() section "__TEXT,__StaticInit,regular,pure_instructions" {
entry:
  call void @__cxx_global_var_init()
  ret void
}

attributes #0 = { "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"="true" "no-info-fp-math"="false" "no-nans-fp-math"="false" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { nounwind }
attributes #2 = { uwtable "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"="true" "no-info-fp-math"="false" "no-nans-fp-math"="false" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #3 = { inlinehint nounwind uwtable "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"="true" "no-info-fp-math"="false" "no-nans-fp-math"="false" "unsafe-fp-math"="false" "use-soft-float"="false" }
```

LLVM-IR
C++ Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
%class.B = type { %class.A }
%class.A = type { i32 (...)**, i32 }
```

```
@_ZTV1B = linkonce_odr unnamed_addr constant [3 x i8*] [i8* null,
        i8* bitcast ({ i8*, i8*, i8* }* @_ZTI1B to i8*),
        i8* bitcast (void (%class.B*)* @_ZN1B3fooEv to i8*)]
```

```
@_ZTV1A = linkonce_odr unnamed_addr constant [3 x i8*] [i8* null,
        i8* bitcast ({ i8*, i8* }* @_ZTI1A to i8*),
        i8* bitcast (void (%class.A*)* @_ZN1A3fooEv to i8*)]
```

LLVM-IR

C++ Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
define i32 @main() #2 {
entry:
  %call = call noalias i8* @_Znwm(i64 16)
  %0 = bitcast i8* %call to %class.B*
  %1 = bitcast %class.B* %0 to i8*
  call void @llvm.memset.p0i8.i64(i8* %1, i8 0, i64 16, i32 8, i1 false)
  call void @_ZN1BC1Ev(%class.B* %0) #1
  %2 = bitcast %class.B* %0 to %class.A*
  %3 = bitcast %class.A* %2 to void (%class.A*)***
  %vtable = load void (%class.A*)*** %3
  %vfn = getelementptr inbounds void (%class.A*)** %vtable, i64 0
  %4 = load void (%class.A*)** %vfn
  call void %4(%class.A* %2)
  ret i32 0
}
```

Unions

TECHNISCHE UNIVERSITÄT DARMSTADT

```c
#include <stdio.h>

typedef union {
      long long ll;
      long      l;
      int       i;
      short     s;
      char      c;
} ints_u;

int main() {
      ints_u U;
      U.ll = 0xDEADBEEFCAFEBABELL;

      printf ("U.s = %d\n", U.s);
      return 0;
}
```

- Typ des größten Elements
- Frontend erzeugt bitcasts für Zugriff auf andere Elemente

```
; ModuleID = 'union.ll'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-
i32:32:32-i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-
a0:0:64-s0:64:64-f80:128:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.9.0"

%union.ints_u = type { i64 }

@.str = private unnamed_addr constant [10 x i8] c"U.s = %d\0A
\00", align 1

; Function Attrs: nounwind ssp uwtable
define i32 @main() #0 {
  %U = alloca %union.ints_u, align 8
  %1 = bitcast %union.ints_u* %U to i64*
  store i64 -2401053089206453570, i64* %1, align 8
  %2 = bitcast %union.ints_u* %U to i16*
  %3 = load i16* %2, align 2
  %4 = sext i16 %3 to i32
  %5 = call i32 (i8*, ...)* @printf(i8*
getelementptr inbounds ([10 x i8]* @.str, i32 0, i32 0), i32 %4)
  ret i32 0
}
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Parameterliste

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Neues Sprachfeature in C++11

```
$ clang++ --std=c++11 lambda.cpp
```

```
#include <iostream>

using namespace std;

int main() {
      int x;
      cin >> x;

      auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
      func(42);

      auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
      func2("nice");

      return 0;
}
```

Parameterliste

Rückgabewert (optional)

Capture specification

[]	„Nichts“
[&]	„By-Reference“
[=]	„By-Value“

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lowering passiert im Frontend

- Für (fast) jede λ-Funktion wird eine neue Klasse erzeugt:
  - Der ()-Operator wird entsprechend (Parameter, Rückgabewert, Rumpf) überschrieben.
  - Ein Konstruktor zur Übergabe der gecapture'ten Variablen wird erzeugt.
  - Ausnahme: λs, die keine Variablen capturen, werden zu Funktionen.

Quelle (und gute Erklärung): http://www.cprogramming.com/c++11/c++11-lambda-closures.html

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8            ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1         ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

Adresse von x capturen
(statt Konstruktor-Aufruf)

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

Darstellung in LLVM (Ausschnitt aus ~ 1000 Zeilen LLVM Assembler…)

```
auto func = [&] (int i) -> void { cout << "Hello, No. " << i << ", you typed " << x << endl; };
func(42);
```

Closure-Objekt

```
define i32 @main() #0 {
  %x = alloca i32, align 4
  %func = alloca %class.anon, align 8          ; %class.anon = type { i32* }
  %func2 = alloca %class.anon.0, align 1       ; %class.anon.0 = type { i8 }
  …
  %3 = getelementptr inbounds %class.anon* %func, i32 0, i32 0
  store i32* %x, i32** %3, align 8
  call void @"_ZZ4mainENK3$_0clEi"(%class.anon* %func, i32 42)
```

Adresse von x capturen
(statt Konstruktor-Aufruf)

Die Methode aufrufen

λ-Funktionen (aka Closures)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
auto func2 = [] (const string &s) -> void { cout << "Have a " << s << " day!" << endl; };
func2("nice");
```

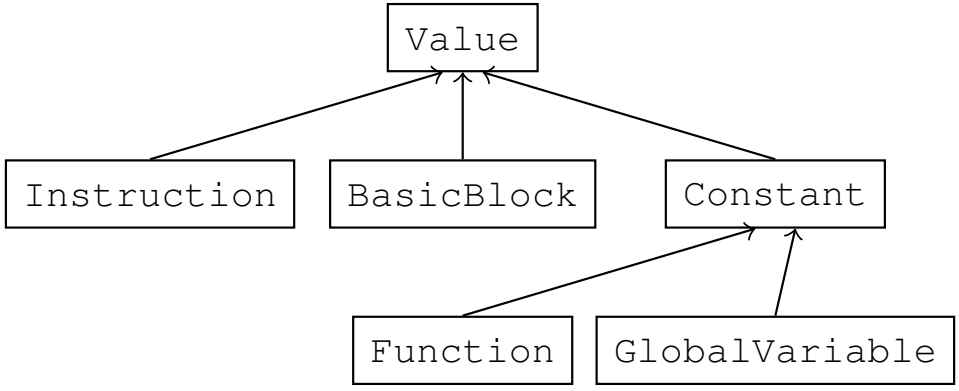
```
invoke void
  @„_ZZ4mainENK3$_1clERKNSt3__112basic_stringIcNS0_11char_traitsIcEENS0_9allocatorIcEEEE"(
  %class.anon.0* %func2, %"class.std::__1::basic_string"* %1)
  to label %10 unwind label %11
```

Davor: String-Initialisierung
Danach: Exception Handling

LLVM-IR
API

TECHNISCHE UNIVERSITÄT DARMSTADT

- Oberklasse für fast alle IR-Elemente: `Value` modelliert (SSA-)Werte.
  - Jeder Wert hat einen `Type`.
- (Vereinfachte) Klassenhierarchie:



- `Instruction`s speichern ihre Operanden als Zeiger zu anderen `Value`-Objekten.
- Beispiel: Konstruktor von `BranchInst`:
  `BranchInst(BasicBlock *IfTrue, BasicBlock *IfFalse, Value *Cond)`

„Navigation“ in der IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Argumente
  - `Instruction *I = … ; I->getOperand(1);`
  - `BranchInst *Br = …; Br->getCondition();`
  - `CallInst *Ci = …; Ci->getArgOperand(4); Ci->getCalledFunction();`

- CFG
  - `Function *F = …; F->getEntryBlock();`
  - `BasicBlock *BB = …;`
    - `for (pred_iterator PI = pred_begin(BB), PE = pred_end(BB); PI != PE; PI++) { …`
    - `for (succ_iterator SI = succ_begin(BB), SE = succ_end(BB); SI != SE; SI++) { …`
    - `TerminatorInst *TI = BB->getTerminator();`
    - `if (BranchInst *BI = dyn_cast<BranchInst>(TI))`
      `BI->getSuccessor(1) // „false“`

„Navigation“ in der IR (alt)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Module *M = …;
for (Module::iterator MI = M->begin(), ME = M->end();
     MI != ME; MI++) {
    Function *F = MI;
    for (Function::iterator FI = F->begin(), FE = F->end();
         FI != FE; FI++) {
        BasicBlock *BB = FI;
        for (BasicBlock::iterator BBI = BB->begin(), BBE = BB->end();
             BBI != BBE; BBI++) {
            Instruction *I = BBI;

            assert (I->getParent() == BB);
            assert (I->getParent()->getParent() == F);
            assert (I->getParent()->getParent()->getParent() == M);
         }
    }
}
```

„Navigation“ in der IR (neu)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Module &M = …;

for (auto &F : M) {
    for (auto &BB : F) {
        for (auto &I : BB) {
            assert (I.getParent() == &BB);
            assert (I.getParent()->getParent() == &F);
            assert (I.getParent()->getParent()->getParent() == &M);
         }
    }
}
```

„Navigation“ in der IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jeder `Value` hat eine Liste von `Usern`

```
Value *V = …;
for (Use &use : V->uses()) {
  User *U = use.getUser();
```

- `User` sind alle IR-Elemente, die andere Werte referenzieren können
  - Hauptsächlich Instructions und Konstanten
- `Use` repräsentiert die Kanten zwischen Werten und ihren Verwendern

- Beispiel: Alle (direkten) Aufrufstellen einer Funktion F

```
for (User &U : F->users()) {
  if (CallInst *CI = dyn_cast<CallInst>(U)) {
    …
```

InstVisitor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class MyVisitor : public InstVisitor<MyVisitor> {
public:
    void visitAdd(BinaryOperator &I) { … }

    void visitStoreInst(StoreInst &I) { … }
    void visitTerminatorInst(TerminatorInst &I) { … }
};

// somewhere
Function &F = …;
MyVisitor V; V.visit(F);
```

- Überschreibbare Methoden für Opcodes, spezifische Instruktionen, und Klassen von Instruktionen.
- Makromagie, keine `accept(…)`-Methode in Instruction-Hierarchie.

DominatorTree

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include <llvm/IR/Dominators.h>

Function &F = …;

DominatorTree DT(F);
if (DT.dominates(BB1, BB2)) …

DomTreeNode *DTN1 = DT[BB3], *DTN2;
DTN2 = DTN1->getIDom();

DT.findNearestCommonDominator(BB1, BB2);
```

LoopInfo

TECHNISCHE UNIVERSITÄT DARMSTADT

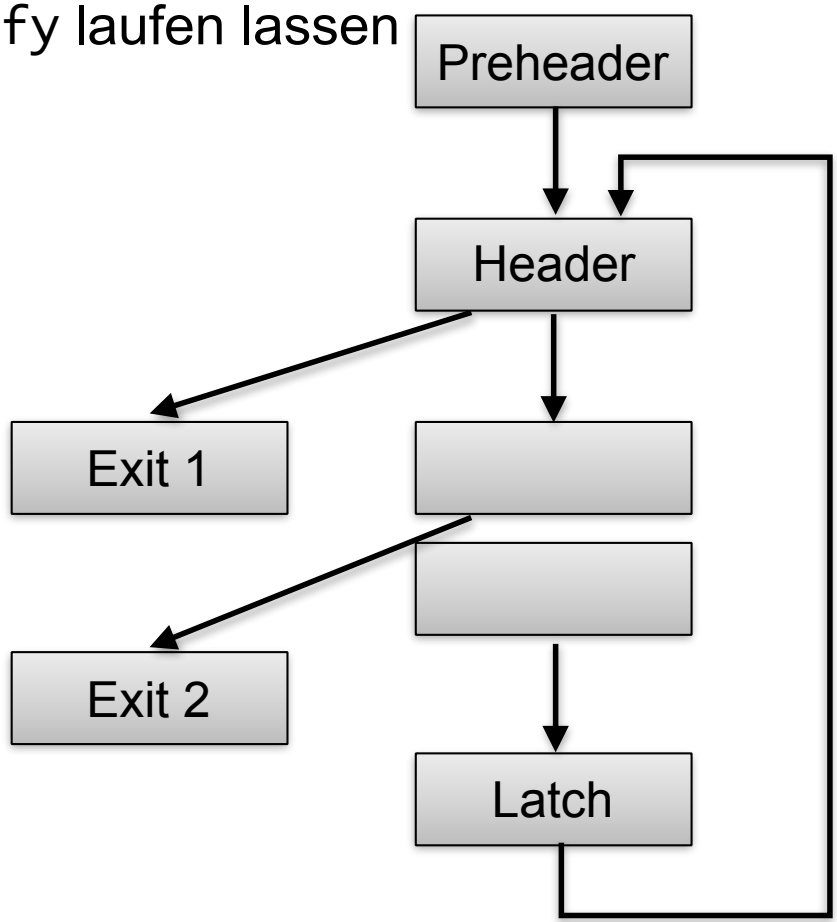
- Liefert natürliche Schleifen (= ein Eintrittspunkt)
- Sehr sinnvoll: Vorher `-loop-simplify` laufen lassen
- Verwendung

```
DominatorTree DT(F);
LoopInfo LI(DT);
for (LoopInfo::iterator …
  // Toplevel Loops

Loop *L = LI.getLoopFor(BB);
unsigned d = LI.getLoopDepth(BB);
```

- Verwendung der Loop-Objekte:

  Zugriff auf ausgezeichnete Blöcke

  (falls eindeutig)



Optimierungen
Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM-IR enthält genug Informationen, um auch “high-level” Analysen und Transformationen durchzuführen.
- Gekapselt als Pässe.
- Auszug aus der Liste der mitgelieferten Pässe:
  - Analysen: (Post-)Dominatorbaum, natürliche Schleifen, Aliasanalyse(n), . . .
  - Transformationen: Dead Code Elimination, Reassociation, Loop Invariant Code Motion, Global Value Numbering, . . .
- Abhängigkeiten zwischen Pässen werden automatisch aufgelöst.
- Man kann sogar Transformationen einzeln auf ein Programm anwenden:

```
$ opt -S -reassociate -o prog_opt.ll prog.ll
$ opt -S -licm -o prog_opt2.ll prog_opt.ll
```

Optimierungen
Konstantenpropagation

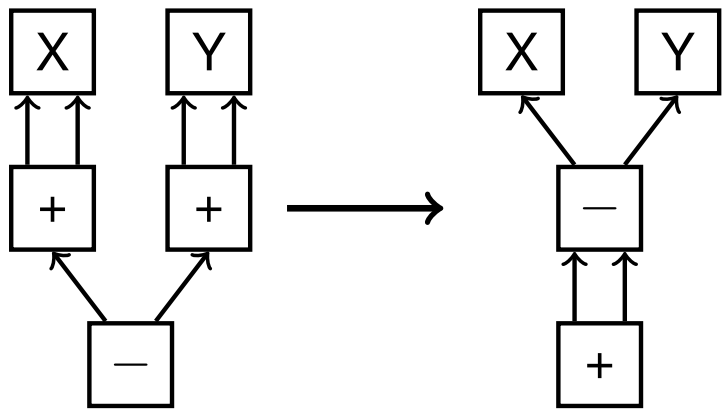
TECHNISCHE UNIVERSITÄT DARMSTADT

```
while (!WorkList.empty()) {
  Instruction *I = *WorkList.begin(); WorkList.erase(WorkList.begin());

  if (Constant *C = ConstantFoldInstruction(I, TD, TLI)) {
    // Add all of the users of this instruction to the worklist, they might
    // be constant propagatable now...
    for (Value::use_iterator UI = I->use_begin(), UE = I->use_end(); UI != UE; ++UI)
      WorkList.insert(cast<Instruction>(*UI));

    // Replace all of the uses of a variable with uses of the constant.
    I->replaceAllUsesWith(C);

    // Remove the dead instruction.
    WorkList.erase(I);
    I->eraseFromParent();
  }
}
```

- Inhalt von `ConstantFoldInstruction`: “Erzeuge neue Konstante, wenn alle Operanden von `I` konstant sind”

TECHNISCHE UNIVERSITÄT DARMSTADT

Implementierung einer Peephole-Optimierung



$(X + X) - (Y + Y)$ $\qquad$ $(X - Y) + (X - Y)$

(aus: Bersch, Thomas: Generierung lokaler Optimierungen. Diplomarbeit, 2012)

Optimierungen
Pass-Implementierung

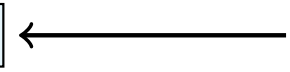
TECHNISCHE UNIVERSITÄT DARMSTADT

```
class SpecialSub : public FunctionPass {
  static char ID;
  SpecialSub() : FunctionPass(ID) {}

  bool runOnFunction(Function &F) {
    for (inst_iterator II = inst_begin(F), IE = inst_end(F);
         II != IE; ++II)
      performSpecialSubOptimization(&*II);
    return true;
  }
};

static RegisterPass<SpecialSub> X("specialsub",
  "Special subtraction transformation pattern");
```

Optimierungen
Pass-Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class SpecialSub : public FunctionPass {
  static char ID;
  SpecialSub() : FunctionPass(ID) {}

  bool runOnFunction(Function &F) {
    for (inst_iterator II = inst_begin(F), IE = inst_end(F);
         II != IE; ++II)
      performSpecialSubOptimization(&*II);
    return true;
  }
};

static RegisterPass<SpecialSub> X("specialsub",
  "Special subtraction transformation pattern");
```



Optimierungen
Pass-Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class SpecialSub : public FunctionPass {
  static char ID;
  SpecialSub() : FunctionPass(ID) {}

  bool runOnFunction(Function &F) {
    for (inst_iterator II = inst_begin(F), IE = inst_end(F);
         II != IE; ++II)
      performSpecialSubOptimization(&*II);
    return true;
  }
};

static RegisterPass<SpecialSub> X("specialsub",
  "Special␣subtraction␣transformation␣pattern");
```

erben

implementieren

Optimierungen

Pass-Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
class SpecialSub : public FunctionPass {
  static char ID;
  SpecialSub() : FunctionPass(ID) {}

  bool runOnFunction(Function &F) {
    for (inst_iterator II = inst_begin(F), IE = inst_end(F);
         II != IE; ++II)
      performSpecialSubOptimization(&*II);
    return true;
  }
};
```

← erben

← implementieren

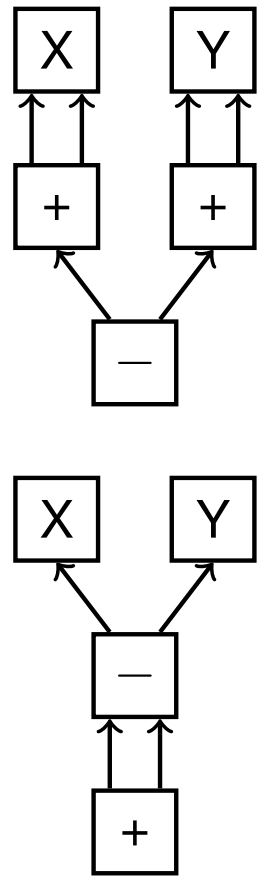
```
static RegisterPass<SpecialSub> X("specialsub",
  "Special subtraction transformation pattern");
```

← registrieren

Optimierungen
Transformation

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void performSpecialSubOptimization(Instruction *I) {
  if (match(I, m_Sub(
    m_Add(m_Value(X1), m_Value(X2)),
    m_Add(m_Value(Y1), m_Value(Y2))
    )) && X1 == X2 && Y1 == Y2) {

      newSub = CreateSub(X1, Y1, "newsub");
      newAdd = CreateAdd(newSub, newSub, "newadd");

      newSub->insertBefore(I);
      newAdd->insertBefore(I);

      I->replaceAllUsesWith(newAdd);
  }
}
```

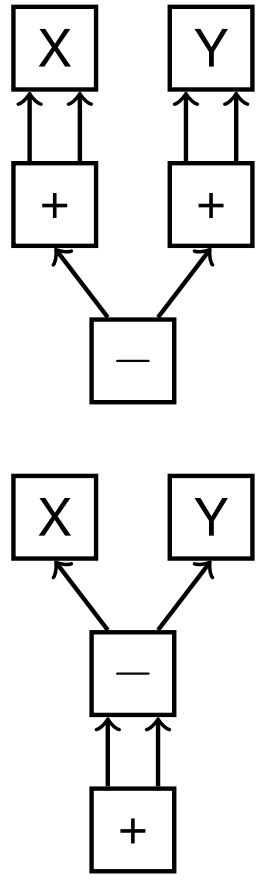


Optimierungen
Transformation

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void performSpecialSubOptimization(Instruction *I) {
  if (match(I, m_Sub(
    m_Add(m_Value(X1), m_Value(X2)),
    m_Add(m_Value(Y1), m_Value(Y2))
    )) && X1 == X2 && Y1 == Y2) {

      newSub = CreateSub(X1, Y1, "newsub");
      newAdd = CreateAdd(newSub, newSub, "newadd");

      newSub->insertBefore(I);
      newAdd->insertBefore(I);

      I->replaceAllUsesWith(newAdd);
  }
}
```

⟵ Muster finden



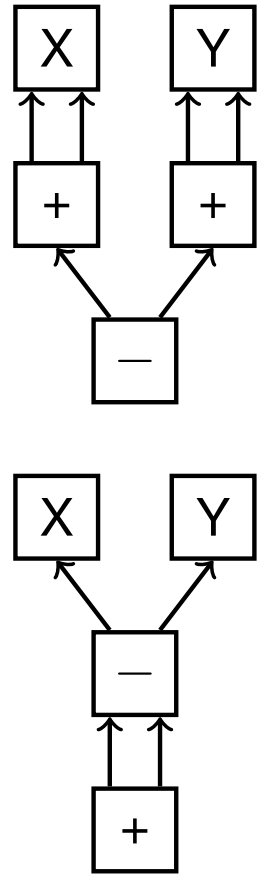
Optimierungen
Transformation

TECHNISCHE UNIVERSITÄT DARMSTADT

```
void performSpecialSubOptimization(Instruction *I) {
  if (match(I, m_Sub(
    m_Add(m_Value(X1), m_Value(X2)),
    m_Add(m_Value(Y1), m_Value(Y2))
    )) && X1 == X2 && Y1 == Y2) {

      newSub = CreateSub(X1, Y1, "newsub");
      newAdd = CreateAdd(newSub, newSub, "newadd");

      newSub->insertBefore(I);
      newAdd->insertBefore(I);

      I->replaceAllUsesWith(newAdd);
  }
}
```

⟵ Muster finden

↑erzeugen



Optimierungen
Transformation

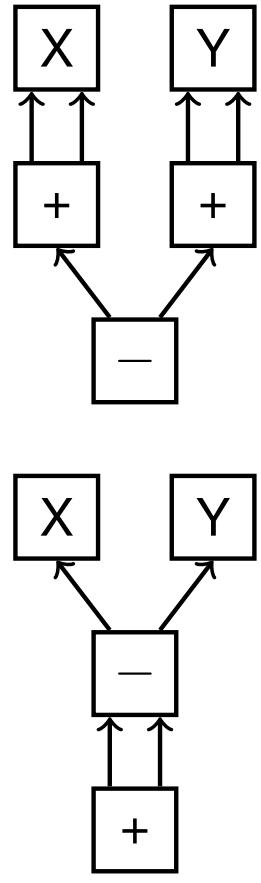
TECHNISCHE UNIVERSITÄT DARMSTADT

```
void performSpecialSubOptimization(Instruction *I) {
  if (match(I, m_Sub(
    m_Add(m_Value(X1), m_Value(X2)),
    m_Add(m_Value(Y1), m_Value(Y2))
    )) && X1 == X2 && Y1 == Y2) {

      newSub = CreateSub(X1, Y1, "newsub");
      newAdd = CreateAdd(newSub, newSub, "newadd");

      newSub->insertBefore(I);
      newAdd->insertBefore(I);

       I->replaceAllUsesWith(newAdd);
  }
}
```

← Muster finden

↑erzeugen

← einfügen



Optimierungen
Transformation

TECHNISCHE UNIVERSITÄT DARMSTADT

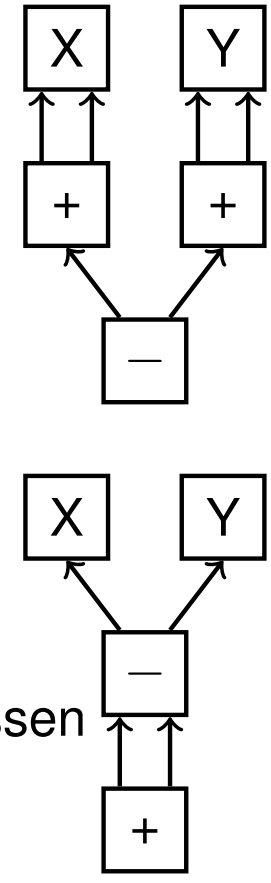
```
void performSpecialSubOptimization(Instruction *I) {
  if (match(I, m_Sub(
    m_Add(m_Value(X1), m_Value(X2)),
    m_Add(m_Value(Y1), m_Value(Y2))
    )) && X1 == X2 && Y1 == Y2) {

    newSub = CreateSub(X1, Y1, "newsub");
    newAdd = CreateAdd(newSub, newSub, "newadd");

    newSub->insertBefore(I);
    newAdd->insertBefore(I);

    I->replaceAllUsesWith(newAdd);
  }
}
```

← Muster finden

↑erzeugen

← einfügen

← Verwender anpassen



Optimierungen
Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int func(int a, int b) { return (a+a)-(b+b); }
```

Optimierungen
Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int func(int a, int b) { return (a+a)-(b+b); }
```

```
%add  = add i32 %a, %a
%add1 = add i32 %b, %b
%sub  = sub i32 %add, %add1
ret i32 %sub
```

Optimierungen
Beispiel


```
int func(int a, int b) { return (a+a)-(b+b); }
```

```
%add  = add i32 %a, %a
%add1 = add i32 %b, %b
%sub  = sub i32 %add, %add1
ret i32 %sub
```

⇓ `opt -load SpecialSub.so -specialsub -dce prog.ll` ⇓

Optimierungen
Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int func(int a, int b) { return (a+a)-(b+b); }
```

```
%add  = add i32 %a, %a
%add1 = add i32 %b, %b
%sub  = sub i32 %add, %add1
ret i32 %sub
```

⇓ `opt -load SpecialSub.so -specialsub -dce prog.ll` ⇓

```
%newsub = sub i32 %a, %b
%newadd = add i32 %newsub, %newsub
ret i32 %newadd
```

Optimierungen
Analysen nutzen

TECHNISCHE UNIVERSITÄT DARMSTADT

Optimierungen können Analyseinformationen anfordern:

```
virtual void getAnalysisUsage(AnalysisUsage &AU) const {
  AU.addRequired<AliasAnalysis>();
}
```

Verwendung:

```
...
AliasAnalysis &AA = getAnalysis<AliasAnalysis>();
if (AA.alias(V1, V2)) {
...
```

Interprozedurale Optimierungen

TECHNISCHE UNIVERSITÄT DARMSTADT

In LLVM gibt es u.a.:

- Inlining
- Interprozedurale Konstantenpropagation (einfach und Sparse Conditional Constant Propagation)

Als Beispiel: IPConstantPropagation

- Wenn ein Funktionsargument bei allen Aufrufen im Programm konstant ist, ersetze es in der Funktion.
- Wenn der Rückgabewert konstant ist, verwende ihn bei allen Aufrufen direkt.

IPCP

Beispiel: konstante Funktionsargumente

- Wie findet man alle Aufrufe einer Funktion?

```
for (Value::use_iterator UI = F.use_begin(), E = F.use_end(); UI != E; ++UI) {
  User *U = *UI;
```

- Wie iteriert man über die Argumente?

```
CallSite CS(cast<Instruction>(U));
// Check out all of the potentially constant arguments.
CallSite::arg_iterator AI = CS.arg_begin();
Function::arg_iterator Arg = F.arg_begin();
for (unsigned i = 0, e = ArgumentConstants.size(); i! e; ++i,++AI,++Arg) {

  // If this argument is known non-constant, ignore it.
  if (ArgumentConstants[i].second) continue;
```

IPCP

Beispiel: konstante Funktionsargumente

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ist das Argument eine Konstante?

```
Constant *C = dyn_cast<Constant>(*AI);
if (C && ArgumentConstants[i].first == 0) {
  ArgumentConstants[i].first = C;   // First constant seen.
```

- Sind zwei Konstanten gleich?

```
    } else if (C && ArgumentConstants[i].first == C) {
        // Still the constant value we think it is.
    } else {
      // Argument became non-constant.  If all arguments are non-constant now,
      // give up on this function.
      if (++NumNonconstant == ArgumentConstants.size()) return false;
      ArgumentConstants[i].second = true;
    }
  }
}
```

IPCP

Beispiel: konstante Funktionsargumente

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wie ersetzt man das Argument durch die gefundene Konstante?

```
Function::arg_iterator AI = F.arg_begin();
for (unsigned i = 0, e = ArgumentConstants.size(); i != e; ++i, ++AI) {
  // Do we have a constant argument?
  if (ArgumentConstants[i].second || AI->use_empty() ||
      (AI->hasByValAttr() && !F.onlyReadsMemory()))
    continue;

  Value *V = ArgumentConstants[i].first;
  if (V == 0) V = UndefValue::get(AI->getType());
  AI->replaceAllUsesWith(V);
}
```

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

- LLVM ist eine Sammlung von Bibliotheken und Werkzeugen zur Compilerentwicklung.
- LLVM-IR $\approx$ Assembler für virtuelle Maschine mit typischen RISC-Befehlssatz plus ein paar Extras.
- Beim Design der Infrastruktur wurde größter Wert auf Modularität und Wiederverwendbarkeit der Komponenten gelegt.

Quellen / weiterführende Informationen

TECHNISCHE UNIVERSITÄT DARMSTADT

- http://llvm.org
- http://llvm.org/docs/LangRef.html
- http://llvm.org/docs/ProgrammersManual.html
- http://llvm.org/docs/WritingAnLLVMPass.html
- http://llvm.org/docs/AliasAnalysis.html
- http://www.aosabook.org/en/llvm.html
- Chris Lattner und Vikram Adve: "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation".

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Backup-Folien

Cheatsheet

TECHNISCHE UNIVERSITÄT DARMSTADT

- C-Programm nach LLVM übersetzen:
  ```
  $ clang -S -emit-llvm -o prog.ll prog.c
  ```
- IR in “richtige” SSA-Form bringen:
  ```
  $ opt -S -mem2reg -o prog-ssa.ll prog.ll
  ```
- Zwischen Assembler- und Bitcode-Format konvertieren:
  ```
  $ llvm-as prog.ll # erzeugt prog.bc
  $ llvm-dis prog.bc # erzeugt prog.ll
  ```
- Codeerzeugung (= System-Assemblercode generieren)
  ```
  $ llc prog.ll # erzeugt prog.s
  ```