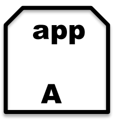
Compiler Construction
Lent Term 2014
Lecture 9 (of 16)

- **Assorted topics**
  - **bootstrapping**
  - **exceptions**

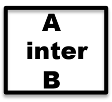
Timothy G. Griffin
tgg22@cam.ac.uk
Computer Laboratory
University of Cambridge

1

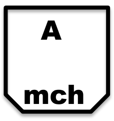
Bootstrapping. We need some notation . . .



An application called **app** written in language **A**

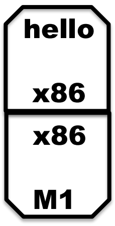


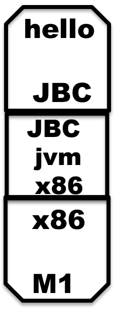
An interpreter or VM for language **A** Written in language **B**



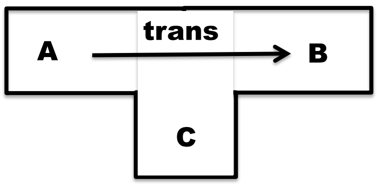
A machine called **mch** running language **A** natively.

Simple Examples



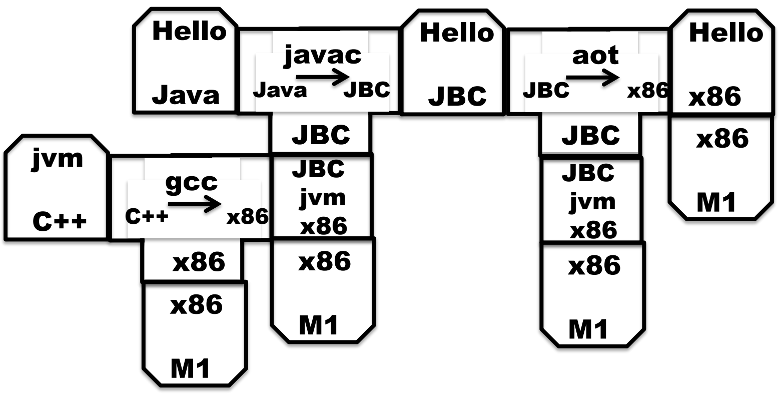


Tombstones



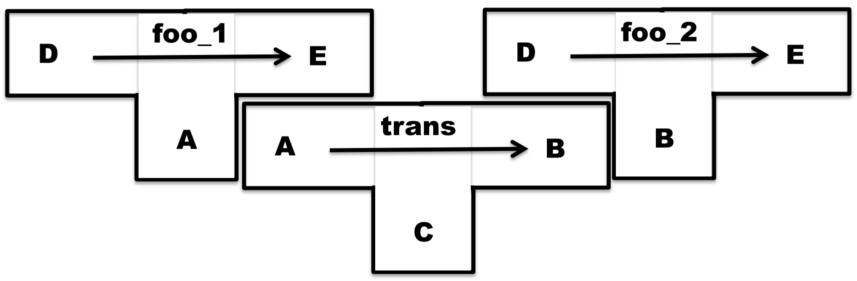
This is an application called **trans** that translates programs in language **A** into programs in language **B**, and it is written in language **C**.

Ahead-of-time compilation



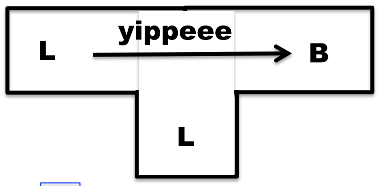
Thanks to David Greaves for the example.

Of course translators can be translated

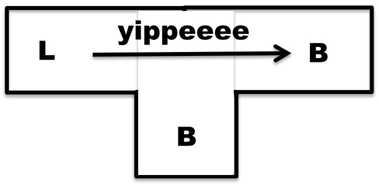


Translator **foo_2** is produced as output from **trans** when given **foo_1** as input.

Our seemingly impossible task



We have just invented a really great new language **L** (in fact we claim that "**L** is far superior to C++"). To prove how great **L** is we write a compiler for **L** in **L** (of course!). This compiler produces machine code **B** for a widely used instruction set (say **B** = x86).



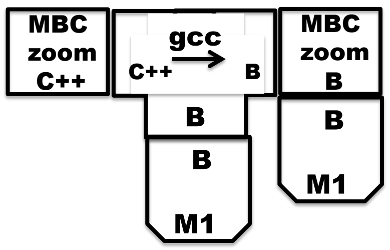
Furthermore, we want to compile our compiler so that it can run on a machine running **B.**

How can we compiler our compiler?

There are many many ways we could go about this task. The following slides simply sketch out one plausible route to fame and fortune.

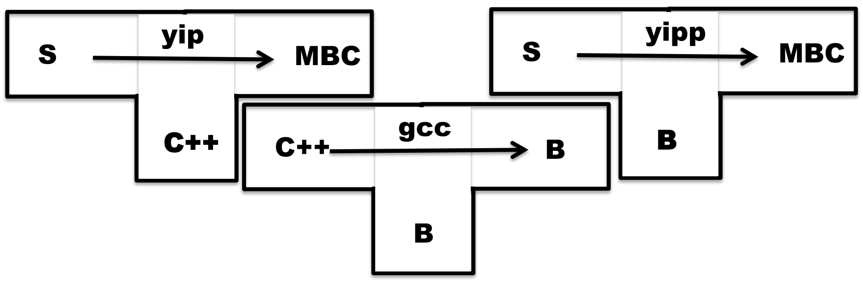
Step 1
Write a small interpreter (VM) for a small language of byte codes

MBC = My Byte Codes



The **zoom** machine!

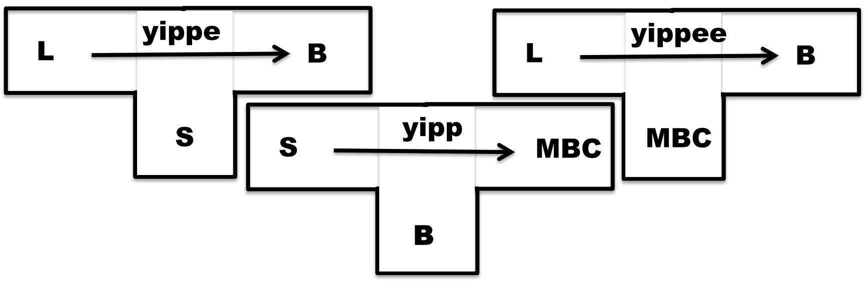
Step 2
Pick a small subset S of L and write a translator from S to MBC



Write **yip** by hand. (It sure would be nice if we could hide the fact that this is written is C++.)

Translator **yipp** is produced as output from **gcc** when **yip** is given as input.

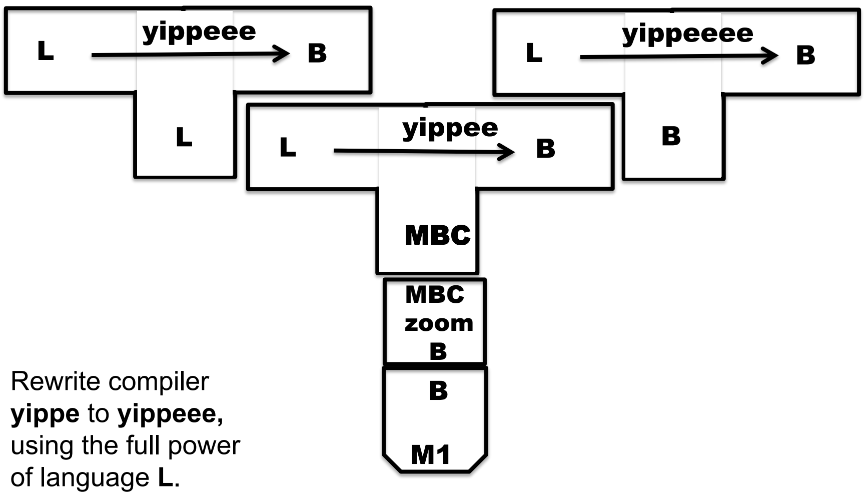
Step 3
Write a compiler for L in S



Write a compiler **yippe** for the full language **L**, but written only in the sub-language **S**.

Compile **yippe** using **yipp** to produce **yippee**

Step 4
Write a compiler for L in L

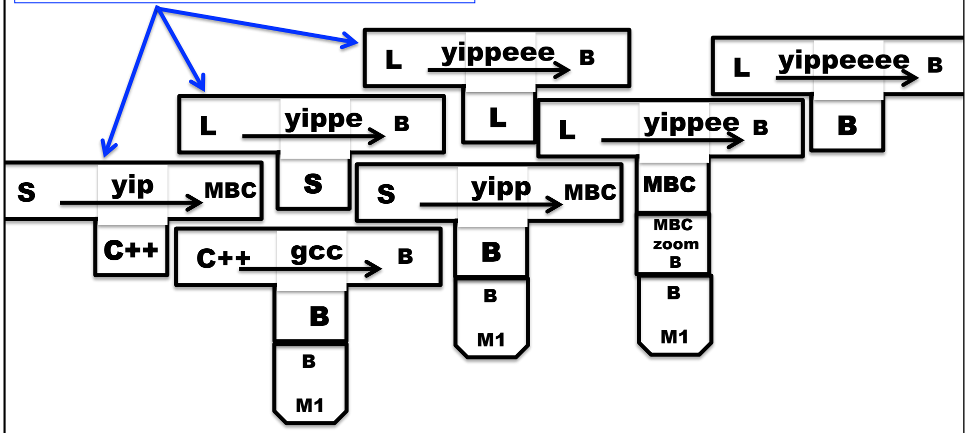


Rewrite compiler **yippe** to **yippeee,** using the full power of language **L**.

Now compile this using **yippee** to obtain our goal!

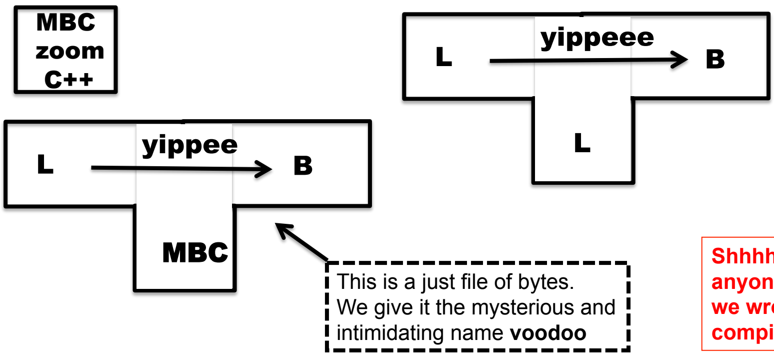
Putting it all together

We wrote only these compilers and the MBC VM.



Step 5
Cover our tracks and leave the world mystified and amazed

Our **L** compiler download site contains <u>only three</u> components:



This is a just file of bytes.
We give it the mysterious and intimidating name **voodoo**

Shhhh! Don't tell anyone that we wrote the first compiler in C++

Our instructions:

1. Use **gcc** to compile the **zoom** interpreter
2. Use **zoom** to run **voodoo** with input **yippeee** to produce output the compiler **yippeeee**

New Topic : Exceptions (informal description)

`e handle f`

If expression e evaluates "normally" to value v, then v is the result of the entire expression.

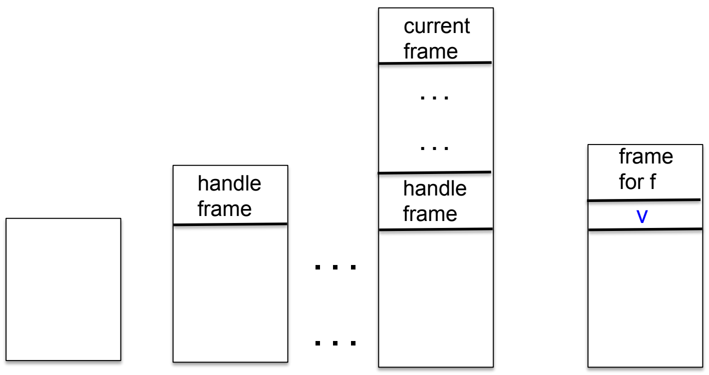
Otherwise, an exceptional value v' is "raised" in the evaluation of e, then result is (f v')

`raise e`

Evaluate expression e to value v, and then raise v as an exceptional value, which can only be "handled".

Implementation of exceptions may require a lot of language-specific consideration and care. Exceptions can interact in powerful and unexpected ways with other language features. Think of C++ and class destructors, for example.

Viewed from the call stack



Call stack just before evaluating code for

`e handle f`

Push a special frame for the handle

"`raise v`" is encountered while evaluating a function body associated with top-most frame

"Unwind" call stack. Depending on language, this may involve some "clean up" to free resources.

Possible pseudo-code implementation

```
e handle f
```

```
let fun _h27 () =
  build special “handle frame”
  save address of f in frame;
  … code for e …
  return value of e
in _h27 () end
```

```
raise e
```

```
… code for e …
save v, the value of e;
unwind stack until first
fp found pointing at a handle frame;
Replace handle frame with frame
for call to (extracted) f using
v as argument.
```