CMSC 430: Introduction to Compilers

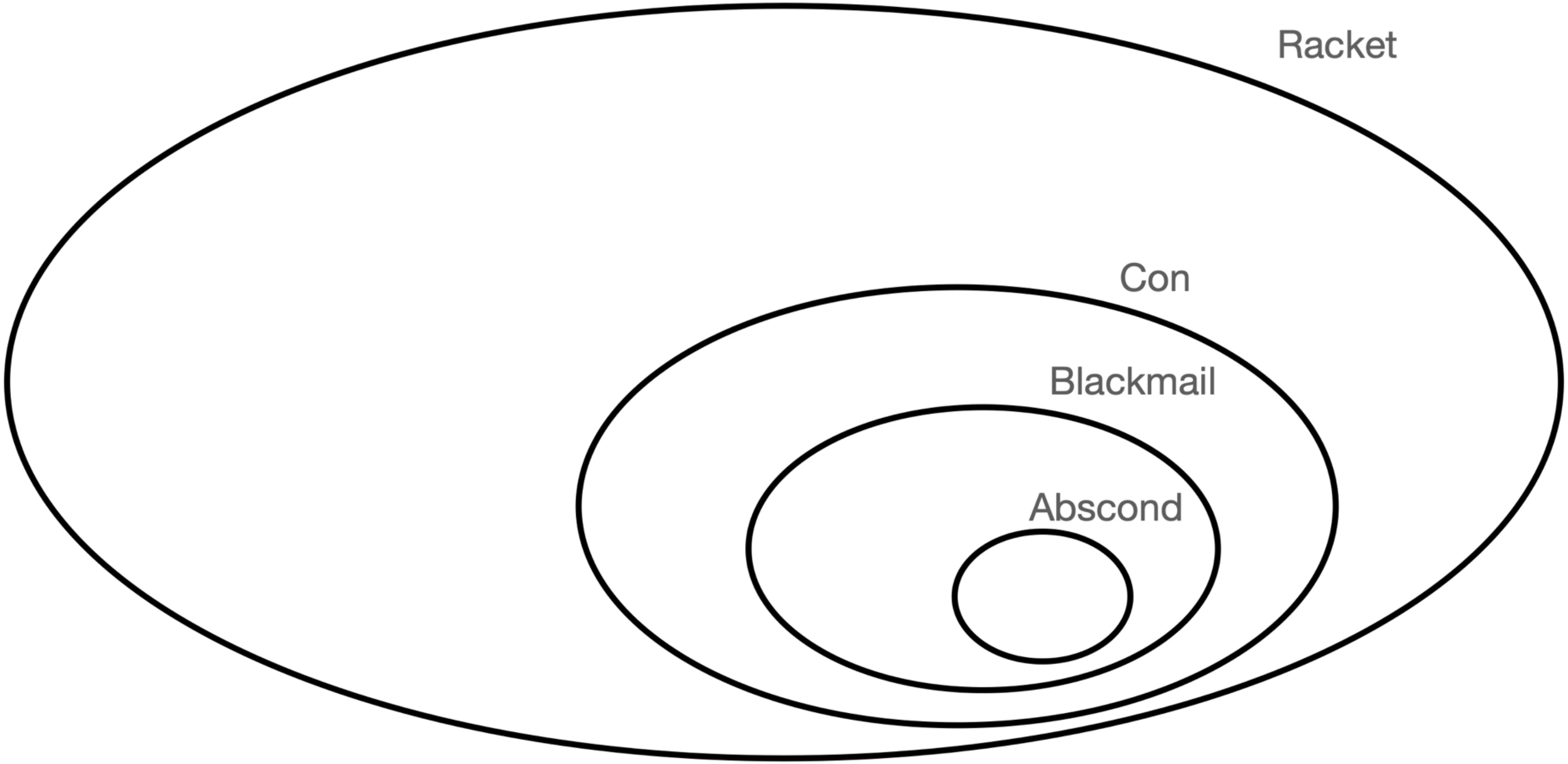
blackmail

Announcements 02/17/2026

- Assignment 2: due on Friday (02/20)
- Quiz 3: due Monday (02/23)
- Anonymous feedback form:
  - https://forms.gle/AgMDcDGfLfpyUeQY9

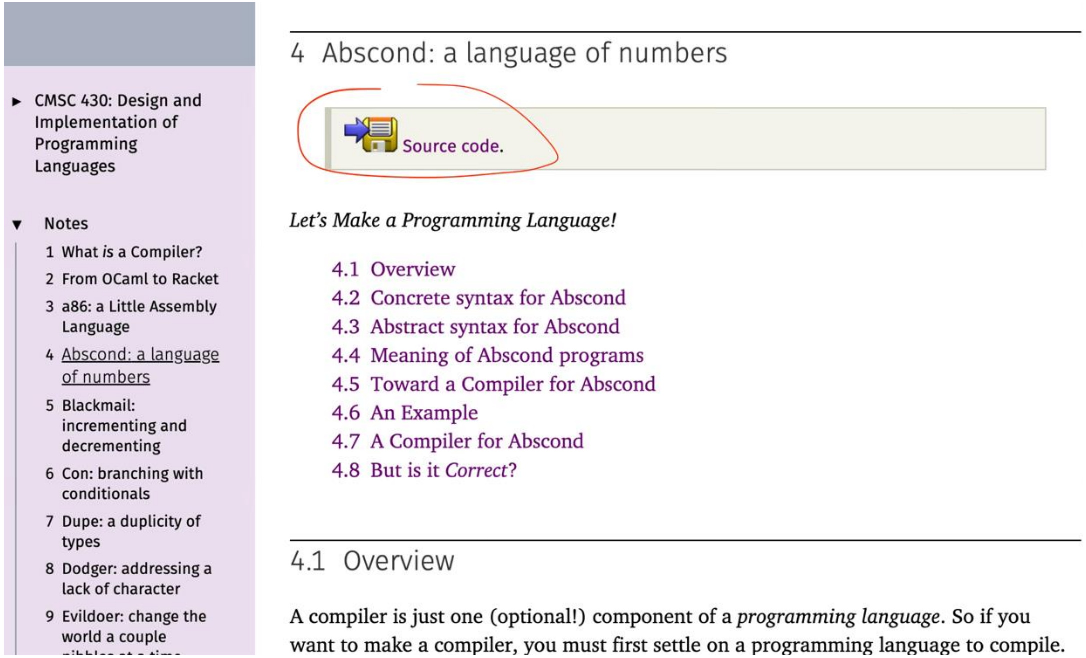
- Today
  - blackmail

Language Subsets



Accessing the source code

- Complete source code for each language linked to in notes:



- CMSC 430: Design and Implementation of Programming Languages
- Notes
  1 What *is* a Compiler?
  2 From OCaml to Racket
  3 a86: a Little Assembly Language
  4 Abscond: a language of numbers
  5 Blackmail: incrementing and decrementing
  6 Con: branching with conditionals
  7 Dupe: a duplicity of types
  8 Dodger: addressing a lack of character
  9 Evildoer: change the world a couple nibbles at a time

4 Abscond: a language of numbers

Source code.

Let's Make a Programming Language!

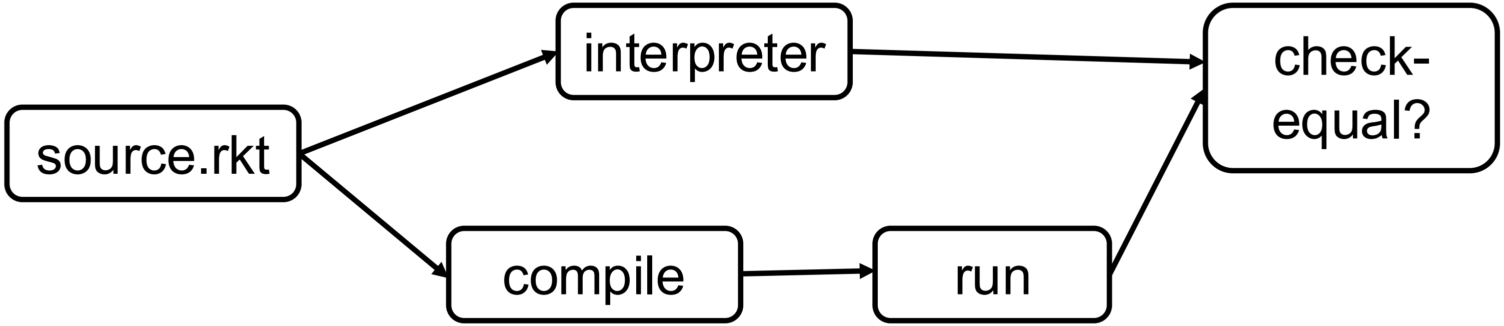
4.1 Overview
4.2 Concrete syntax for Abscond
4.3 Abstract syntax for Abscond
4.4 Meaning of Abscond programs
4.5 Toward a Compiler for Abscond
4.6 An Example
4.7 A Compiler for Abscond
4.8 But is it *Correct*?

4.1 Overview

A compiler is just one (optional!) component of a *programming language*. So if you want to make a compiler, you must first settle on a programming language to compile.

Language Specification: Definitional Interpreters

- Idea: write a program: `interp : Expr -> Value`
- simpler than writing compiler
- consider it the specification for compiler
- Compiler correctness:



Interpreter

- Reader : Input → S-Expr
  - The main function interp-stdin.rkt
- Parser: S-Expr → Expr
  - The parse function in parse.rkt
- Interpreter: Expr → Value
  - The interp function in interp.rkt

Interpreter structure

- interp-stdin.rkt: interpret source code on stdin to value on stdout
- ast.rkt: type definition for AST
- parse.rkt: s-expression to AST parser
- interp.rkt: AST interpreter

- run the interpreter
  - `racket -t interp-stdin.rkt -m`

Compiler

- Reader : Input → S-Expr
- Parser: S-Expr → Expr
- Compiler: Expr → a86
- Assembler: a86 → Object
- Linker: Object → Executable

- Runtime system: C code linked together w/ program object code

Compiler Structure

- compile-stdin.rkt: compile source code on stdin to x86 on stdout
- ast.rkt: type definition for AST
- parse.rkt: s-expression to AST parser
- compile.rkt: AST to a86 compiler
- main.c, print.c, print.h: run-time system

- runs the compiler
  - `racket -t compile-stdin.rkt -m`

Recipe for growing a language

- Write examples
- Extend concrete syntax
- Extend abstract syntax
- Extend parser
- Revise interpreter to specify semantics
- Revise compiler & run-time system to implement semantics
- Test against examples

Blackmail Grammar

$$e ::= \mathit{integer} \mid (\mathsf{add1}\ e) \mid (\mathsf{sub1}\ e)$$

- Example program:

```
#lang racket
(add1 (add1 40))
```

Blackmail AST

```
(struct Lit (i) #:prefab)
(struct Prim1 (p e) #:prefab)
```

- Examples:
  - `(Prim1 'add1 (Lit 0))`
  - `(Sub1 (Lit 120))`
  - `(Prim1 'add1 (Prim1 'add1 (Prim1 'add1 (Lit -42))))`

Blackmail Parser

```
(define (parse s)
  (match s
  [(? exact-integer?) (Lit s)]
  [(list (? op1? o) e) (Prim1 o (parse e))]
  [_ (error "parse error")]))


(define (op1? op)
  (memq op '(add1 sub1)))
```

The actual parser (parse.rkt) will generate more helpful error messages.

Blackmail Compiler

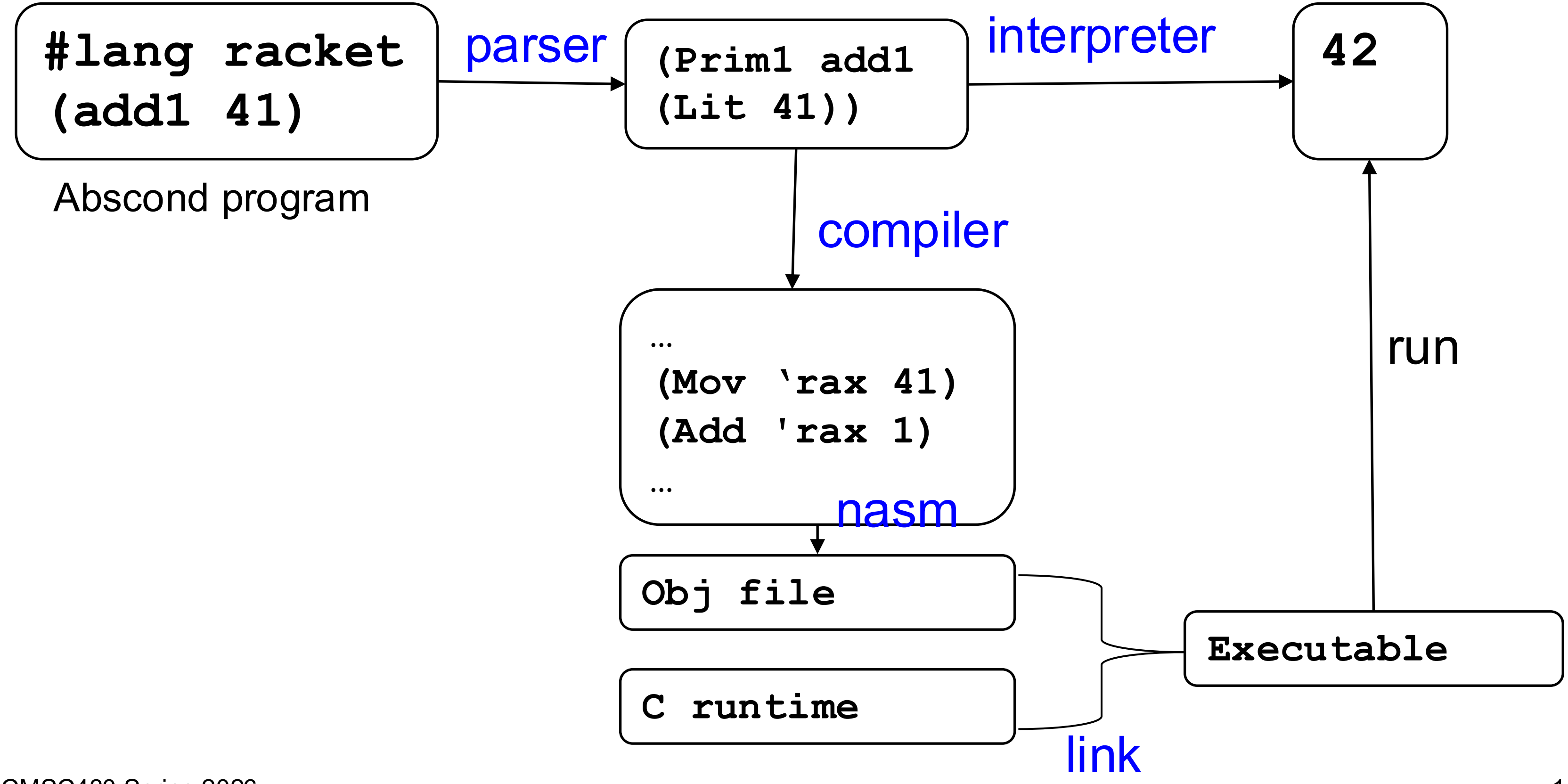
For:

```
(add1 (add1 40))
```

We Produce:

```
(Mov rax 40)
(Add rax 1)
(Add rax 1)
```

Blackmail: components


#lang racket
(add1 41)
Abscond program
parser
(Prim1 add1
(Lit 41))
interpreter
42
compiler
…
(Mov `rax 41)
(Add 'rax 1)
…
nasm
Obj file
C runtime
link
Executable
run