# Desugaring: Our First Elaborator

CS 350

---

Dr. Joseph Eremondi

Last updated: July 9, 2024

# Syntactic Sugar

# What is syntactic sugar

- Some language features strictly increase the power of a language

# What is syntactic sugar

- Some language features strictly increase the power of a language
  - Let you do things that can’t be done any other way

# What is syntactic sugar

- Some language features strictly increase the power of a language
  - Let you do things that can't be done any other way
- Some language features aren't strictly necessary, but are nice to have

# What is syntactic sugar

- Some language features strictly increase the power of a language
  - Let you do things that can't be done any other way
- Some language features aren't strictly necessary, but are nice to have
  - Otherwise we'd just code in machine code/assembly all the time

# What is syntactic sugar

- Some language features strictly increase the power of a language
  - Let you do things that can't be done any other way
- Some language features aren't strictly necessary, but are nice to have
  - Otherwise we'd just code in machine code/assembly all the time
- These "nice to haves" are called *syntactic sugar*

# What is syntactic sugar

- Some language features strictly increase the power of a language
  - Let you do things that can't be done any other way
- Some language features aren't strictly necessary, but are nice to have
  - Otherwise we'd just code in machine code/assembly all the time
- These "nice to haves" are called *syntactic sugar*
  - They "sweeten" the experience of programming

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter
  - Instead, translate to a smaller “core” AST type

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter
  - Instead, translate to a smaller “core” AST type
  - Keeps the interpreter small, easier to maintain

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter
  - Instead, translate to a smaller “core” AST type
  - Keeps the interpreter small, easier to maintain
- Have two types for AST

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter
  - Instead, translate to a smaller “core” AST type
  - Keeps the interpreter small, easier to maintain
- Have two types for AST
  - Surface AST

# Desugaring

- When one feature can be expressed in terms of another, sometimes we implement it by *desugaring*
  - Translating the AST for a feature into other language features
- Desugared features: no case in the interpreter
  - Instead, translate to a smaller “core” AST type
  - Keeps the interpreter small, easier to maintain
- Have two types for AST
  - Surface AST
  - Core AST

# Example: Subtraction

- We’ll add subtraction to our language

# Example: Subtraction

- We'll add subtraction to our language
  - *without changing the interpreter at all*

# Example: Subtraction

- We'll add subtraction to our language
  - *without changing the interpreter at all*
- Separate AST into surface and core AST

# Example: Subtraction

- We'll add subtraction to our language
  - *without changing the interpreter at all*
- Separate AST into surface and core AST
- Add translation from surface to core AST

# Subtraction: Datatype

- First need the surface AST

# Subtraction: Datatype

- First need the surface AST

# Subtraction: Datatype

- First need the surface AST

```
(define-type SurfaceExpr
  (SurfNumLit [n : Number])
  (SurfPlus [left : SurfaceExpr]
            [right : SurfaceExpr])
  (SurfTimes [left : SurfaceExpr]
             [right : SurfaceExpr])
  (SurfIf0 [test : SurfaceExpr]
           [thenBranch : SurfaceExpr]
           [elseBranch : SurfaceExpr])
  (SurfSub [left : SurfaceExpr]
           [right : SurfaceExpr]))
```

# Subtraction: Parser Case

# Subtraction: Parser Case

```
(define (parse [s : S-Exp]) : SurfaceExpr
  ...
    [(s-exp-match? `{- ANY ANY} s)
     (SurfSub (parse (second (s-exp->list s)))
              (parse (third (s-exp->list s))))])
```

# Subtraction: Elaborator

- Define function that translates from `SurfaceExpr` to `Expr`

# Subtraction: Elaborator

- Define function that translates from `SurfaceExpr` to `Expr`

# Subtraction: Elaborator

- Define function that translates from `SurfaceExpr` to `Expr`

```
(define (elab [surf : SurfaceExpr]) : Expr
  (type-case SurfaceExpr surf
             [(SurfNumLit n) (NumLit n)]
             [(SurfPlus x y) (Plus x y)]
             [(SurfTimes x y) (Times x y)]
             [(SurfIfo test thn els) (Ifo test thn els)]
             ;; Sub isn't in our core syntax
             [(SurfSub x y) (Plus x (Times (NumLit -1) y))])
  )
```

# Compilation vs. Elaboration vs. Desugaring

- Technically we just wrote our first compiler

# Compilation vs. Elaboration vs. Desugaring

- Technically we just wrote our first compiler
  - Translated a small language into an even smaller one

# Compilation vs. Elaboration vs. Desugaring

- Technically we just wrote our first compiler
  - Translated a small language into an even smaller one
- Compilation is just a bunch of desugaring passes

# Compilation vs. Elaboration vs. Desugaring

- Technically we just wrote our first compiler
  - Translated a small language into an even smaller one
- Compilation is just a bunch of desugaring passes
  - Simpler and simpler languages untill we have something simple enough for assembly code