# S9: Application of Programming Principles

CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2018



## Advanced Programming Principles

- We've introduced a number of topics, all realized in the OCaml language.
- But we can, and you will, apply these ideas in other languages.



## Functional programming

- What we've seen in OCaml are a pure view of functional programming ideas and constructs.
- There is some migration of these ideas into mainstream languages:
  - parametric polymorphism: Java generics
  - garbage collection: Java, C#, Python ...
  - lambda expressions: Java 8, Python
  - disjoint unions: Scala, Swift, Hack
  - static type inference: very limited forms in C#
- Clearly less support for disjoint unions and type inference.
- There is little we can do about losing out on type inference, but we can see how to "code up" disjoint unions in other languages.

# Disjoint unions in C

- Disjoint unions are "sums of products."
- So what kind of sum and product types do we have in C?



# Disjoint unions in C

- The `struct` is a product type.
- The `union` is a sum type.
- We can use these to build types and values corresponding to disjoint unions.
- Questions: How safe is it? How convenient is it?



We'll build something similar to our first expression evaluator in C.

The code is in `SamplePrograms/eval_in_C.c` in the public repository.

# C structs

```
struct bin_op_struct {
    struct expr *l ;
    struct expr *r ;
} ;
```



# C enums

```
enum tag {add, mul, cnst, neg} ;
```



# C unions

```
union all_components {
    struct bin_op_struct add_components ;
    struct bin_op_struct mul_components ;
    int v ;
    struct expr *ne ;
} ;
```

# The recipe

To implement a disjoint union in C:

- for each value constructor
  - we need a field in a union,
  - its components may be put in struct,
  - a tag in an enumerated type is created
- a struct for the type is also created to hold
  - the tag
  - and the union of all possible values



# Assessment

How safe is it?



# Assessment

How safe is it?

- It is not safe. This exposes a hole in the C type system.
- Thus, C has a weak type system since type errors can go undetected.

## Assessment

How convenient is it?



## Assessment

How convenient is it?

- Not very. Quite painful actually.
- So painful that it is rarely done and even less safe ways are used.
- C is fine for many applications, but it is entirely unsuited for complex symbolic data.



## Disjoint unions in OOP

- Constructors in an disjoint union are sometime called variants.
- Each defines a different kind or variant of the type.
- It is natural to think of sub types here, and thus classes and sub classes.
- So what might we do to "code up" inductive types in OOP?

# Disjoint unions in OOP

We create

- an abstract class `Expr` with a method named `eval`.
- a subclass `Add` of `Expr`

  It has field `l` and `r` of type `Expr`.

  Its constructor initialized them.

  It implements the `eval` method appropriately.
- Create similar subclasses for other constructors.



# The recipe

To implement a disjoint union in an OOP language:

- an abstract class for the type is defined.
- for each value constructor
  - a subclass is created
  - it has fields for each component in the value constsructor's product
  - its constructor method has parameters for each of these values.



# Lacking pattern matching

- It seems the main pattern matching in `eval` is handled by dynamic dispatch in OO languages.
- What about type checking and the need to inspect types?

```
type typ = IntType | BoolType |
           FuncType if typ * typ | ErrType
let rec check (e: expr) : typ = match e with
  | App (e1, e2) ->
    let t1 = check e1 in
    let t2 = check e2 in
      match t1 with
        | FuncType (inType, outType) -> ...
        | _ -> ErrType
```

# Assessment

How safe is it?



# Assessment

How safe is it?

- It is safe. There are no holes in an OOP type system that arise because of this.



# Assessment

How convenient is it?

## Assessment

How convenient is it?

- Not very. Still seems rather verbose.
- It isn't as painful as in C.
  But at least in C we can inspect the data directly.
- Objects are opaque, sometimes this is useful, sometimes not.



## Extensibility of software

Question: Can new variants or new operations easily be added to a data type as a single unit?



## Extensibility of software

- With disjoint unions
- With classes

This is the jumping off point for my research in extensible languages.

So maybe a few slides about this might be of interest.