Tagged union

From Wikipedia, the free encyclopedia

In computer science, a **tagged union**, also called a **variant**, **variant record**, **discriminated union**, **disjoint union**, or **sum type**, is a data structure used to hold a value that could take on several different, but fixed, types. Only one of the types can be in use at any one time, and a **tag** field explicitly indicates which one is in use. It can be thought of as a type that has several "cases," each of which should be handled correctly when that type is manipulated. Like ordinary unions, tagged unions can save storage by overlapping storage areas for each type, since only one is in use at a time.

Tagged unions are most important in functional languages such as ML and Haskell, where they are called **datatypes** (see algebraic data type) and the compiler is able to verify that all cases of a tagged union are always handled, avoiding many types of errors. They can, however, be constructed in nearly any language, and are much safer than untagged unions, often simply called unions, which are similar but do not explicitly keep track of which member of the union is currently in use.

Tagged unions are often accompanied by the concept of a type constructor, which is similar but not the same as a constructor for a class. Type constructors produce a tagged union type, given the initial tag type and the corresponding type.

Mathematically, tagged unions correspond to *disjoint* or *discriminated unions*, usually written using +. Given an element of a disjoint union *A* + *B*, it is possible to determine whether it came from *A* or *B*. If an element lies in both, there will be two effectively distinct copies of the value in *A* + *B*, one from *A* and one from *B*.

In type theory, a tagged union is called a **sum type**. Sum types are the dual of product types. Notations vary, but usually the sum type $A + B$ comes with two introduction forms $\mathrm{inj}_1 : A \to A + B$ and $\mathrm{inj}_2 : B \to A + B$. The elimination form is case analysis, known as pattern matching in ML-style programming languages: if e has type $A + B$ and e_1 and e_2 have type τ under the assumptions $x : A$ and $y : B$ respectively, then the term $\mathsf{case}\ e\ \mathsf{of}\ x \Rightarrow e_1 | y \Rightarrow e_2$ has type τ. The sum type corresponds to intuitionistic logical disjunction under the Curry–Howard correspondence.

An enumerated type can be seen as a degenerate case: a tagged union of unit types. It corresponds to a set of nullary constructors and may be implemented as a simple tag variable, since it holds no additional data besides the value of the tag.

Many programming techniques and data structures – including rope (data structure), lazy evaluation, class hierarchy (see below), arbitrary-precision arithmetic, CDR coding, the indirection bit and other kinds of tagged pointers, etc. – are usually implemented using some sort of tagged union.

A tagged union can be seen as the simplest kind of self-describing data format. The tag of the tagged union can be seen as the simplest kind of metadata.

Contents

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Polski
Русский
中文

Edit links

Advantages and disadvantages [edit]

The primary advantage of a tagged union over an untagged union is that all accesses are safe, and the compiler can even check that all cases are handled. Untagged unions depend on program logic to correctly identify the currently active field, which may result in strange behavior and hard-to-find bugs if that logic fails.

The primary advantage of a tagged union over a simple record containing a field for each type is that it saves storage by overlapping storage for all the types. Some implementations reserve enough storage for the largest type, while others dynamically adjust the size of a tagged union value as needed. When the value is immutable, it is simple to allocate just as much storage as is needed.

The main disadvantage of tagged unions is that the tag occupies space. Since there are usually a small number of alternatives, the tag can often be squeezed into 2 or 3 bits wherever space can be found, but sometimes even these bits are not available. In this case, a helpful alternative may be **folded**, **computed** or **encoded tags**, where the tag value is dynamically computed from the contents of the union field. Common examples of this are the use of *reserved values*, where, for example, a function returning a positive number may return -1 to indicate failure, and sentinel values, most often used in tagged pointers.

Sometimes, untagged unions are used to perform bit-level conversions between types, called reinterpret casts in C++. Tagged unions are not intended for this purpose; typically a new value is assigned whenever the tag is changed.

Many languages support, to some extent, a universal data type, which is a type that includes every value of every other type, and often a way is provided to test the actual type of a value of the universal type. These are sometimes referred to as *variants*. While universal data types are comparable to tagged unions in their formal definition, typical tagged unions include a relatively small number of cases, and these cases form different ways of expressing a single coherent concept, such as a data structure node or instruction. Also, there is an expectation that every possible case of a tagged union will be dealt with when it is used. The values of a universal data type are not related and there is no feasible way to deal with them all.

Like option types and exception handling, tagged unions are sometimes used to handle the occurrence of exceptional results. Often these tags are folded into the type as "reserved values", and their occurrence is not consistently checked: this is a fairly common source of programming errors. This use of tagged unions can be formalized as a monad with the following functions:

$$\text{return}\colon A \to (A + E) = a \mapsto \text{value}\, a$$

$$\text{bind}\colon (A + E) \to (A \to (B + E)) \to (B + E) = a \mapsto f \mapsto \begin{cases} \text{err}\, e & \text{if } a = \text{err}\, e \\ f\, a' & \text{if } a = \text{value}\, a' \end{cases}$$

where "value" and "err" are the constructors of the union type, *A* and *B* are valid result types and *E* is the type of error conditions. Alternately, the same monad may be described by *return* and two additional functions, *fmap* and *join*:

$$\text{fmap}\colon (A \to B) \to ((A + E) \to (B + E)) = f \mapsto a \mapsto \begin{cases} \text{err}\, e & \text{if } a = \text{err}\, e \\ \text{value}\, f\, a' & \text{if } a = \text{value}\, a' \end{cases}$$

$$\text{join}\colon ((A + E) + E) \to (A + E) = a \mapsto \begin{cases} \text{err}\, e & \text{if } a = \text{err}\, e \\ \text{err}\, e & \text{if } a = \text{value}\, \text{err}\, e \\ \text{value}\, a' & \text{if } a = \text{value}\, \text{value}\, a' \end{cases}$$

Examples [edit]

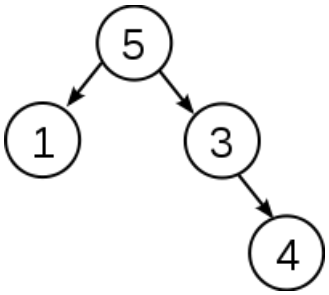
Say we wanted to build a binary tree of integers. In ML, we would do this by creating a datatype like this:

```
datatype tree = Leaf
              | Node of (int * tree * tree)
```

This is a tagged union with two cases: one, the leaf, is used to terminate a path of the tree, and functions much like a null value would in imperative languages. The other branch holds a node, which contains an integer and a left and right subtree. Leaf and Node are the constructors, which enable us to actually produce a particular tree, such as:

```
Node(5, Node(1,Leaf,Leaf), Node(3, Leaf, Node(4, Leaf, Leaf)))
```

which corresponds to this tree:



Now we can easily write a typesafe function that, say, counts the number of nodes in the tree:

```
fun countNodes(Leaf) = 0
  | countNodes(Node(int,left,right)) =
      1 + countNodes(left) + countNodes(right)
```

Timeline of language support [edit]

1960s [edit]

In ALGOL 68, tagged unions are called *united modes*, the tag is implicit, and the `case` construct is used to determine which field is tagged:

```
mode node = union (real, int, compl, string);
```

Usage example for `union` `case` of `node`:

```
node n := "1234";

case n in
  (real r):   print(("real:", r)),
  (int i):    print(("int:", i)),
  (compl c):  print(("compl:", c)),
  (string s): print(("string:", s))
 out          print(("?:", n))
esac
```

1970s & 1980s [edit]

Although primarily only functional languages such as ML and Haskell (from 1990s) give a central role to tagged unions and have the power to check that all cases are handled, other languages have support for tagged unions as well. However, in practice they can be less efficient in non-functional languages due to optimizations enabled by functional language compilers that can eliminate explicit tag checks and avoid explicit storage of tags.

Pascal, Ada, and Modula-2 call them **variant records** (formally **discriminated type** in Ada), and require the tag field to be manually created and the tag values specified, as in this Pascal example:

```
type shapeKind = (square, rectangle, circle);
     shape = record
                centerx : integer;
                centery : integer;
                case kind : shapeKind of
                   square : (side : integer);
                   rectangle : (length, height : integer);
                   circle : (radius : integer);
       end;
```

and this Ada equivalent:

```
type Shape_Kind is (Square, Rectangle, Circle);
type Shape (Kind : Shape_Kind) is record
   Center_X : Integer;
   Center_Y : Integer;
```

```
   case Kind is
      when Square =>
         Side : Integer;
      when Rectangle =>
         Length, Height : Integer;
      when Circle =>
         Radius : Integer;
   end case;
end record;

-- Any attempt to access a member whose existence depends
-- on a particular value of the discriminant, while the
-- discriminant is not the expected one, raises an error.
```

In C and C++, a tagged union can be created from untagged unions using a strict access discipline where the tag is always checked:

```
enum ShapeKind { Square, Rectangle, Circle };

struct Shape {
    int centerx;
    int centery;
    enum ShapeKind kind;
    union {
        struct { int side; };           /* Square */
        struct { int length, height; }; /* Rectangle */
        struct { int radius; };         /* Circle */
    };
};

int getSquareSide(struct Shape* s) {
    assert(s->kind == Square);
    return s->side;
}

void setSquareSide(struct Shape* s, int side) {
    s->kind = Square;
    s->side = side;
}

/* and so on */
```

As long as the union fields are only accessed through the functions, the accesses will be safe and correct. The same approach can be used for encoded tags; we simply decode the tag and then check it on each access. If the inefficiency of these tag checks is a concern, they may be automatically removed in the final version.

C and C++ also have language support for one particular tagged union: the possibly-null pointer. This may be compared to the `option` type in ML or the `Maybe` type in Haskell, and can be seen as a tagged pointer: a tagged union (with an encoded tag) of two types:

- Valid pointers,
- A type with only one value, `null`, indicating an exceptional condition.

Unfortunately, C compilers do not verify that the null case is always handled, and this is a particularly prevalent source of errors in C code, since there is a tendency to ignore exceptional cases.

2000s [edit]

One advanced dialect of C called Cyclone has extensive built-in support for tagged unions. See the tagged union section of the on-line manual for more information.

The enum types in the Rust and Swift languages also work as tagged unions.

The variant library from Boost has demonstrated it was possible to implement a safe tagged union as a library in C++, visitable using functors.

```
struct display : boost::static_visitor<void>
{
    void operator()(int i)
```

```
    {
        std::cout << "It's an int, with value " << i << std::endl;
    }

    void operator()(const std::string& s)
    {
        std::cout << "It's a string, with value " << s << std::endl;
    }
};

boost::variant<int, std::string> v = 42;
boost::apply_visitor(display(), v);

boost::variant<int, std::string> v = "hello world";
boost::apply_visitor(display(), v);
```

Scala has case classes:

```
sealed abstract class Tree
case object Leaf extends Tree
case class Node(value: Int, left: Tree, right: Tree) extends Tree

val tree = Node(5, Node(1,Leaf,Leaf), Node(3, Leaf, Node(4, Leaf, Leaf)))
```

Because the class hierarchy is sealed, the compiler can check that all cases are handled in a pattern match:

```
tree match {
  case Node(x, _, _) => println("top level node value: " + x)
  case Leaf          => println("top level node is a leaf")
}
```

Scala's case classes also permit reuse through subtyping:

```
sealed abstract class Shape(centerX: Int, centerY: Int)
case class Square(side: Int, centerX: Int, centerY: Int) extends Shape(centerX,
centerY)
case class Rectangle(length: Int, height: Int, centerX: Int, centerY: Int) extends
Shape(centerX, centerY)
case class Circle(radius: Int, centerX: Int, centerY: Int) extends Shape(centerX,
centerY)
```

Class hierarchies as tagged unions [edit]

In a typical class hierarchy in object-oriented programming, each subclass can encapsulate data unique to that class. The metadata used to perform virtual method lookup (for example, the object's vtable pointer in most C++ implementations) identifies the subclass and so effectively acts as a tag identifying the particular data stored by the instance (see RTTI). An object's constructor sets this tag, and it remains constant throughout the object's lifetime.

Nevertheless, a class hierarchy involves true subtype polymorphism; it can be extended by creating further subclasses of the same base type, which could not be handled correctly under a tag/dispatch model. Hence, it is usually not possible to do case analysis or dispatch on a subobject's 'tag' as one would for tagged unions. Some languages such as Scala allow base classes to be "sealed", and unify tagged unions with sealed base classes.

See also [edit]

- Discriminator, the type tag for discriminated unions in CORBA
- Apache Thrift, an RPC system with tagged unions

External links [edit]

- boost::variant is a C++ typesafe discriminated union

- std.variant is an implementation of variant type in D 2.0

v · t · e	Data types
Uninterpreted	Bit · Byte · Trit · Tryte · Word
Numeric	Bignum · Complex · Decimal · Fixed point · Floating point (Double precision · Extended precision · Half precision · Minifloat · Octuple precision · Quadruple precision · Single precision) · Integer (signedness) · Interval · Rational
Text	Character · String (null-terminated)
Pointer	Address (physical · virtual) · Reference
Composite	Algebraic data type (generalized) · Array · Associative array · Class · Dependent · Equality · Inductive · List · Object (metaobject) · Option type · Product · Record · Set · Union (**tagged**)
Other	Boolean · Bottom type · Collection · Enumerated type · Exception · Function type · Opaque data type · Recursive data type · Semaphore · Stream · Top type · Type class · Unit type · Void
Related topics	Abstract data type · Data structure · Generic · Kind (metaclass) · Parametric polymorphism · Primitive data type · Protocol (interface) · Subtyping · Type constructor · Type conversion · Type system

Categories: Data types | Type theory