1. 概述

本文档描述当前时间对Alioth类型系统的最新思考,讨论了关于数据类型的分类的更合理的方案,讨论了关于元素原型和数据类型的新的设计方案,此外还讨论了如何进行类型转换,引入了`类型转换图`来描述类型转换关系

最后,本文档附带了对`aliothc 0.0.1`制定的类型系统重构计划方案.

2. 数据类型分类

数据类型的分类影响着数据类型所携带的信息被使用和修改时的效率和复杂度.

在上一代类型系统中,Alioth将数据类型设计为无嵌套的平坦结构,使用指针层次数字来判定它是否成为一个指针.

而实际开发过程中,我们发现,指针数据类型对应的运算总是特殊的,我们应该先得知一个数据类型是指针,之后再得知这个指针所指向的数据类型是如何,才更合理.

因此,在新一代的Alioth类型系统中,我们引入了全新的数据类型表达形式设计.

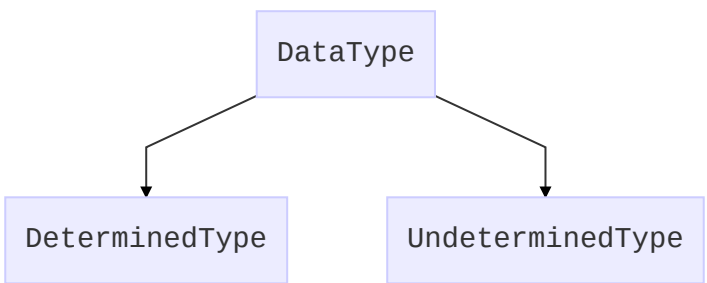
DataType	Flags
VoidType	`DeterminedType`
UnconstraintedPointer	`DeterminedType|PointerType`
ConstraintedPointer	`DeterminedType|PointerType`
Boolean	`DeterminedType|BasicDataType`
Int8	`DeterminedType|BasicDataType|IntegerType|SignedIntegerType`
Int16	`DeterminedType|BasicDataType|IntegerType|SignedIntegerType`
Int32	`DeterminedType|BasicDataType|IntegerType|SignedIntegerType`
Int64	`DeterminedType|BasicDataType|IntegerType|SignedIntegerType`
Uint8	`DeterminedType|BasicDataType|IntegerType|UnsignedIntegerType`
Uint16	`DeterminedType|BasicDataType|IntegerType|UnsignedIntegerType`
Uint32	`DeterminedType|BasicDataType|IntegerType|UnsignedIntegerType`
Uint64	`DeterminedType|BasicDataType|IntegerType|UnsignedIntegerType`
Float32	`DeterminedType|BasicDataType|FloatPointType`
Float64	`DeterminedType|BasicDataType|FloatPointType`
CompositeType	`DeterminedType`
NamedType	`UndeterminedType`
UnknownType	`UndeterminedType`

2.1. 总览

首先,数据类型在最大的尺度上,被分为`DeterminedType`和`UndeterminedType`两种类型,在这两种类型之下还有若干层分类,接下来的每个子标题讨论其中一个拥有子类的分类.

`DeterminedType`是确定的数据类型,可以直接参与中间代码的生成工作.

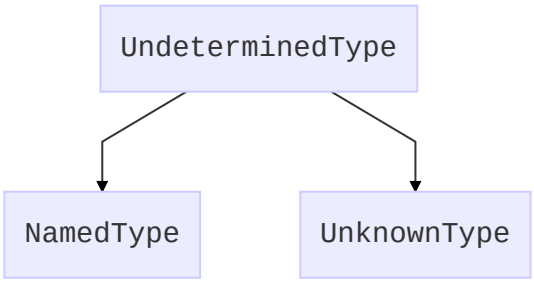
`UndeterminedType`是未确定的数据类型,需要语义分析器进行辅助的类型推导工作.



> 数据类型的不足:
>
> 当前的数据类型系统的设计,没有考虑SSE指令集所支持的数据包类型,或更长的比如80bit长度的浮点数据类型,关于这个问题,我尚没有良好的对策,因为引入新的基础数据类西就需要引入新的基础运算.然而针对数据包的基础运算有哪些?怎么组合,这些问题我还不能很好的回答.

2.2. 未确定的数据类型

`UndeterminedType`用于表示未确定的数据类型,其包含两个子类型`UnknownType`和`NamedType`.



- 命名的数据类型

  `NamedType`用于表示在源码中通过书写类型名称构成类型表达式的形式,此时编译器尚不能确定源码中书写的名称是否真的代表一个存在的且可用的数据类型,所以NamedType是Undetermined类型.

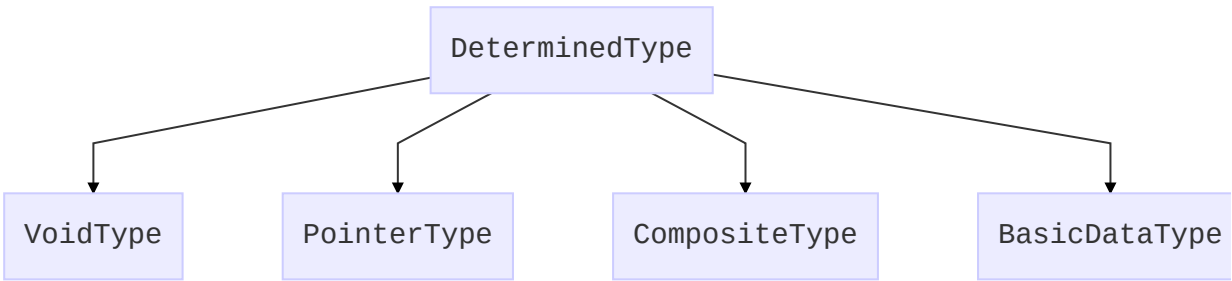
  当编译器搜集了足够多的信息,命名数据类型会被转化成相应的数据类型.

- 未知的数据类型

  未知的数据类型表示源码中并未明确指定数据类型,编译器需要通过上下文来确定数据类型,比如对于一个带有构造表达式的元素构造语句,若不指定数据类型,编译器认为构造表达式的结果类型就是元素的数据类型.

2.3. 确定的数据类型

确定的数据类型在使用的过程中基本上不需要再访问语法树来搜集其他信息了.



- 空类型

  空类型在特殊的情况下用作类型的占位符,比如当方法不拥有返回值时,返回值类型需要使用空类型占位,因为如果不书写返回值类型,其表达的语意是方法可以返回任何类型的数据.

- 指针类型

  指针元素只能绑定指针对象,指针类型为这样的限制提供了判断条件的便利.

- 复合类型

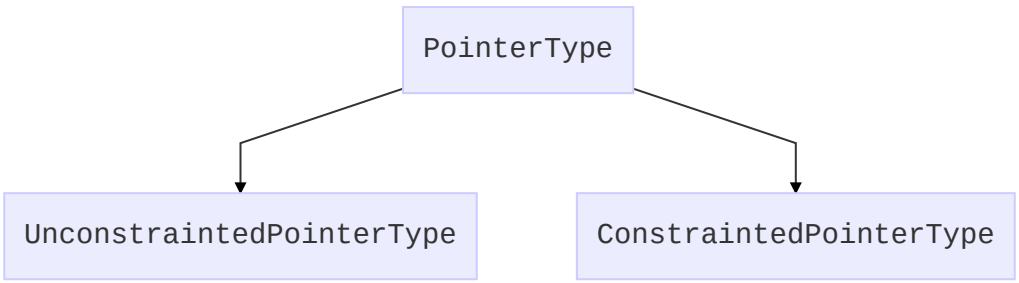
  通过书写类定义产生的数据类型被称为复合数据类型,其中包括类的元数据类型和类的实例数据类型.复合数据类型要绑定一个类定义,这其中就要涉及继承,抽象,模板类等细节问题了,此处在泛型类定义文档中会得到更详细的讨论.

- 基础数据类型

  Alioth仅提供了共11种基础数据类型以供直接使用或组合成复合数据类型.基础数据类型之间的运算规则和转换规则是硬编码的,所以当运算符所关联的两个运算子的数据类型都是基础数据类型时,要进行特别的换算操作.

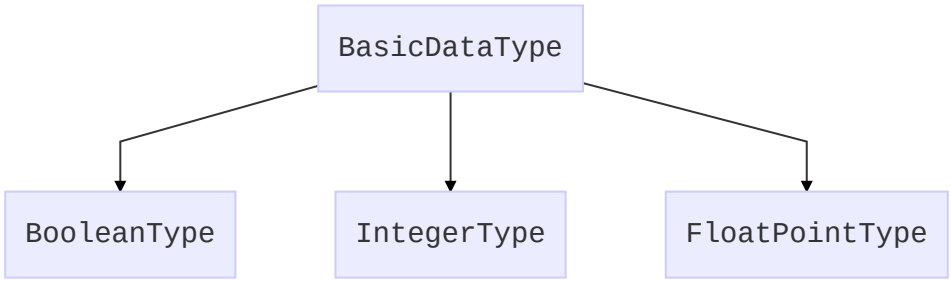
2.4. 指针数据类型

指针数据类型有一个关于写权限约束的属性,这个属性只属于指针数据类型,为了优化设计,我们将其设计为指针数据类型的两种子类.



2.5. 基础数据类型

基础数据类型大致上可以分为三类,整数,浮点数和布尔类型.



- 布尔类型

  布尔类型`BooleanType`是唯一能被用作条件来引导条件分支语句或循环语句的数据类型.

- 整数类型

  整数类型`IntegerType`共包含8个具体的数据类型,他们之间的自动转换关系遵循着绝对不丢失精确度的原则.

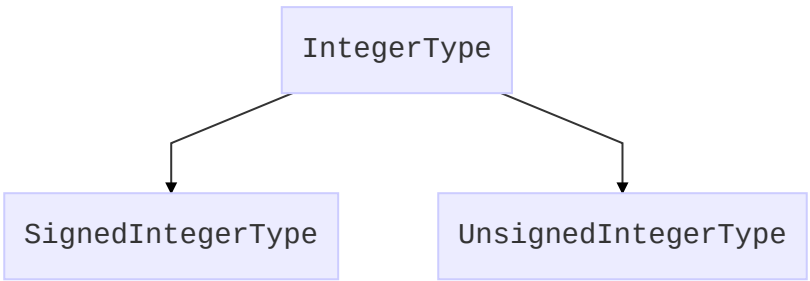
- 浮点数类型

  浮点数类型`FloatPointType`包含能用于进行浮点运算的数据类型.

2.6. 整数类型

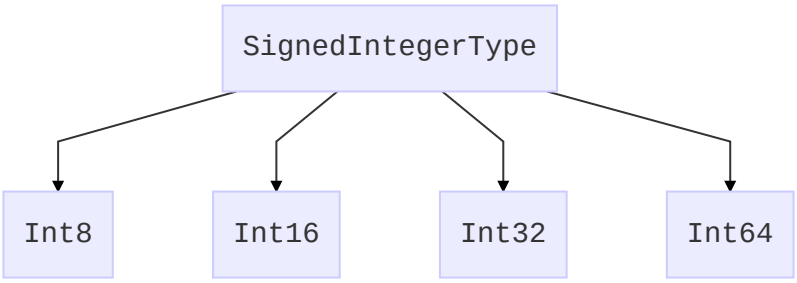
整体上,整数数据类型分为带符号位和不带符号位两大类,在这之下,包含4中长度的数据类型.

有无符号位影响着一个数据类型被转换和参与运算时,默认情况下是否考虑符号位.



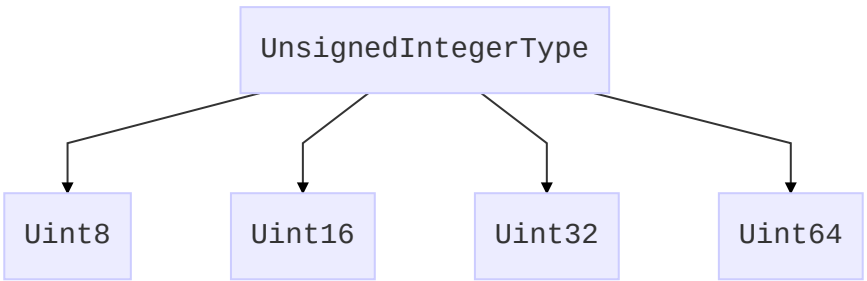
2.7. 带符号位整数类型

带符号位整数类型在扩展时带符号位扩展,计算乘除余时带符号位计算.



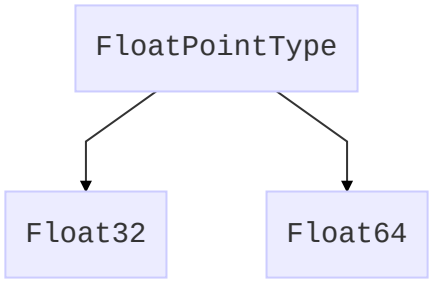
2.8. 不带符号位整数类型

不带符号位整数类型的运算完全不考虑符号位.



2.9. 浮点数类型

浮点数类型包含了两种常用长度的浮点数类型,然而我们还没办法考虑更长的浮点数类型.



3. 数据类型转换

本章讨论关于数据类型转换的规则.在Alioth中,数据类型依附于元素原型而存在,在此我们暂时只讨论纯粹的数据类型转换.在数据类型转换同时发生元素类型转换的情况我们会单独拿出一章进行全面的讨论.

数据类型的转换动作大致上分为强制类型转换和自动类型转换,无论是何种类型转换都区别于强制类型视角.

3.1. 强制类型视角

在Alioth中使用`as!`引导强制类型对待语句,强制Alioth将一个对象视为一种目标数据类型的对象进行操作.

这个过程类似于如下的**CPP**语言语句.

```
(*(TargetType*)&Obj);
```

强制类型视角对类型之间的关系没有任何要求.

3.2. 自动类型转换

自动类型转换发生在两个相容数据类型的对象之间进行匹配时,如传参,运算,返回.

> 此处讨论的运算是指不调用任何运算符重载而进行的基础运算.
>
> 在任何运算中,运算符左侧的运算子都是主运算子,即使右结合性运算符也是这样.
>
> 编译器总是先尝试将从运算子转化为主运算子类型再运算.

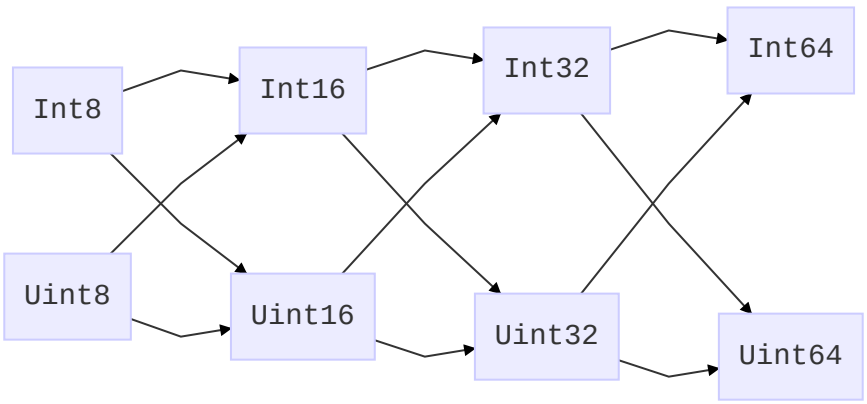
自动类型转换要讨论基础数据类型、指针数据类型、复合数据类型和空类型之间之间的转换关系.

显然空类型不能与任何其他数据类型进行转换,在其他数据类型中的子类型若粒度涉及时,会展开讨论.

下述类型转换关系都是传递的.

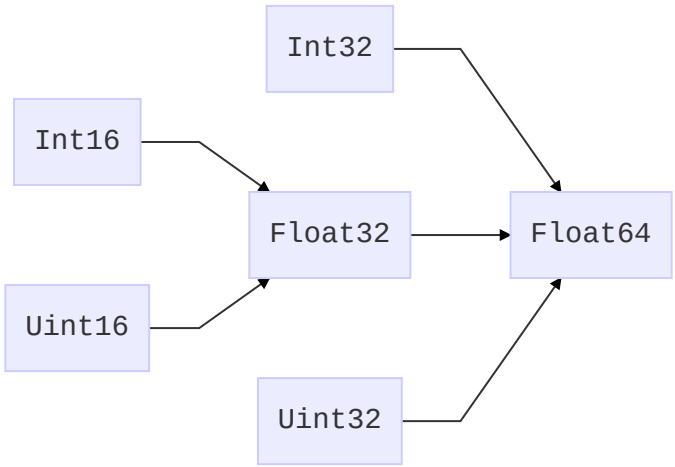
3.2.1. 规则一: 扩展无代价

如果从一个数据类型转换到另一个数据类型,需要扩展数据的宽度,无论如何都认为是安全的.



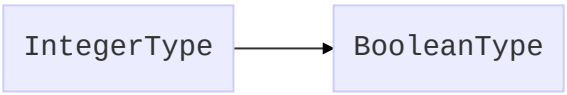
3.2.2. 规则二: 目标为浮点数时绝不牺牲精度

如果目标是一个浮点数,一定要确保绝对不损失一丁点精确度.



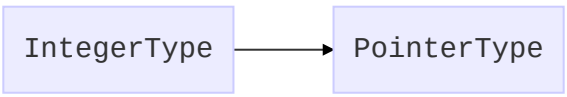
3.2.3. 规则三:整数类型与布尔

布尔类型不能转化为任何整数类型,但是整数类型可以转化为布尔类型



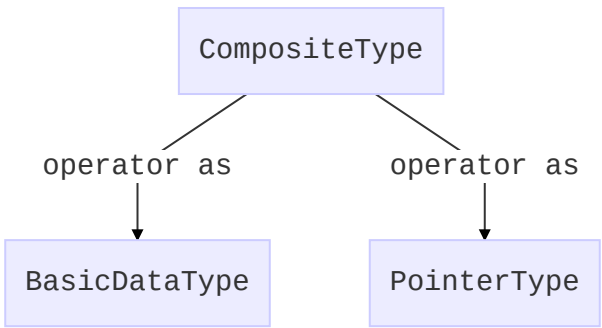
3.2.4. 规则四:指针类型与整数

指针不能自动转换为任何数字类型,但是整数类型可以转换位指针类型

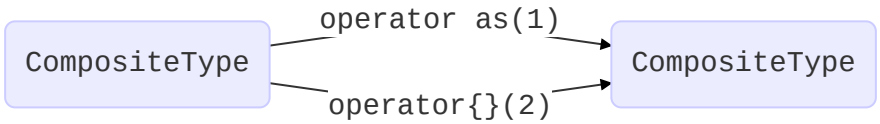


3.2.5. 规则五:复合数据类型

复合数据类型与其他数据类型之间的转换依靠类型转换运算符重载和构造运算符.这两种转换都被计作有成本的类型转换,成本用于在复杂的转换图中挑选一条成本最低的转换路径.



复合数据类型与复合数据类型之间的转换,优选考虑运算符重载,再考虑构造方法



4. 情景讨论

数据类型或元素类型的变化,发生在传参,返回或运算时,本章节分别从不同的情况入手,仔细讨论其中的差别.

4.1. 传参时

4.1.1. VAR-VAR

通过类型转换创造新的对象.

4.1.2. VAR-REF

类型转换过程不能创造一个新的对象.

4.2. 返回时

4.3. 运算时