Несколько слов о reference и value объектах. Основное семантическое (смысловое, поведенческое) различие проявляется в том, как работает присваивание. Возможны 4 варианта

1. reference1 := reference2 // присваивается один ссылочный объект другому
2. reference := value // присваивается объект значение объекту ссылка
3. value1 := value2 // присваивается один объект значение другому
4. value := reference // присваивается ссылочный объект объекту значение

При этом все тип объекта в правой части присваивания конформен типу объекта в левой.

1. reference1 := reference2 // присваивается один ссылочный объект другому
   1. после выполнения такого присваивания на один и тот же объект указывают две ссылки
2. reference := value // присваивается объект значение объекту ссылка
   1. создаётся копия value объекта и ссылка на нее размещается в reference
3. value1 := value2 // присваивается один объект значение другому
   1. в каждое поле (member attribute) value1 копируется содержимое соответствующего поля из value2
4. value := reference // присваивается ссылочный объект объекту значение
   1. в каждое поле (member attribute) value копируется содержимое соответствующего поля из объекта, на который указывает ссылка reference

Аналогичное правило действует и для передачи параметров подпрограммам. Пример с комментариями ниже.

reference **is ref** A

value **is val** A

foo (reference) // просто передаётся ссылка на объект (reference) как параметр процедуры

foo (value) // создаётся копия value и ссылка на неё передаётся процедуре

foo (**ref** value) // создаётся ссылка на объект value и эта ссылка передаётся, как параметр процедуры

goo (reference) /\* в каждое поле нового объекта значения типа А прописывается соответствующее значение поля из того объекта, на который ссылается reference и этот новый объект и будет параметром goo \*/

goo (value) /\* в каждое поле нового объекта значения типа А прописывается соответствующее значение поля из объекта value и этот новый объект и будет параметром goo \*/

foo (argument: **ref** A) **do** … **end**

goo (argument: **val** A) **do** … **end**

* Ограничение на использование **ref** <expression>. Такая конструкция валидна если тип выражения <expression> является типом значением (value)
* Небольшой комментарий про перегрузку имён. Нельзя перегружать имена подпрограмм, если их сигнатуры различаются лишь видом передаваемых параметров. Т.е. нельзя делать перегрузку вида

foo (argument: **ref** A) **do** … **end**

foo (argument: **val** A) **do** … **end**

Другими словами **ref** A и **val** A – это все равно один и тот же тип А.

А теперь ещё пример о параметрах подпрограмм. Пусть некоторая процедура описана следующим образом

foo (arg: **ref** Integer) **do**

arg := arg + 1

**end**

x1: **val** Integer is 5

x2: **ref** Integer is 5

// очевидно, что значения x1 и x2 совпадают и равны 5

foo (x1)

foo (x2)

// А чему тут равны x1 и x2?

На этот вопрос сразу нет ясного ответа. Поэтому самое простое решение — это запретить присваивание в параметры подпрограмм внутри тел подпрограмм. Т.е. можно рассматривать параметры любой подпрограммы как локальные сущности, описанные с префиксом const. Примерно так

foo (\_1) **do** // здесь \_1 означает, что foo надо вызывать с не менее чем одним параметром

**const** arg: **ref** Integer **is** \_1 // а это обращение к первому параметру

arg := arg + 1 // И тогда это ошибка времени компиляции

**end**

И есть ещё один вариант вызова

foo (**ref** x1)

но семантически он эквивалентен foo (х2)

Таким образом все параметры передаются по значению и семантика параметров задана в сигнатуре подпрограммы явно через префикс ref или val перед типом или при отсутствии префикса будет тот тип параметра как указано в теле юнита, который задаёт тип параметра.