Automatically Deriving Test Data for Julia Functions

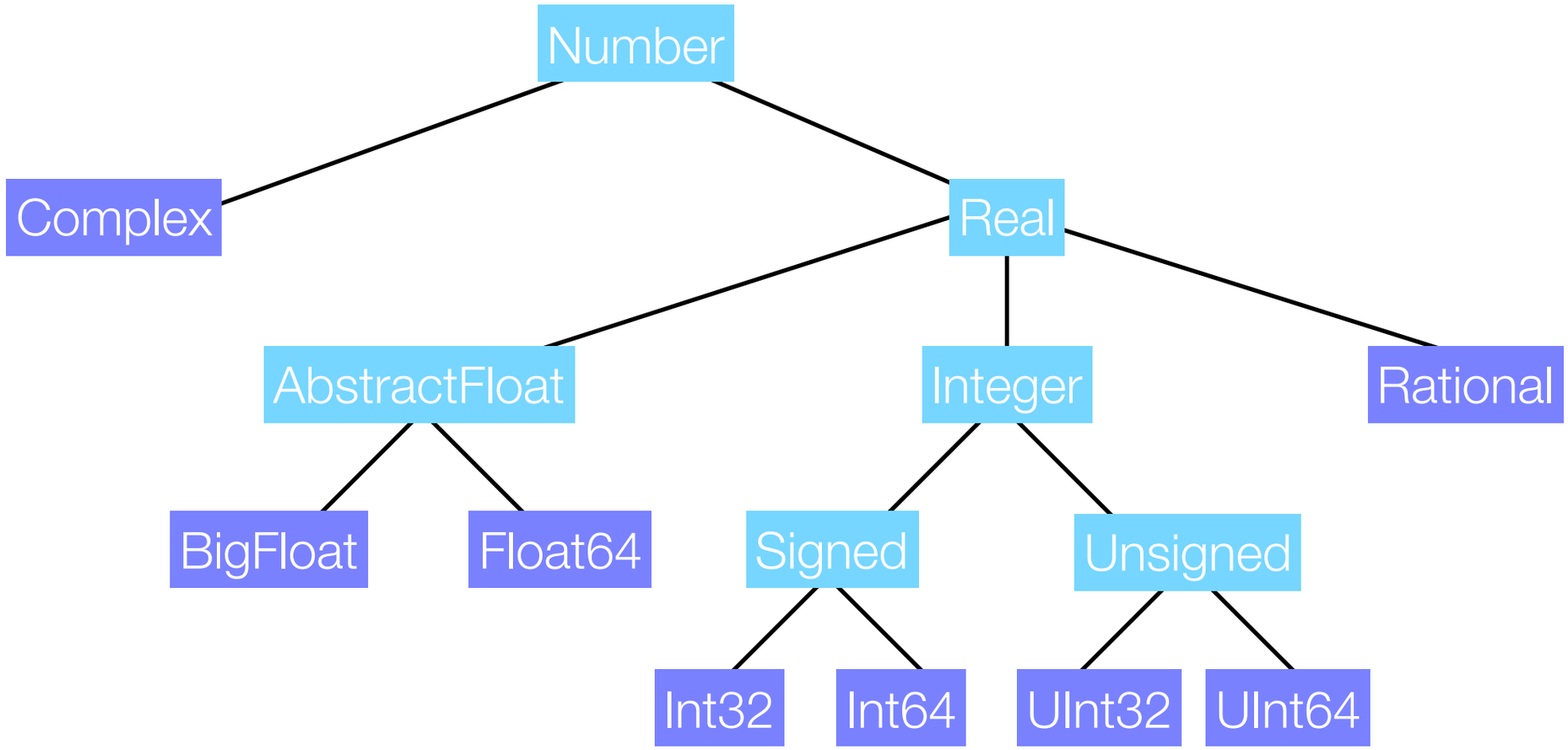
Simon Poulding and Robert Feldt
Blekinge Institute of Technology, Sweden

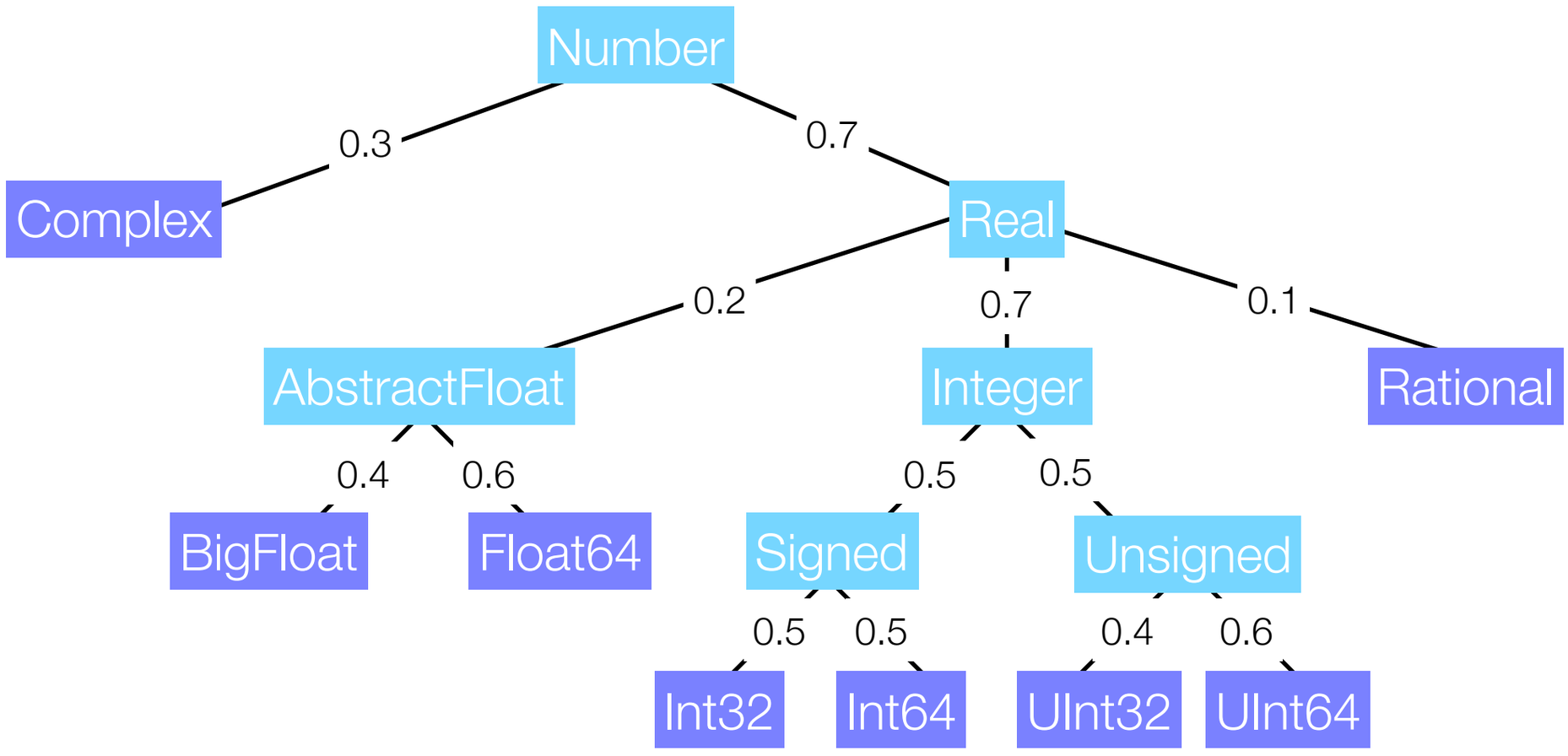
Typically …

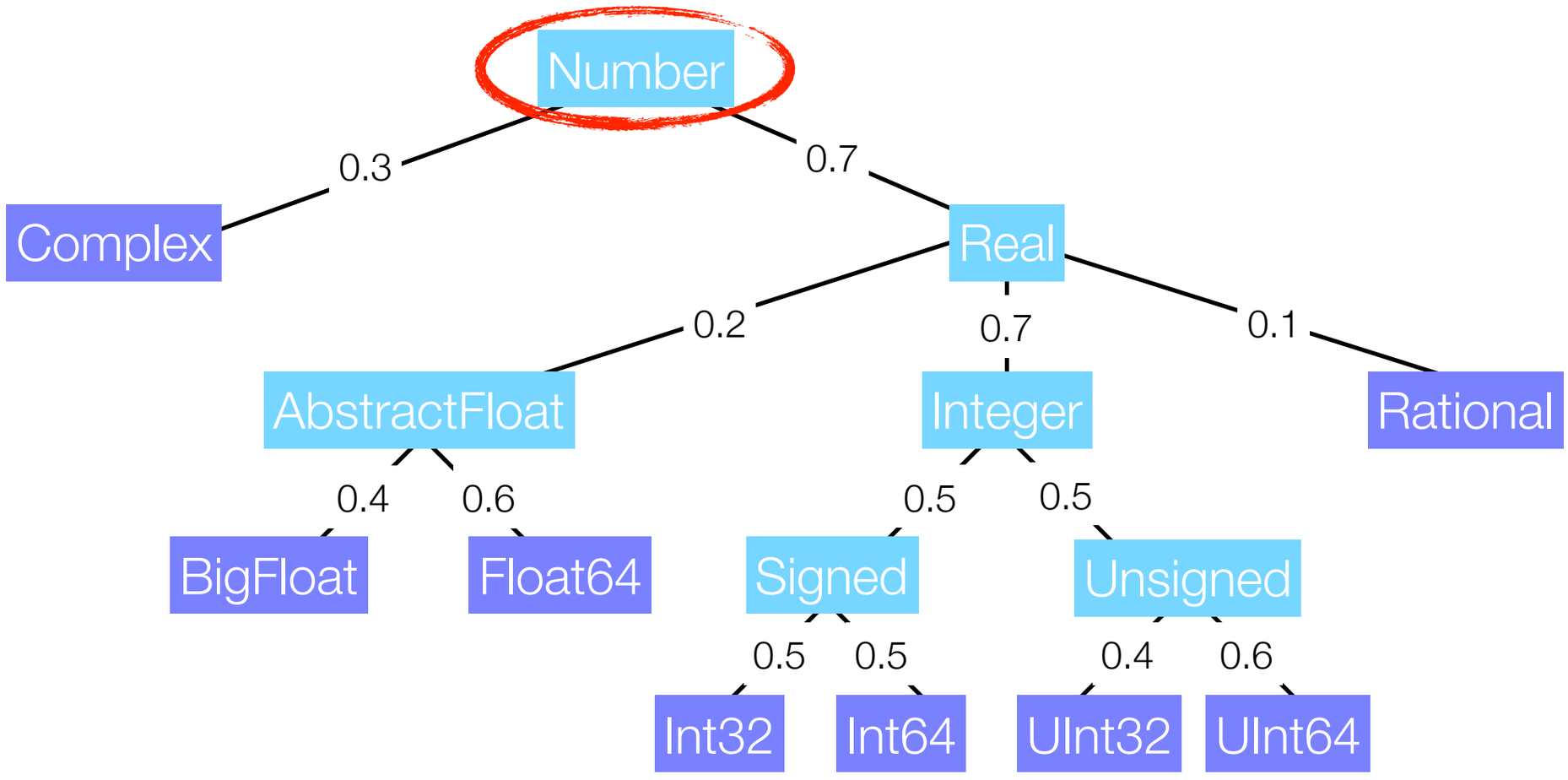
random testing: inputs are diverse **values**

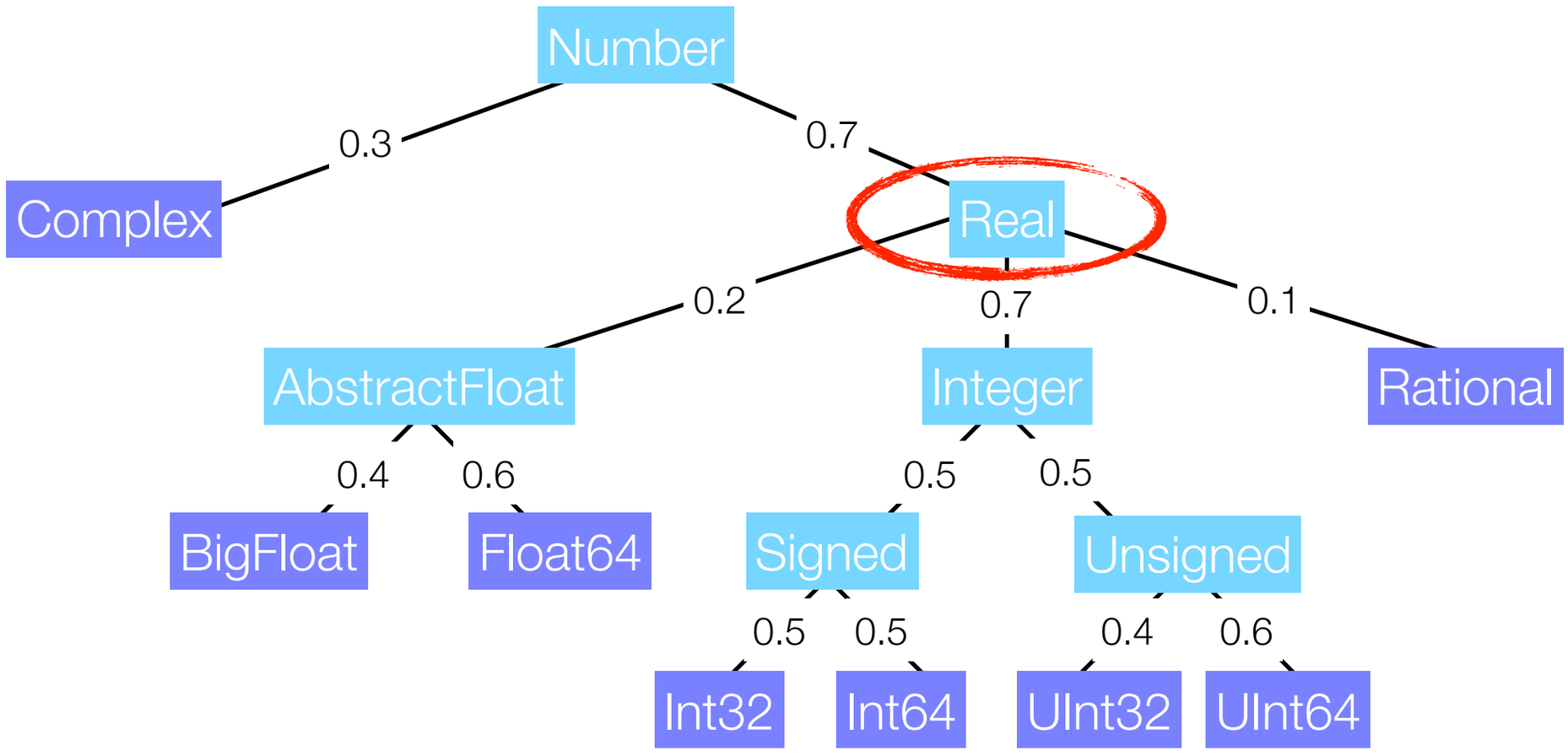
Multiple Dispatch …

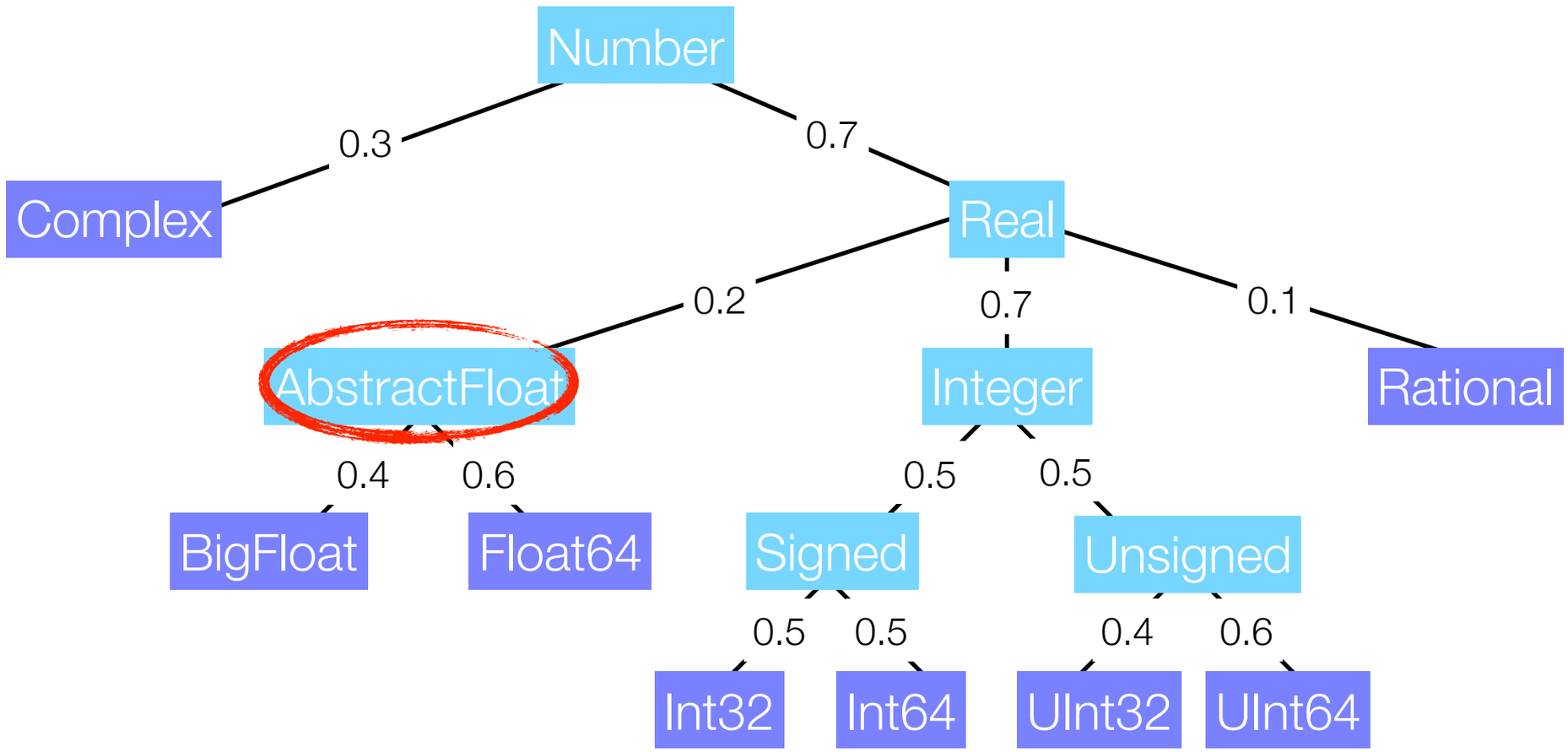
random testing: inputs are diverse **values and types**

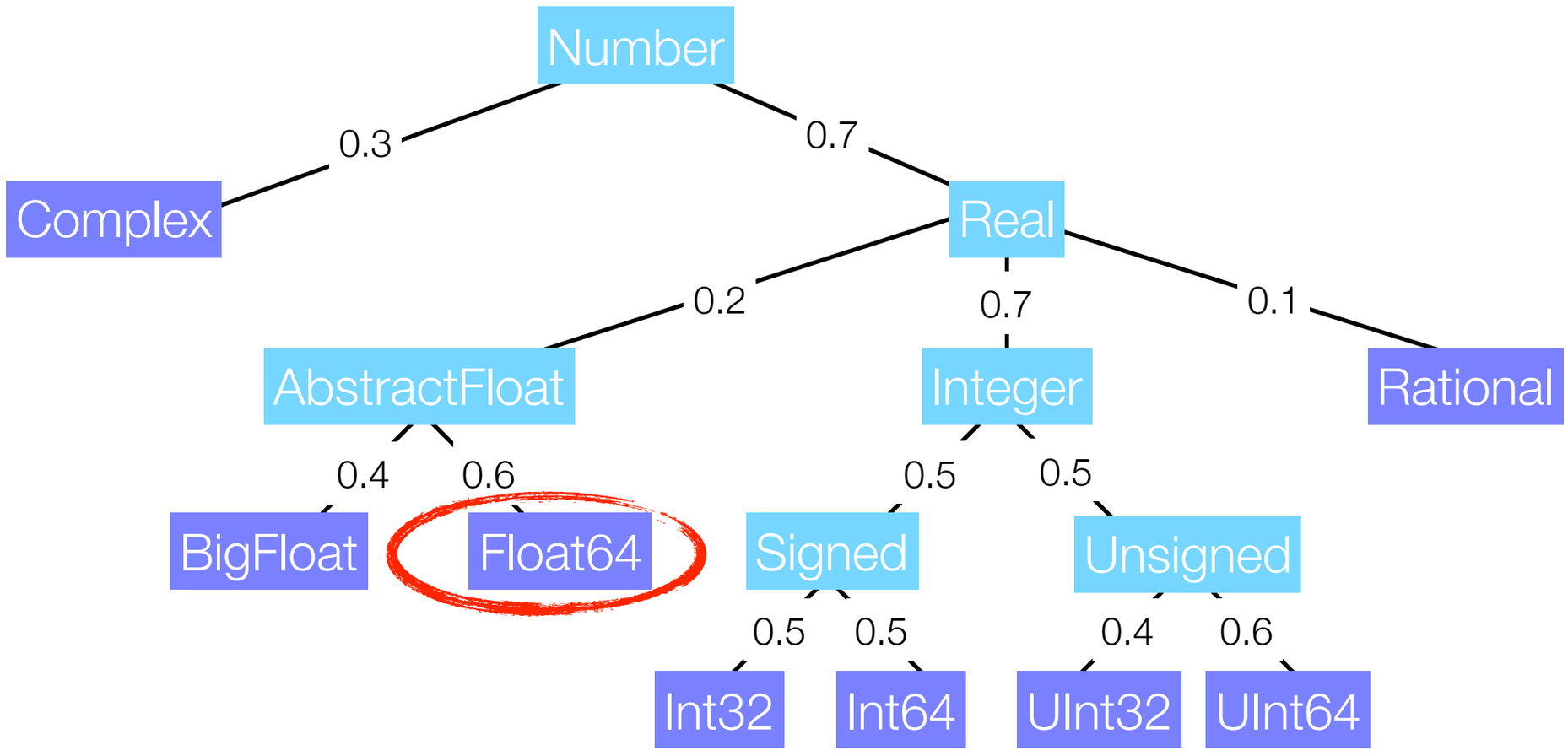

Number
Complex
Real
AbstractFloat
Integer
Rational
BigFloat
Float64
Signed
Unsigned
Int32
Int64
UInt32
UInt64

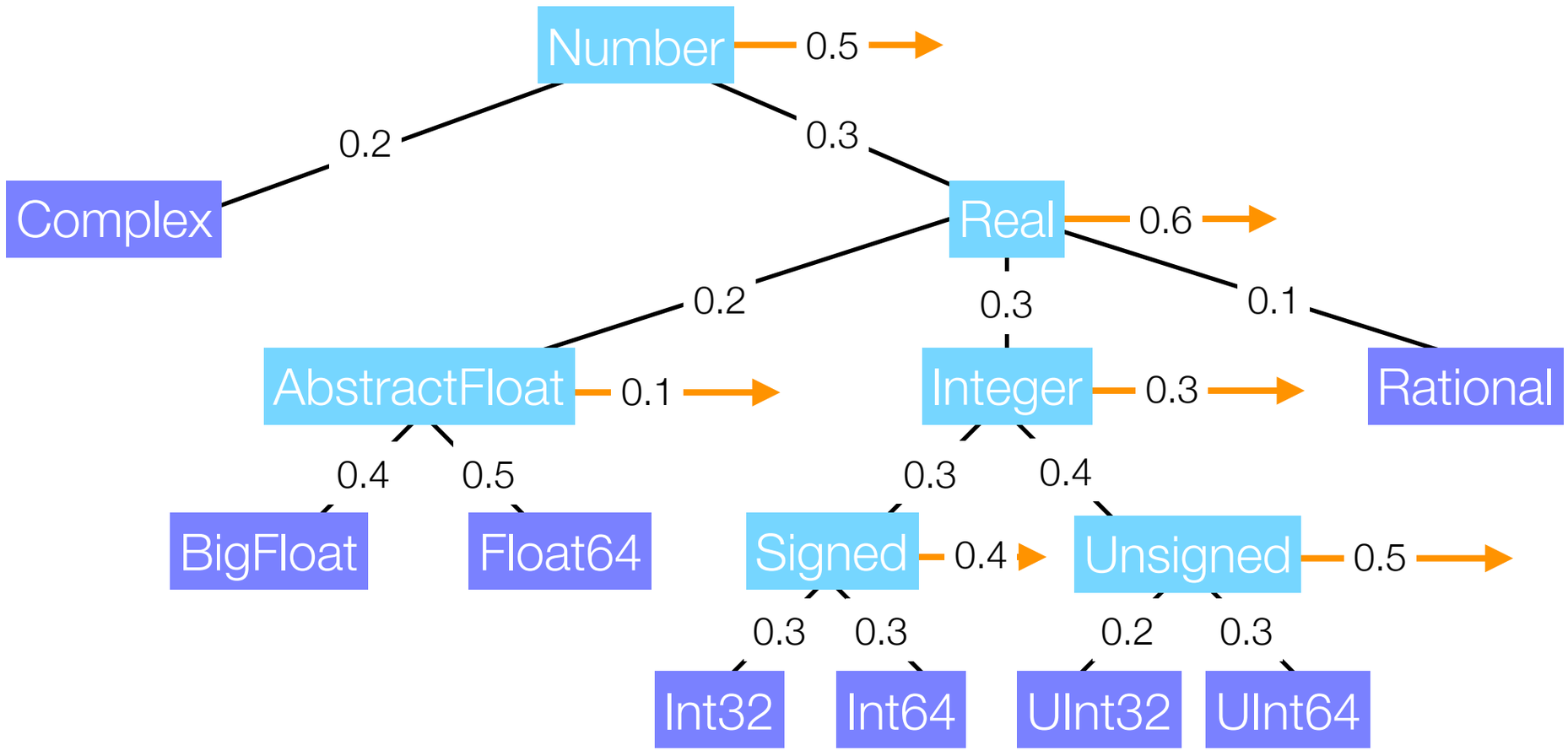

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

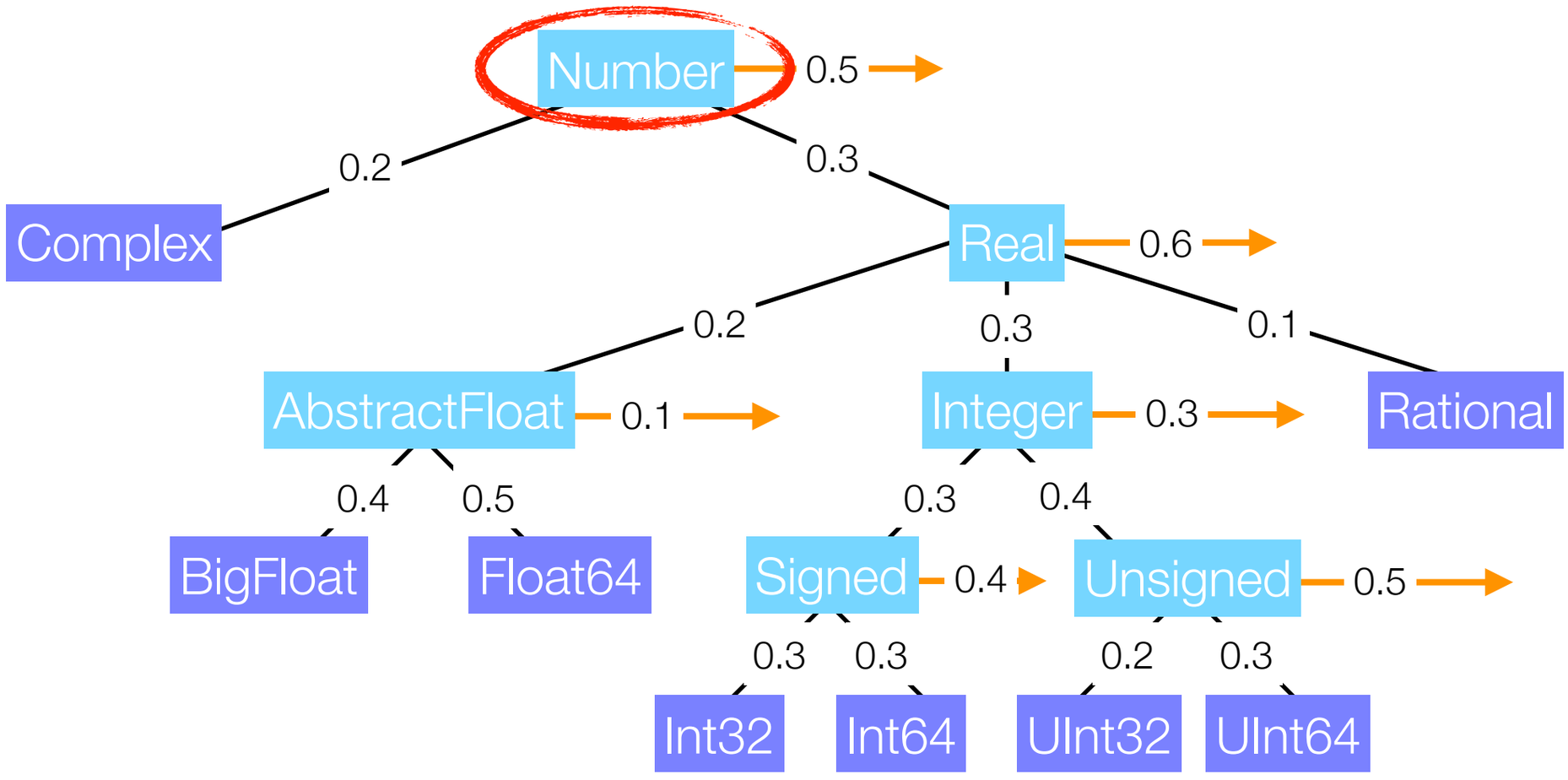

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

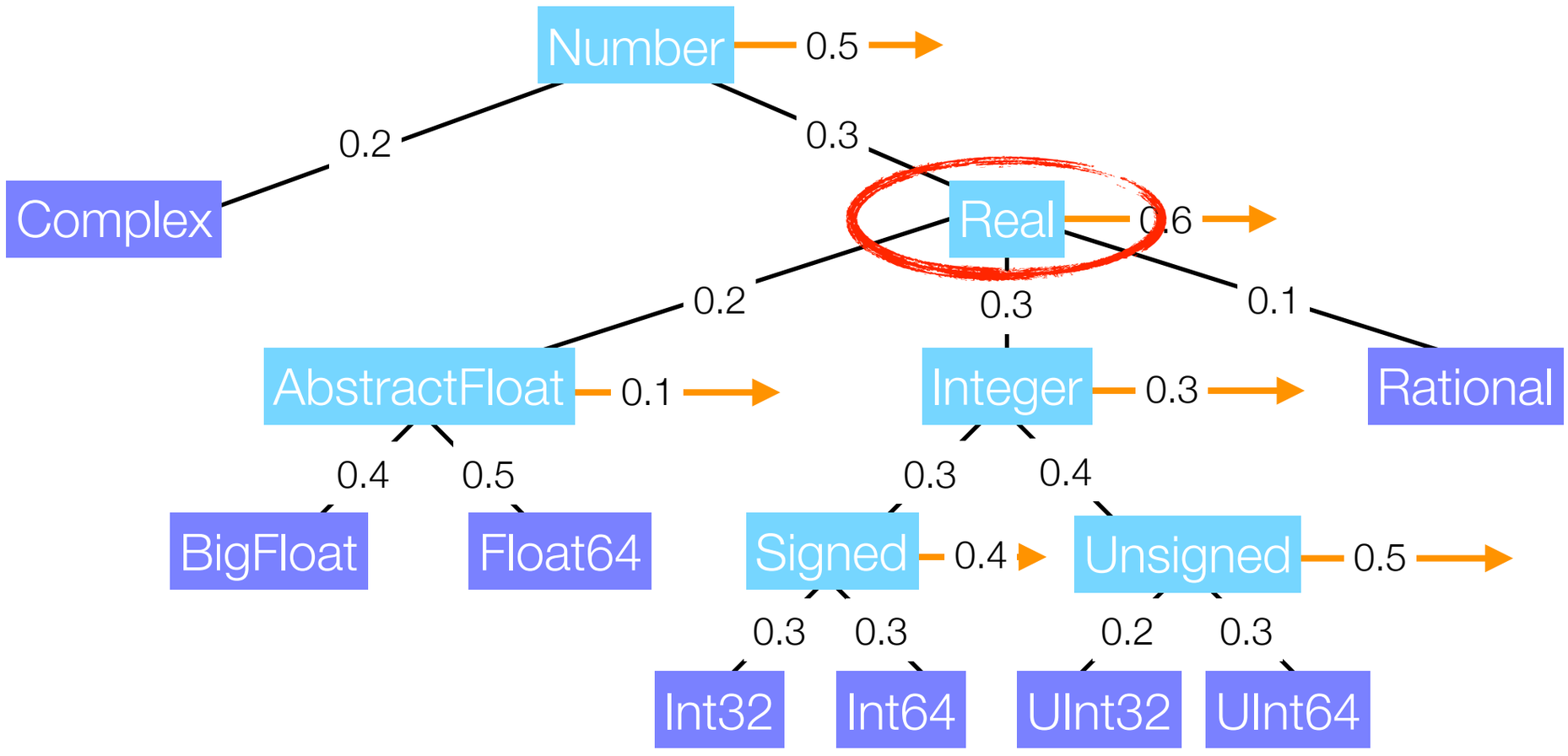

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

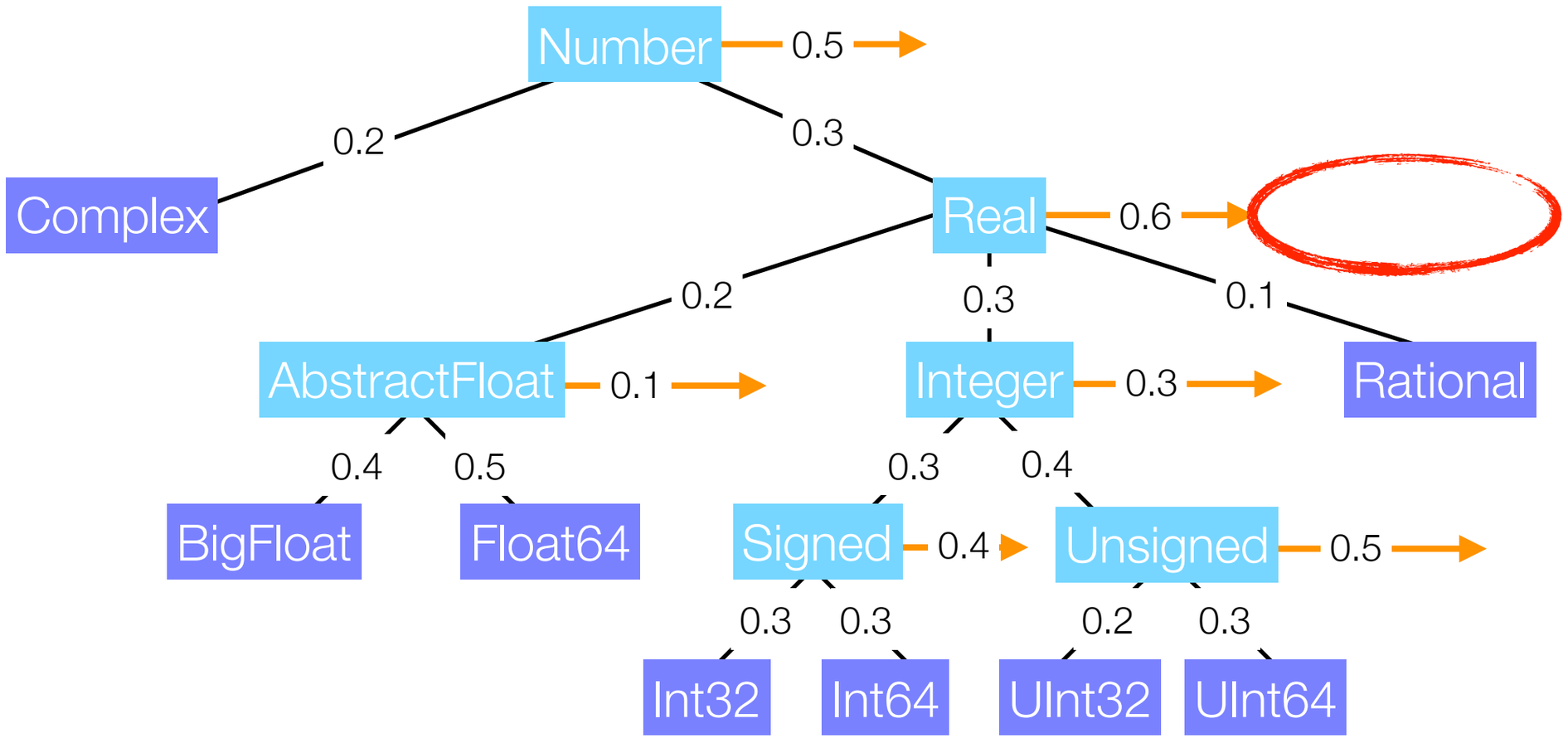

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

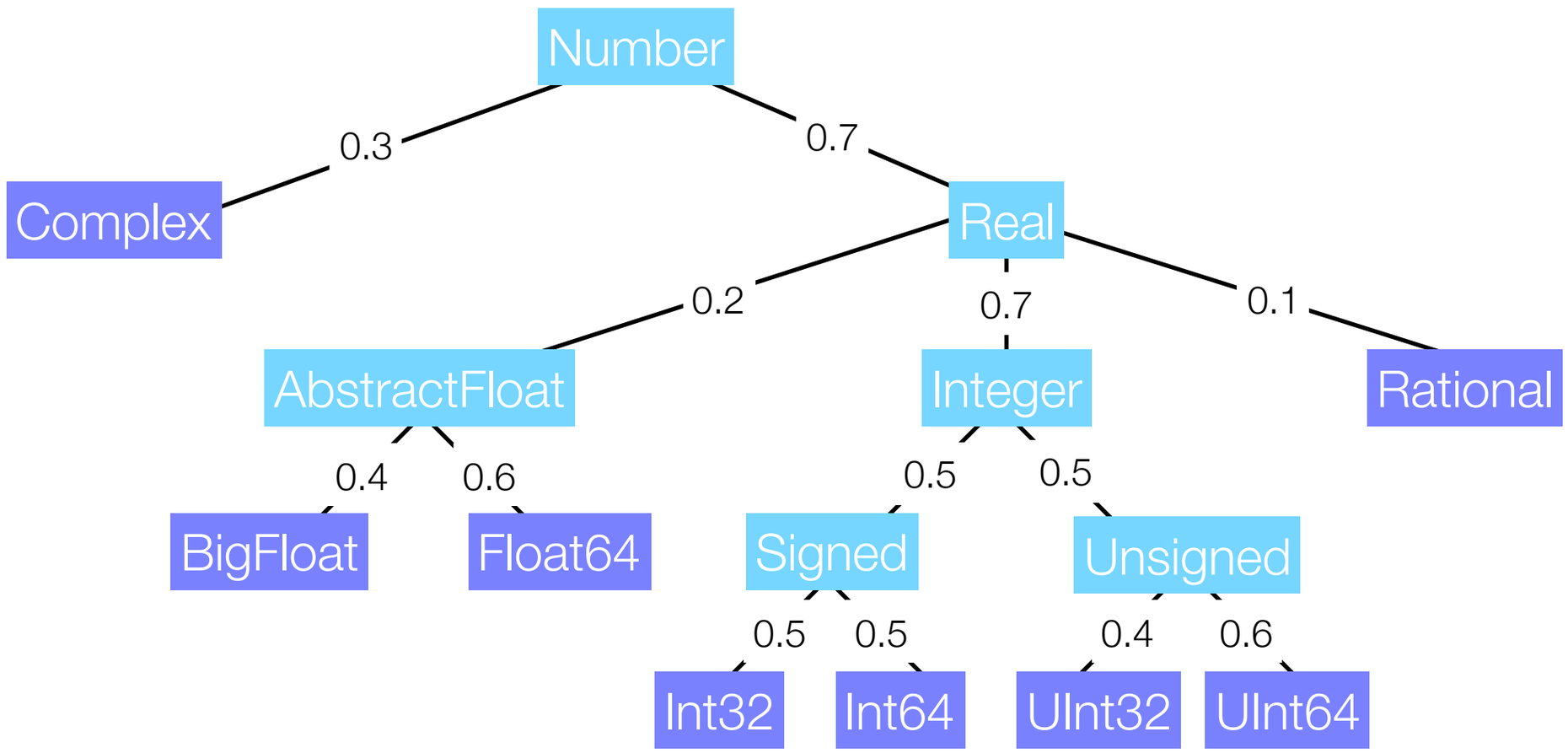

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

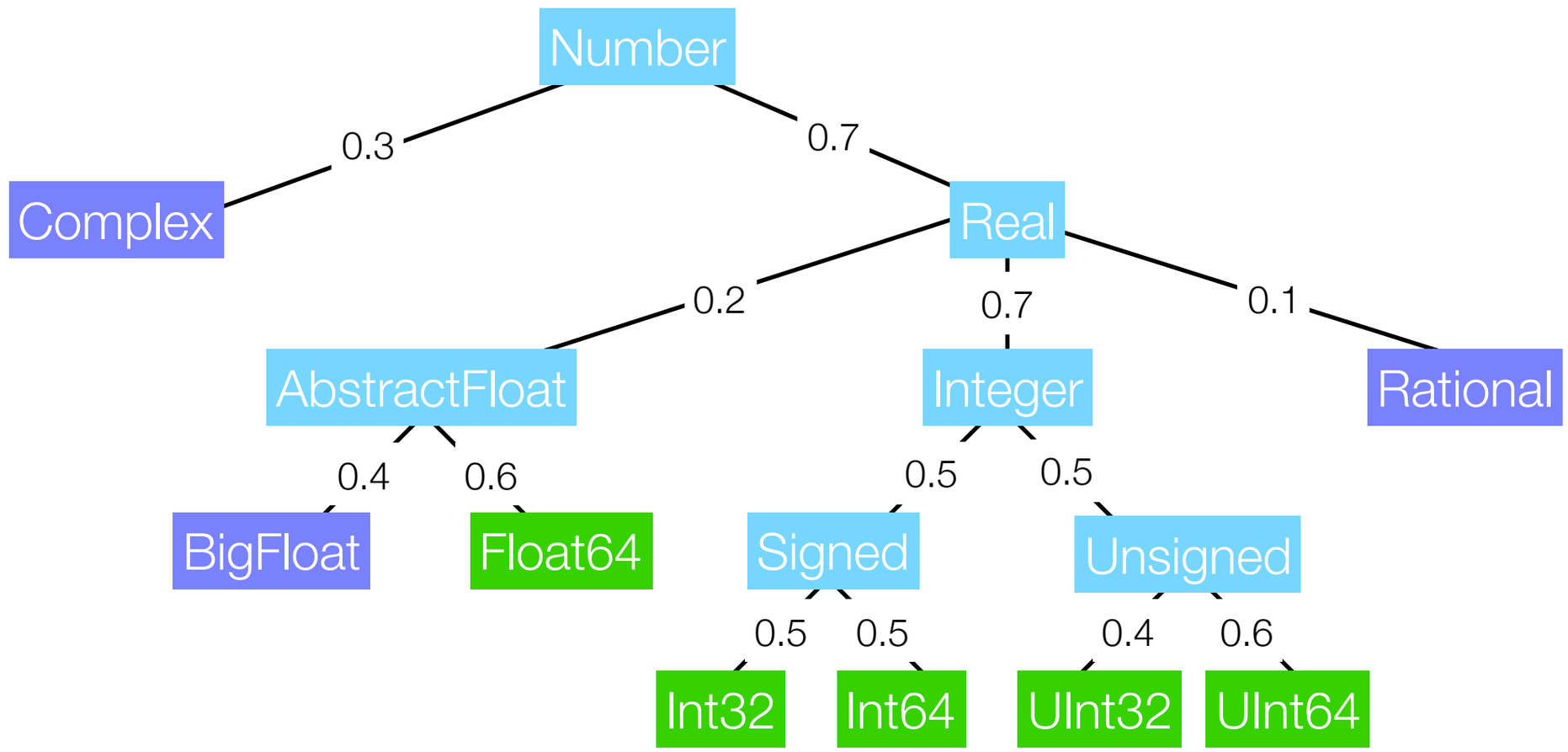

Number
0.5
0.2
0.3
Complex
Real
0.6
0.2
0.3
0.1
AbstractFloat
0.1
Integer
0.3
Rational
0.4
0.5
0.3
0.4
BigFloat
Float64
Signed
0.4
Unsigned
0.5
0.3
0.3
0.2
0.3
Int32
Int64
UInt32
UInt64

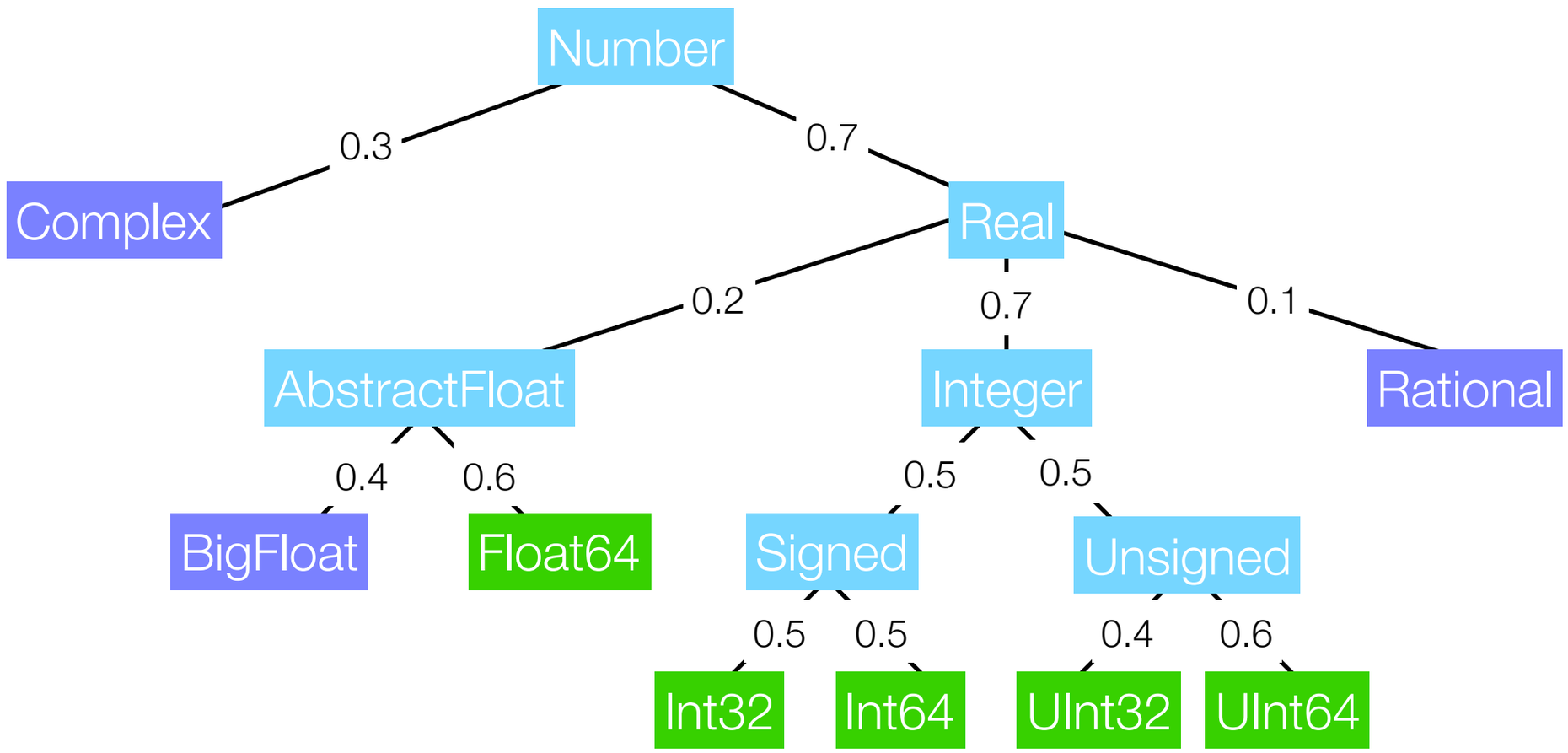

Number
0.5
0.2
0.3
Complex
Real
0.6
0.2
0.3
0.1
AbstractFloat
0.1
Integer
0.3
Rational
0.4
0.5
0.3
0.4
BigFloat
Float64
Signed
0.4
Unsigned
0.5
0.3
0.3
0.2
0.3
Int32
Int64
UInt32
UInt64


Number
0.5
0.2
0.3
Complex
Real
0.6
0.2
0.3
0.1
AbstractFloat
0.1
Integer
0.3
Rational
0.4
0.5
0.3
0.4
BigFloat
Float64
Signed
0.4
Unsigned
0.5
0.3
0.3
0.2
0.3
Int32
Int64
UInt32
UInt64

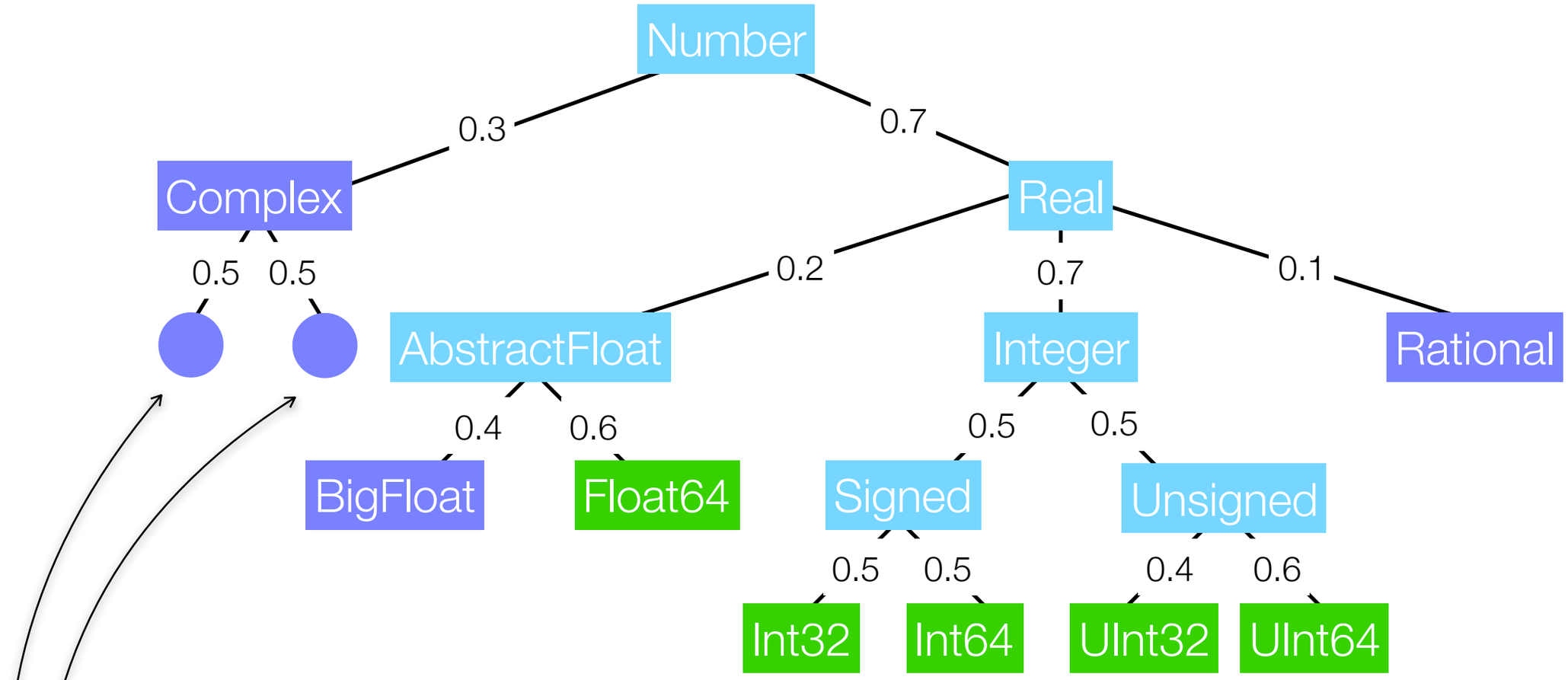

Number
0.5
0.2
0.3
Complex
Real
0.6
0.2
0.3
0.1
AbstractFloat
0.1
Integer
0.3
Rational
0.4
0.5
0.3
0.4
BigFloat
Float64
Signed
0.4
Unsigned
0.5
0.3
0.3
0.2
0.3
Int32
Int64
UInt32
UInt64


Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64

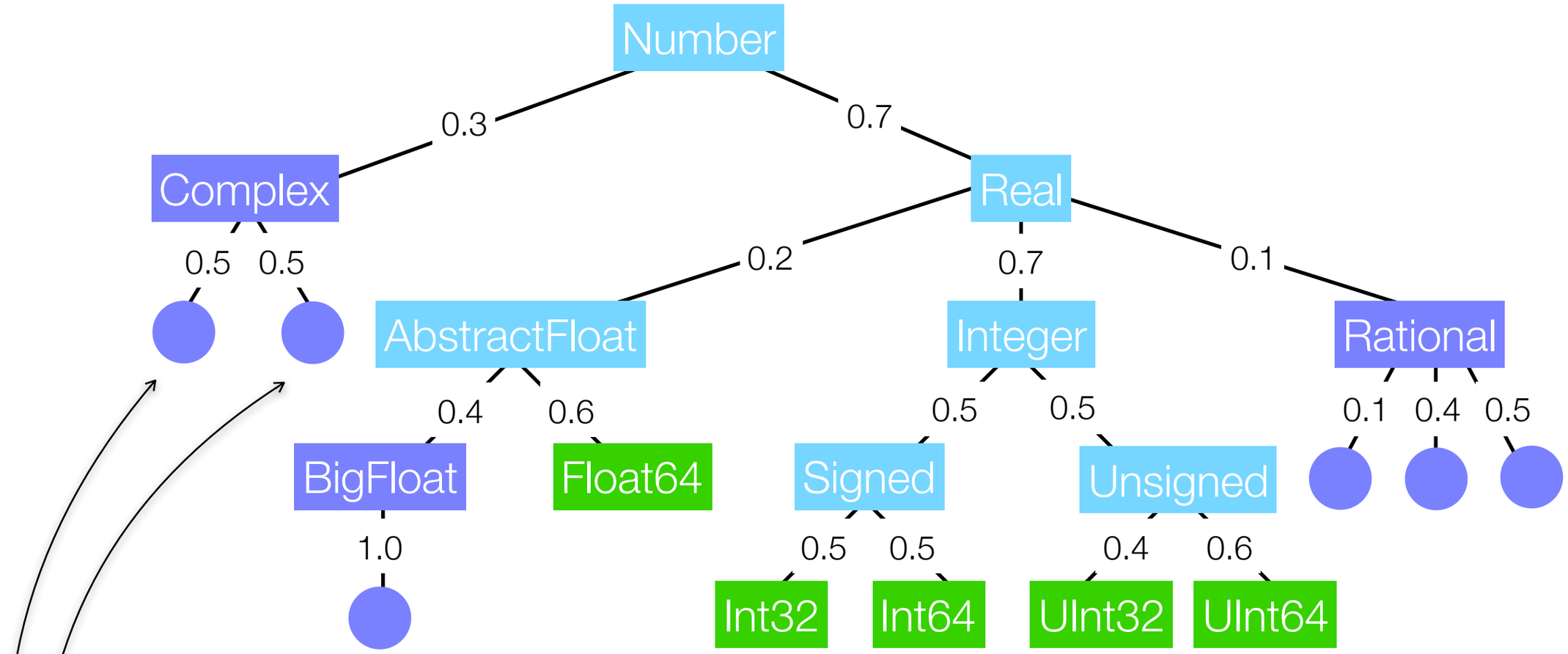

Number
0.3
0.7
Complex
Real
0.2
0.7
0.1
AbstractFloat
Integer
Rational
0.4
0.6
0.5
0.5
BigFloat
Float64
Signed
Unsigned
0.5
0.5
0.4
0.6
Int32
Int64
UInt32
UInt64




```
Complex(x::Real)

Complex(re::Real, im::Real)
```




```
Complex(x::Real)

Complex(re::Real, im::Real)
```




```
Complex(x::Real)

Complex(re::Real, im::Real)
```

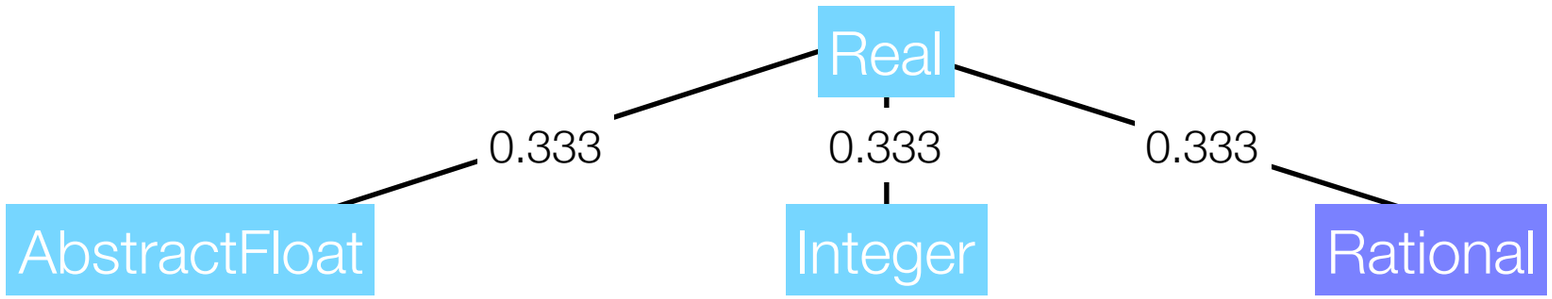
```
using DataGeneratorTranslators
using DataGenerators

type_generator(:NumberGen, Number)

for i in 1:20
    try
        x = choose(NumberGen)
        dump(x)
    catch _e
        if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
            rethrow(_e)
        end
        warn("exception raised in constructor method")
    end
end
```

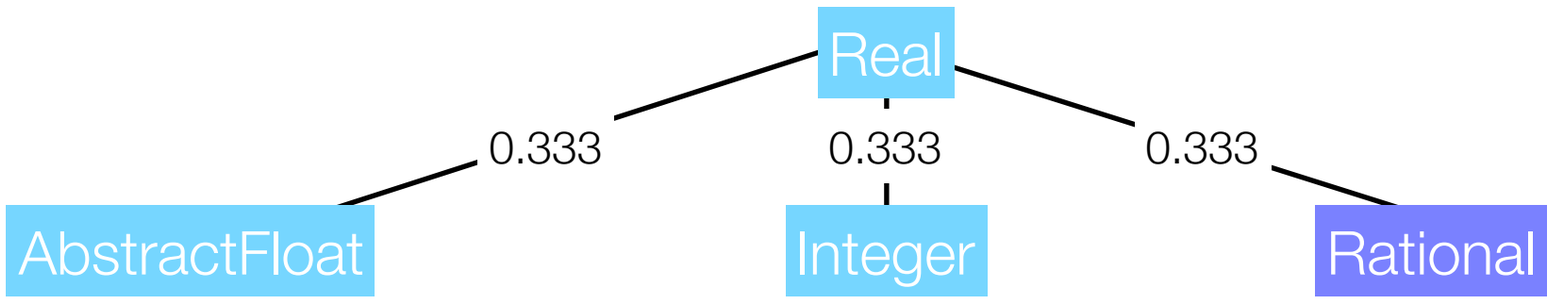
```
using DataGeneratorTranslators
using DataGenerators

type_generator(:NumberGen, Number)

for i in 1:20
    try
        x = choose(NumberGen)
        dump(x)
    catch _e
        if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
            rethrow(_e)
        end
        warn("exception raised in constructor method")
    end
end
```

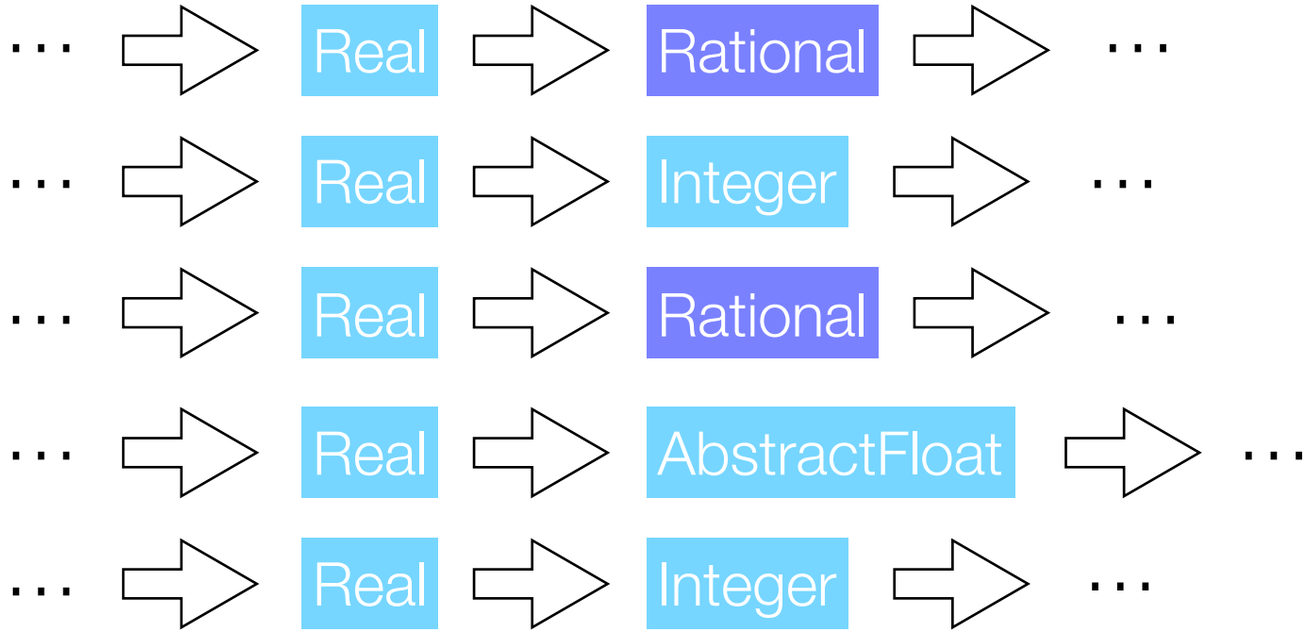
```
using DataGeneratorTranslators
using DataGenerators

type_generator(:NumberGen, Number)

for i in 1:20
    try
        x = choose(NumberGen)
        dump(x)
    catch _e
        if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
            rethrow(_e)
        end
        warn("exception raised in constructor method")
    end
end
```

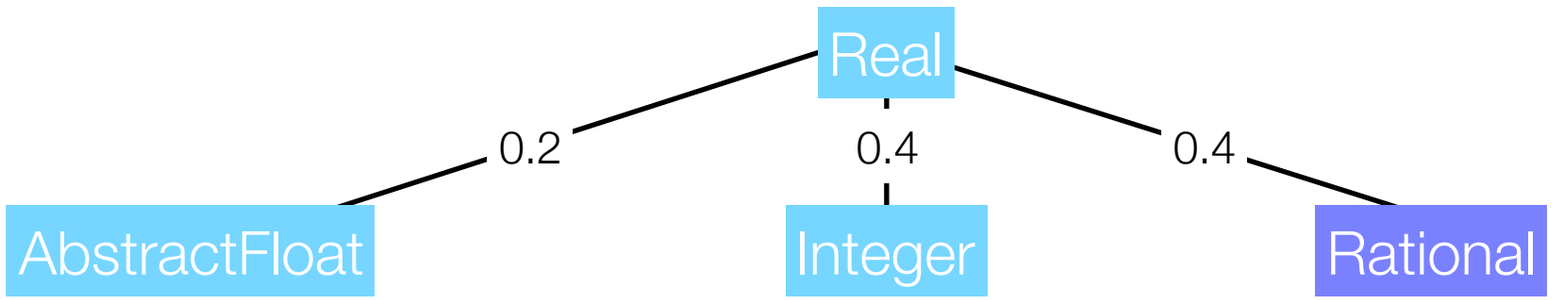
```
using DataGeneratorTranslators
using DataGenerators

type_generator(:NumberGen, Number)

for i in 1:20
    try
        x = choose(NumberGen)
        dump(x)
    catch _e
        if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
            rethrow(_e)
        end
        warn("exception raised in constructor method")
    end
end
```

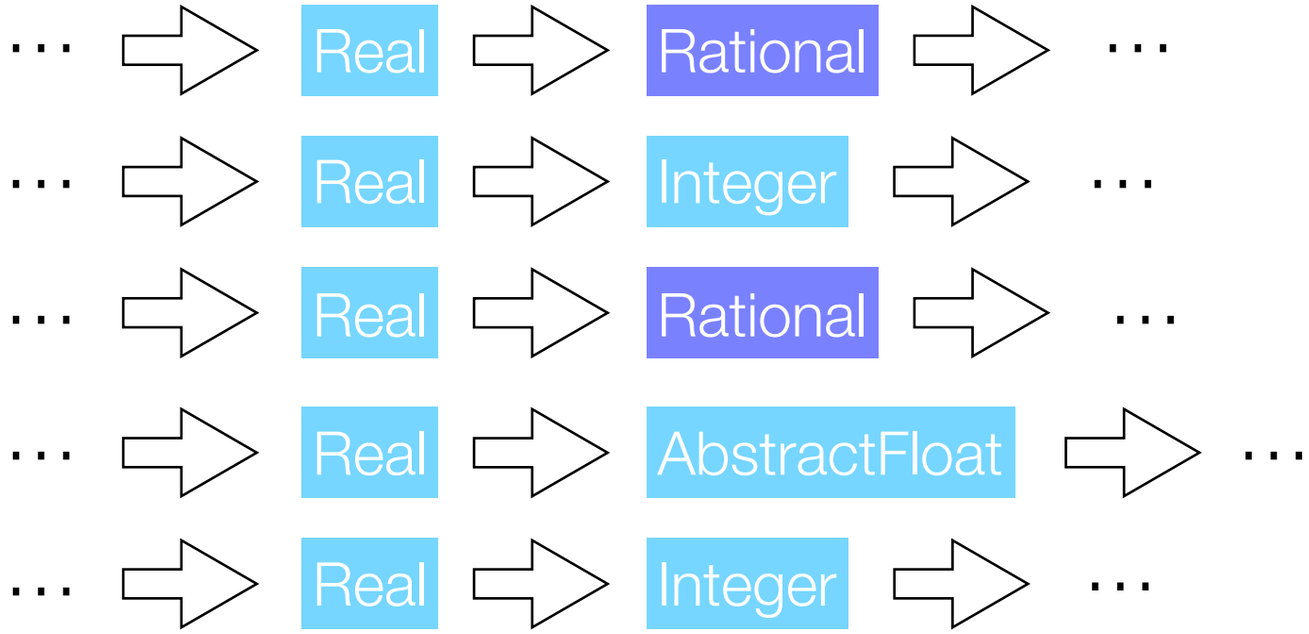
```
using DataGeneratorTranslators
using DataGenerators

type_generator(:NumberGen, Number)

for i in 1:20
    try
        x = choose(NumberGen)
        dump(x)
    catch _e
        if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
            rethrow(_e)
        end
        warn("exception raised in constructor method")
    end
end
```


Real
0.333
0.333
0.333
AbstractFloat
Integer
Rational


Real
0.333
0.333
0.333
AbstractFloat
Integer
Rational


… Real Rational …
… Real Integer …
… Real Rational …
… Real AbstractFloat …
… Real Integer …


Real
0.2
0.4
0.4
AbstractFloat
Integer
Rational


… Real Rational …
… Real Integer …
… Real Rational …
… Real AbstractFloat …
… Real Integer …

```
type_generator(:NumberGen, Number)
g = NumberGen()

for j in 1:5

    besttraces = Vector{Any}()

    for i in 1:100
        try
            x, state = generate(g)
            dump(x)
            push!(besttraces, state.cmtrace)
        catch _e
            if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
                rethrow(_e)
            end
            warn("exception raised in constructor method")
        end
    end

    estimateparams!(choicemodel(g), besttraces)

end
```

```
type_generator(:NumberGen, Number)
g = NumberGen()

for j in 1:5

    besttraces = Vector{Any}()

    for i in 1:100
        try
            x, state = generate(g)
            dump(x)
            push!(besttraces, state.cmtrace)
        catch _e
            if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
                rethrow(_e)
            end
            warn("exception raised in constructor method")
        end
    end

    estimateparams!(choicemodel(g), besttraces)

end
```

```
type_generator(:NumberGen, Number)
g = NumberGen()

for j in 1:5

    besttraces = Vector{Any}()

    for i in 1:100
        try
            x, state = generate(g)
            dump(x)
            push!(besttraces, state.cmtrace)
        catch _e
            if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
                rethrow(_e)
            end
            warn("exception raised in constructor method")
        end
    end

    estimateparams!(choicemodel(g), besttraces)

end
```

```
type_generator(:NumberGen, Number)
g = NumberGen()

for j in 1:5

    besttraces = Vector{Any}()

    for i in 1:100
        try
            x, state = generate(g)
            dump(x)
            push!(besttraces, state.cmtrace)
        catch _e
            if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
                rethrow(_e)
            end
            warn("exception raised in constructor method")
        end
    end

    estimateparams!(choicemodel(g), besttraces)

end
```

```
type_generator(:NumberGen, Number)
g = NumberGen()

for j in 1:5

    besttraces = Vector{Any}()

    for i in 1:100
        try
            x, state = generate(g)
            dump(x)
            push!(besttraces, state.cmtrace)
        catch _e
            if !isa(_e, DataGeneratorTranslators.TypeGenerationException)
                rethrow(_e)
            end
            warn("exception raised in constructor method")
        end
    end

    estimateparams!(choicemodel(g), besttraces)

end
```

```
@testset "fuzz test +" begin

    type_generator(:AddSigGen, Tuple{Number, Number})
    g = AddSigGen()

    for j in 1:5

        besttraces = Vector{Any}()

        for i in 1:100
            try
                x, state = generate(g)
                dump(x)
                push!(besttraces, state.cmtrace)
                res = try +(x...) catch _f _f end
                @test any(t->isa(res, t),
                      [Number, InexactError, OverflowError, DivideError,])
             catch _e
            ...
        end

        estimateparams!(choicemodel(g), besttraces)

    end

end
```

```
@testset "fuzz test +" begin

    type_generator(:AddSigGen, Tuple{Number, Number})
    g = AddSigGen()

    for j in 1:5

        besttraces = Vector{Any}()

        for i in 1:100
            try
                x, state = generate(g)
                dump(x)
                push!(besttraces, state.cmtrace)
                res = try +(x...) catch _f _f end
                @test any(t->isa(res, t),
                      [Number, InexactError, OverflowError, DivideError,])
             catch _e
               ...
        end

        estimateparams!(choicemodel(g), besttraces)

    end

end
```

```
@testset "fuzz test +" begin

    type_generator(:AddSigGen, Tuple{Number, Number})
    g = AddSigGen()

    for j in 1:5

        besttraces = Vector{Any}()

        for i in 1:100
            try
                x, state = generate(g)
                dump(x)
                push!(besttraces, state.cmtrace)
                res = try +(x...) catch _f _f end
                @test any(t->isa(res, t),
                      [Number, InexactError, OverflowError, DivideError,])
            catch _e
                ...
            end

        estimateparams!(choicemodel(g), besttraces)

    end

end
```

```
@testset "fuzz test +" begin

    type_generator(:AddSigGen, Tuple{Number, Number})
    g = AddSigGen()

    for j in 1:5

        besttraces = Vector{Any}()

        for i in 1:100
            try
                x, state = generate(g)
                dump(x)
                push!(besttraces, state.cmtrace)
                res = try +(x...) catch _f _f end
                @test any(t->isa(res, t),
                      [Number, InexactError, OverflowError, DivideError,])
             catch _e
            ...
        end

        estimateparams!(choicemodel(g), besttraces)

    end

end
```

https://github.com/simonpoulding/DataGeneratorTranslators.jl 0.6 soon

https://github.com/simonpoulding/DataGenerators.jl

Simon Poulding and Robert Feldt,
Automated Random Testing in Multiple Dispatch Languages,
Intn'l Conf. on Software Testing, Verification and Validation (ICST) 2017
www.simonpoulding.net/papers/icst_2017_preprint.pdf

Email: simon.poulding@gmail.com

Twitter: @simonpoulding

JuliaCon 2017 Slack: @simonpoulding