More type systems

Physical units

A unit expression is defined by following grammar

$u, v := b \mid 1 \mid u * v \mid u^{-1}$

where u, v are unit expressions themselves and b is a base unit:

$b := m \mid kg \mid s \mid A \mid K \mid cd \mid mol$

You may use B to denote the set of the unit types

$B = \{ m, kg, s, A, K, cd, mol \}$

For readability, we use the syntactic sugar

$u^\wedge n = u * \ldots * u$ if $n > 0$

1 if $n = 0$

$u^{-1} * \ldots * u^{-1}$ if $n < 0$

and $u/v = u * v^{-1}$

Physical units

a) Give the type rules for the arithmetic operations + , *, /, √, sin, abs. Assume that the trigonometric functions take as argument radians, which are dimensionless (since they are defined as the ratio of arc length to radius). You can denote that a variable is dimensionless by $\Gamma \vdash e : 1$

$$\frac{\Gamma \vdash a : U \qquad \Gamma \vdash b : U}{\Gamma \vdash a + b : U}$$

$$\frac{\Gamma \vdash a : U \qquad \Gamma \vdash b : V}{\Gamma \vdash a * b : U*V}$$

$$\frac{\Gamma \vdash a : U \qquad \Gamma \vdash b : V}{\Gamma \vdash a / b : U/V}$$

$$\frac{\Gamma \vdash a : U*U}{\Gamma \vdash \sqrt{a} : U}$$

$$\frac{\Gamma \vdash a : 1}{\Gamma \vdash \sin(a) : 1}$$

$$\frac{\Gamma \vdash a : U}{\Gamma \vdash \mathrm{abs}(a) : U}$$

Physical units

b) The unit expressions as defined above are strings, so that e.g.
(s^4*m^2)/(s^2*m^3) ≠ s^2*m
however physically these units match.

Define a procedure on the unit expressions such that your type rules type check expressions, whenever they are correct according to physics.

```
trait PUnit
case class Times(a: PUnit, b: PUnit) extends PUnit
case class Inverse(a: PUnit) extends PUnit
case class SI(v: String) extends PUnit
case class One extends PUnit
```

Physical units

```
def numerator(t: PUnit): List[SI] = t match {
  case Times(a, b) => numerator(a) ++ numerator(a)
  case Inverse(a) => denominator(a)
  case SI(_) => List(t)
  case One => Nil
}
def denominator(t: PUnit): List[SI] = t match {
  case Times(a, b) => denominator(a) ++ denominator (a)
  case Inverse(a) => numerator(a)
  case SI(_) => List()
  case One => Nil
}
```

Physical units

```
def simplify(t: PUnit): PUnit = {
  val num = numerator(t)
  val den = denominator(t)
  val inter = num intersect den
  val num2 = (num -- inter).sortBy(_.v)
  val den2 = (den – inter).sortBy(_.v)
  val a = (One /: num2) { case (res, p) => Times(res, p)}
  val b = (One /: den2) { case (res, p) => Times(res, p)}
  Times(num2, Inverse(denum2))
}
```

$$\frac{\Gamma \vdash a : U}{\Gamma \vdash a : \mathrm{simplify}(U)}$$

$$\frac{\Gamma \vdash a : (U*U)^{-1}}{\Gamma \vdash \sqrt{a} : U^{-1}}$$

Physical units

c)Determine the type of T in the following code fragment. The values in angle brackets give the unit type expressions of the variables and Pi is the usual constant π in the Scala math library. Give the full type derivation tree using your rules from a) and b), i.e. the tree that infers the types of the variables R, w, T.

```
val x: <m> = 800
val y: <m> = 6378
val g: <m/(s*s)> = 9.8
val R = x + y
val w = sqrt(g/R)
val T = (2 * Pi) / w
```

Physical units

```
val x: <m> = 800
val y: <m> = 6378
val g: <m/(s*s)> = 9.8
val R = x + y
val w = sqrt(g/R)
val T = (2 * Pi) / w
```

$$\frac{\Gamma \vdash x : m \qquad \Gamma \vdash y : m}{\Gamma \vdash x + y : m}$$

$$\frac{\Gamma \vdash g : m/(s*s) \qquad \dfrac{}{\Gamma \vdash R : m}}{\Gamma \vdash g / R : (m/(s*s)) / m}$$

$$\frac{\Gamma \vdash g / R : (m/(s*s)) / m}{\Gamma \vdash g / R : 1/(s*s)}$$

$$\frac{\Gamma \vdash g / R : 1/(s*s)}{\Gamma \vdash g / R : (1/s)*(1/s)}$$

$$\frac{\Gamma \vdash g / R : (1/s)*(1/s)}{\Gamma \vdash \sqrt{(g/R)} : 1/s}$$

$$\frac{\Gamma \vdash \sqrt{(g/R)} : 1/s}{\Gamma \vdash w : 1/s}$$

$$\frac{\Gamma \vdash 2 : 1 \qquad \Gamma \vdash \pi : 1}{\Gamma \vdash 2*\pi : 1*1}$$

$$\frac{\Gamma \vdash 2*\pi : 1*1}{\Gamma \vdash 2*\pi : 1}$$

$$\frac{\Gamma \vdash 2*\pi : 1 \qquad \Gamma \vdash w : 1/s}{\Gamma \vdash (2*\pi / w) : 1/(1/s)}$$

$$\frac{\Gamma \vdash (2*\pi / w) : 1/(1/s)}{\Gamma \vdash (2*\pi / w) : s}$$

Physical units

d) Consider the following function that computes the Coulomb force, and suppose for now that the compiler can parse the type expressions:

```
def coulomb(k: <(N*m)/(C*C)>, q1: <C>, q2: <C>, r:
<m>): <N> {
  return (k* q1 * q2)/(r*r)
}
```

The derived types are `C = A*s` and `N = kg*m / s^2`.

Does the code type check? Justify your answer rigorously.

No: Expected N, got N/m. Type tree for return expression.

Physical units

Γ⊢ k : N*m/(C*C)   Γ⊢ q1 : C

Γ⊢ k*q1 : N*m/(C*C) * C

Γ⊢ k*q1 : N*m/C   Γ⊢ q2 : C

Γ⊢ k*q1*q2 : (N*m/C * C)   Γ⊢ r: m   Γ⊢ r: m

Γ⊢ k*q1*q2 : m*N   Γ⊢ r*r: m*m

Γ⊢ k*q1*q2/(r*r) : m*N / (m*m)

Γ⊢ k*q1*q2/(r*r) : N/m