MSL

```
Monoid := Theory {
    U : type;
    * : (U,U) -> U;
    e : U;
    axiom right_identity_*_e :
    forall x : U . (x * e) = x
    axiom left_identity_*_e :
    forall x : U . (e * x) = x;
    axiom associativity_* :
    forall x,y,z : U .
    ((x * y) * z) = (x * (y * z));
}
```

Coq

```
Class Monoid {A : type}
    (dot : A -> A -> A)
    (one : A) : Prop := {
    dot_assoc :
      forall x y z : A,
        (dot x (dot y z))
        = dot (dot x y) z
    unit_left :
      forall x, dot one x = x
    unit_right :
      forall x, dot x one = x
}
```

Alternative Definition:

```
Record monoid := {
    dom : Type;
    op : dom -> dom -> dom
        where "x␣*␣y" := (op x y);
    id : dom where "1" := id ;
    assoc : forall x y z, x * (y *
        z)
                        = (x * y)
                          * z;
    left_neutral : forall x, 1 * x
        = x;
    right_neutral : forall x, x *
       1 = x
}.
```

Haksell

```
class Semigroup a => Monoid a
    where
     mempty :: a
     mappend :: a -> a -> a
     mappend = (<>)
     mconcat :: [a] -> a
     mconcat =
     foldr mappend mempty
```

MMT

```
theory Semigroup : ?NatDed =
  u : sort
  comp : tm u → tm u → tm u
                  #1 * 2 prec 40
  assoc : ⊢ ∀ [x] ∀ [y] ∀ [z]
             (x * y) * z = x * (y *
                 z)
  assocLeftToRight :
      { x y z } ⊢ (x * y) * z
                   = x * (y * z)
      = [x,y,z]
          allE (allE (allE assoc x)
               y) z
      #assocLR %I1 %I2 %I3
  assocRightToLeft :
      {x,y,z} ⊢ x * (y * z)
                  = (x * y) * z
      = [x,y,z] sym assocLR
      # assocRL %I1 %I2 %I3
theory Monoid : ?NatDed
     includes ?Semigroup
     unit : tm u # e
     @_description the unit
            element of the monoid
     unit_axiom : ⊢ ∀ [x] = x * e =
         x
     @_description the axiom
            of the neutral element
```

Agda

```
data Monoid (A : Set)
              (Eq : Equivalence A) :
                   Set
where
    monoid :
      (z : A)
      (_+_ : A -> A -> A)
      (left_Id : LeftIdentity Eq z
           _+_)
      (right_Id : RightIdentity Eq
           z _+_)
      (assoc : Associative Eq _+_)
           ->
      Monoid A Eq
```

Alternative Definition:

```
record Monoid c ℓ :
    Set (suc (c ⊔ ℓ)) where
      infixl 7 _•_
      infix 4 _≈_
      field
        Carrier : Set c
        _≈_ : Rel Carrier ℓ
        _•_ : Op₂ Carrier
        isMonoid :
        IsMonoid _≈_ _•_ ϵ
```

where IsMonoid is defined as

```
record IsMonoid (• : Op₂) (ε : A)
                 : Set (a ⊔ ℓ) where
    field
        isSemigroup : IsSemigroup
            •
        identity : Identity ε •

        identityˡ : LeftIdentity ε
            •
        identityˡ = proj₁ identity

        identityʳ : RightIdentity
            ε •
        identityʳ = proj₂ identity
```