Haskell Platform + Semigroups + Semigroupoids + Either: Most Frequently used types typeclasses

	Class:	*SemiGroup* Data.Semigroup	*Monoids* Data.Monoid	*Functor* Data.Functor	*Alt* Data.Functor.Alt	*Plus* Data.Functor.Plus	*Apply* Data.Functor.Apply	*Applicative* Control.Applicative	*Alternative* Control.Applicative	*Bind* Data.Functor.Bind	*Monad* Control.Monad	*Monad Plus* Control.Monad
"Base"	**Type**	**<>**	**<>, mempty**	**fmap**	**<!>**	**zero**	**<.>**	**<*>, pure**	**<|>,empty**	**>>-,join**	**>>=, return**	**mplus,mzero**
	Ordering DO	x	x									
	()	x	x									
Bool	**All** DMONO	x	x									
Bool	**Any** DMONO	x	x									
Word8	**ByteString** DBS,DBSL	x	x									
Char	**Text** DT,DTL	x	x									
Num	**Sum a** MONO	Num a	Num a									
	Product a MONO	Num a	Num a									
Ord	**Min a** DSEMI	Ord a	Ord a, Bounded a									
	Max a DSEMI	Ord a	Ord a, Bounded a									
	Dual a DMONO	Monoid a	Monoid a									
	[a] DLIST	x	x									
[]	**[]** DLIST			x	x	x	x	x	x	x	x	x
	ZipList CAPP			x			x	x	*			
	Seq a DSEQ	x	x									
	Seq DSEQ			x	x	x	x	x	x	x	x	x
	NonEmpty a DLNE	x										
	NonEmpty DLNE			x	x		x	x		x	x	
	Set a DSET	Ord a	Ord a									
Int	**IntSet** DISET	x	x									
	HashSet a DHSET	Hashable a, Eq a	Hashable a, Eq a									
	Map k a DMAP	Ord k	Ord k									
	Map k DMAP			x	Ord k	Ord k	Ord k			Ord k		
Int	**IntMap a** DIMAP	x	x									
	IntMap DIMAP			x	x	x	x			x		
	HashMap k a DHMAP	Hashable k, Eq k	Hashable k, Eq k									
	HashMap k DHMAP			x								
	Tree DTREE			x				x		x	x	
Maybe	**Maybe** DM			x	x	x	x	x	x	x	x	x
	Maybe a DM	Semigroup a	Monoid a									
Maybe	**Option** DSEMI			x	x	x	x	x	x	x	x	x
	Option a DSEMI	Semigroup a	Semigroup a									
Maybe	**First a** DM		x									
	Last a DM		x									
	First a DSEMI	x										
	Last a DSEMI	x										
	Either a DEITH			x	x		x	x		x	x	
	Either a b DEITH	x										
	Identity DFI			x			x	x		x	x	
	IdentityT m CMTI			Functor m	Alt m	Plus m	Apply m	Applicative m	Alternative m	Bind m	Monad m	MonadPlus m
Maybe	**MaybeT m** CMTM			Functor m	Bind m, Monad m	Bind m, Monad m	Bind m, Monad m	Functor m, Monad m	Functor m, Monad m	Bind m, Monad m	Monad m	Monad m
[]	**ListT m** CMTL			Functor m	Apply m	Apply m, Applicative m	Apply m	Applicative m	Applicative m	Bind m, Monad m	Monad m	Monad m
	ReaderT e m CMTR			Functor m	Alt m	Plus m	Apply m	Applicative m	Alternative m	Bind m	Monad m	MonadPlus m
	WriterT w m CMTW			Functor m	Alt m	Plus m	Apply m, Semigroup w	Monoid w, Applicative m	Monoid w, Alternative m	Bind m, Semigroup w	Monoid w, Monad m	Monoid w, MonadPlus m
	StateT s m CMTS			Functor m	Alt m	Plus m	Bind m	Functor m, Monad m	Functor m, MonadPlus m	Bind m	Monad m	MonadPlus m
Either e	**ErrorT e m** CMTERR			Functor m	Bind m, Monad m	Bind m, Monad m, Error e	Bind m, Monad m	Functor m, Monad m	Functor m, Monad m, Error e	Bind m, Monad m	Monad m, Error e	Monad m, Error e
	RWST r w s m CMTRWS			Functor m	Alt m	Plus m	Bind m, Semigroup w	Monoid w, Functor m, Monad m	Monoid w, Functor m, MonadPlus m	Bind m, Semigroup w	Monoid w, Monad m	Monoid w, MonadPlus m
Either	**EitherT e m** CMTE			Monad m	Monad m, Semigroup e		Monad m	Monad m, Monoid e	Monad m, Monoid e	Monad m	Monad m	Monad m, Monoid e
Either	**EitherT e m a** CMTE	Semigroup m										
	Parser DATTO			x				x	x		x	x
	ParsecT s u m TPARSEC			x				x	x		x	x
	WrappedMonoid m DSEMI	Monoid m	Monoid m									
	WrappedApplicative f DFA				Alternative f	Alternative f	Applicative f					
	WrappedMonad m CAPP			Monad m	MonadPlus m	MonadPlus m	Monad m	Monad m	MonadPlus m	Monad m		
	WrappedArrow a b CAPP			Arrow a	ArrowPlus a	ArrowPlus a	Arrow a	Arrow a	ArrowZero a, ArrowPlus a			
	ArrowMonad a CARR			Arrow a				Arrow a	ArrowPlus a		ArrowApply a	ArrowApply a, ArrowPlus a
	IO SIO			x	x	x	x	x		x	x	
	ST s CMST			x				x			x	
	STM			x				x	x		x	x
	ReadP			x				x	x		x	x
	ReadPrec			x				x	x		x	x
Tuples	**(a, b)**	Semigroup a, Semigroup b	Monoid a, Monoid b									
	(a, b, c)	Semigroup a, Semigroup b, Semigroup c	Monoid a, Monoid b, Monoid c									
	(,) a			x			Semigroup a	Monoid a		Semigroup m		
	a → b	Semigroup b	Monoid b									
	Endo a DMONO	x	x									
	(→) a			x			x	x		x	x	
	Const a b CAPP	Semigroup a		x								
	Const m CAPP			x			Semigroup m	Monoid m				
	Static f a DSS			Functor f	Alt f	Plus f	Apply f	Applicative f				
	Compose f g DSS			Functor f, Functor g				Aplicative f, Applicative g	Alternative f, Alternative g			
	Product f g DFC			Functor f, Functor g			Apply f, Apply g	Applicative f, Applicative g	Applicative f, Applicative g	Bind f, Bind g		
	Cokleisli w a			x			x	x			x	

Note 1: **Typeclasses in Haskell imply laws, not (necessarily) semantics**

Note 2: **While a uniform semantic / behaviour would be nice to have, most 'class laws' were found insufficient to provide this, and yet no additional laws were specified (hysterical raisins)**

Note 3: **The following classes were not included: Eq, Show, Monad*, …**

CAPP	Control.Applicative	base	DBS	Data.ByteString	bytestring	DM	Data.Maybe, Prelude	base
CARR	Control.Arrow	base	DBSL	Data.ByteString.Lazy	bytestring	DMAP	Data.Map	containers
CMST	Control.Monad.ST	base	DEITH	Data.Either, Prelude	base	DMONO	Data.Monoid, Prelude	base
CMTE	Control.Monad.Trans.Either	transformers	DFA	Data.Functor.Apply	semigroupoids	DO	Data.Ord, Prelude	base
CMTERR	Control.Monad.Trans.Error	transformers	DFB	Data.Functor.Bind	semigroupoids	DSEMI	Data.Semigroup	semigroups
CMTI	Control.Monad.Trans.Identity	transformers	DFC	Data.Functor.Compose	transformers	DSEQ	Data.Sequence	containers
CMTL	Control.Monad.Trans.List	transformers	DFI	Data.Functor.Identity	transformers	DSET	Data.Set	containers
CMTM	Control.Monad.Trans.Maybe	transformers	DHMAP	Data.HashMap	unordered-containers	DSS	Data.Semigroupoid.*	semigroupoids
CMTR	Control.Monad.Trans.Reader	transformers	DHSET	Data.HashSet	unordered-containers	DT	Data.Text	text
CMTRWS	Control.Monad.Trans.RWST	transformers	DIMAP	Data.IntMap	containers	DTL	Data.Text.Lazy	text
CMTS	Control.Monad.Trans.State	transformers	DISET	Data.InsSet	containers	DTREE	Data.Tree	containers
CMTW	Control.Monad.Trans.Writer	transformers	DLIST	Data.List, Prelude	base	SIO	System.IO, Prelude	base
DATTO	Data.Attoparsec	attoparsec	DLNE	Data.List.NonEmpty	semigroups	TPARSEC	Text.Parsec.Prim	parsec

	Binary operation semantic / Mempty 'meaning' / Additional Laws											
	Class:	*SemiGroup* Data.Semigroup	*Monoids* Data.Monoid		*Alt* Data.Functor.Alt	*Plus* Data.Functor.Plus	*Apply* Data.Functor.Apply	*Applicative* Control.Applicative	*Alternative* Control.Applicative	*Bind* Data.Functor.Bind	*Monad* Control.Monad	*Monad Plus* Control.Monad
"Base"	**Type**	**<>**	**<>**	**mempty**	**<!>**	**zero**	**<.>**	**<*>, pure**	**<|>,empty**	**>>-,join**	**>>=, return**	**mplus,mzero**
	Ordering	Choice	Choice	EQ								
	()	None	None	()								
Bool	**All**	Combine	Combine	True								
Bool	**Any**	Combine	Combine	False								
Word8	**ByteString**	Combine	Combine	Empty								
Char	**Text**	Combine	Combine	Empty								
Num	**Sum a**	Combine	Combine	0								
	Product a	Combine	Combine	1								
Ord	**Min a**	Choice	Choice	maxBound								
	Max a	Choice	Choice	minBound								
	Dual a	~ a	~ a	~ a								
	[a]	Both	Both	Empty								
[]	**[]**				Both	Empty	<*>	Both	x	x	x	Left Dist.
	ZipList						<*>	Both	*			
	Seq a	Both	Both	Empty								
	Seq				Both	Empty	ap	x	x	x	x	x
	NonEmpty a	Both										
	NonEmpty				Both		ap	x		x	x	
	Set a	Both	Both	Empty								
Int	**IntSet**	Both	Both	Empty								
	HashSet a	Both	Both	Empty								
	Map k a	Both	Both	Empty								
	Map k				Both	Empty	(…)			Ord k		
Int	**IntMap a**	Both	Both	Empty								
	IntMap				Both	Empty	(…)			x		
	HashMap k a	Both	Both	Empty								
	HashMap k											
	Tree							x		x	x	
Maybe	**Maybe**				Choice	Empty	apDefault	x	Choice	x	x	Left Catch
	Maybe a	Combine	Combine	Empty								
Maybe	**Option**				Choice	Empty	<*>	x	Choice	x	x	x
	Option a	Combine	Combine	Empty								
Maybe	**First a**		Choice	Empty								
	Last a		Choice	Empty								
	First a	Choice										
	Last a	Choice										
	Either a				Choice		(…)	x		x	x	
	Either a b	Choice		~ Empty								
	Identity						<*>	x		x	x	
	IdentityT m				Combine	~ m	<.>	Applicative m	Alternative m	Bind m	Monad m	MonadPlus m
Maybe	**MaybeT m**				Choice	Empty	apDefault	Functor m, Monad m	Functor m, Monad m	Bind m, Monad m	Monad m	Monad m
[]	**ListT m**				Combine	Empty	(…)	Applicative m	Applicative m	Bind m, Monad m	Monad m	Monad m
	ReaderT e m				Combine	~ m	(…)	Applicative m	Alternative m	Bind m	Monad m	MonadPlus m
	WriterT w m				Combine	~m	(…)	Monoid w, Applicative m	Monoid w, Alternative m	Bind m, Semigroup w	Monoid w, Monad m	Monoid w, MonadPlus m
	StateT s m				Combine	~ m	apDefault	Functor m, Monad m	Functor m, MonadPlus m	Bind m	Monad m	MonadPlus m
Either e	**ErrorT e m**				Choice	~ Empty	apDefault	Functor m, Monad m	Functor m, Monad m, Error e	Bind m, Monad m	Monad m, Error e	Monad m, Error e
	RWST r w s m				Combine	~ m	apDefault	Monoid w, Functor m, Monad m	Monoid w, Functor m, MonadPlus m	Bind m, Semigroup w	Monoid w, Monad m	Monoid w, MonadPlus m
Either	**EitherT e m**				Both?		(…)	Monad m, Monoid e	Both	Monad m	Monad m	Monad m, Monoid e
Either	**EitherT e m a**	Choice										
	Parser DATTO			x				x	x		x	x
	ParsecT s u m TPARSEC			x				x	x		x	x
	WrappedMonoid m	~ m	~ m	~ m								
	WrappedApplicative f				~ f	~ f	~ f					
	WrappedMonad m				~ f	~ m	~ m	~ m	~ m	~ m		
	WrappedArrow a b				~ a	~ a	~ a	~ a	ArrowZero a, ArrowPlus a			
	ArrowMonad a							~ a	~ a		ArrowApply a	ArrowApply a, ArrowPlus a
	IO				Choice	error	<*>	x		x	x	
	ST s							x			x	
	STM							x	x		x	Left Catch
	ReadP							x	x		x	x
	ReadPrec							x	x		x	x
Tuples	**(a, b)**	~a, ~b	~a, ~b	~a, ~b								
	(a, b, c)	~a, ~b, ~c	~ a, ~ b, ~c	~ a, ~ b, ~c								
	(,) a						(..)	Monoid a		Semigroup m		
	a → b	~ b	~ b	~ b								
	Endo a	Neither	Neither	Neither								
	(→) a						<*>	x		x	x	
	Const a b	~ a										
	Const m						<>	Monoid m				
	Static f a				~ f	Plus f	(..)	Applicative f				
	Compose f g							Aplicative f, Applicative g	Alternative f, Alternative g			
	Product f g (D.F.P)						(..)	Applicative f, Applicative g	Applicative f, Applicative g	Bind f, Bind g		
	Cokleisli w a						(..)	x			x	

Choice	Binary operation chooses one of the values
Combine	Binary operation combines both values
Both [Lists/Seq/Nempty]	Binary operation chooses all possible outcomes, thus combining them
Both [Sets/Maps]	Binary operation combines both values, choosing from left when conflicts arise
Neither	None of the above concepts makes sense in this context
~ a	as a
Empty	mempty/zero value reflects empty container