1. Tipo Lista⟨T⟩

```
tipo Lista⟨T⟩ {
  observador asSeq (l : Lista⟨T⟩) : [T];
}
```

El observador asSeq devuelve una lista del lenguaje de especificación que contiene los mismos elementos, y en el mismo orden, que la Lista⟨T⟩ recibida. En adelante, haremos un abuso de notación, y omitiremos el observador asSeq en las especificaciones que involucren al tipo Lista⟨T⟩. Uds. pueden hacer lo mismo si lo desean.

```
problema Lista⟨T⟩ (this : Lista⟨T⟩) {
  modifica this;
  asegura this == [];
}

problema longitud (this : Lista⟨T⟩) = result : ℤ {
  asegura result == |this|;
}

problema iesimo (this : Lista⟨T⟩,i : ℤ) = result : T {
  requiere i ∈ [0..|this|);
  asegura result = this_i;
}

problema agregar (this : Lista⟨T⟩,e : T) {
  modifica this;
  asegura this == e : pre(this);
}

problema agregarAtras (this : Lista⟨T⟩,e : T) {
  modifica this;
  asegura this == pre(this) ++[e];
}

problema cabeza (this : Lista⟨T⟩) = result : T {
  requiere |this| > 0;
  asegura result == cabeza(this);
}

problema cola (this : Lista⟨T⟩) {
  modifica this;
  requiere |this| > 0;
  asegura this == cola(pre(this));
}

problema pertenece (this : Lista⟨T⟩,e : T) = result : Bool {
  asegura result == (e ∈ this);
}

problema concatenar (this,otraLista : Lista⟨T⟩) {
  modifica this;
  asegura this == pre(this) ++otraLista;
}

problema operator== (this,otraLista : Lista⟨T⟩) {
  asegura result == (this == otraLista);
}

problema sacar (this : Lista⟨T⟩,e : T) {
  modifica this;
  asegura this == [x | x ← pre(this), x ≠ e];
}

problema darVuelta (this : Lista⟨T⟩) {
  modifica this;
  asegura |this| == |pre(this)| ∧ (∀i ← [0..|this|)) this_i = pre_{|this|−i−1};
}
```

```
problema posicion (this : Lista⟨T⟩,e : T) = result : ℤ {
  requiere e ∈ this ;
  asegura result = [ i | i ← [0..|this|), this_i = e ]_0 ;
}

problema eliminarPosicion (this : Lista⟨T⟩,i : ℤ) {
  modifica this ;
  requiere i ∈ [0..|pre(this)|) ;
  asegura this = pre(this)[0..i) + +pre(this)(i..|pre(this)|) ;
}

problema cantidadDeApariciones (this : Lista⟨T⟩, e:T) = result : ℤ {
  asegura result = |[ e | e' ← this, e' == e ]| ;
}
```

2. SMS

```
tipo SMS {
  observador nroDestino (s: SMS) : Numero ;
  observador texto (s: SMS) : Texto ;

  invariante noEstaVacio : |texto(s)| > 0 ;
  invariante noSupera160 : |texto(s)| ≤ 160 ;
}

problema nuevoSMS (n: Numero, t: Texto, this: SMS) {
  requiere 0 < |t| ≤ 160 ;
  modifica this ;
  asegura nroDestino(this) == n ∧ texto(this) == t ;
}

problema nroDestinoSMS (this: SMS) = result : Numero {
  asegura result == nroDestino(this) ;
}

problema textoSMS (this: SMS) = result : Texto {
  asegura result == texto(this) ;
}
```

3. Pregunta

```
tipo Pregunta {
  observador texto (p: Pregunta) : Texto ;
  observador rtaCorrecta (p: Pregunta) : Texto ;
  observador puntaje (p: Pregunta) : Puntaje ;

  invariante puntajesPositivos : puntaje(p) > 0 ;
}

problema nuevaP (t, r: Texto, p: Puntaje, this: Pregunta) {
  requiere p > 0 ;
  modifica this ;
  ; asegura texto(this) == t ∧ rtaCorrecta(this) == r ∧ puntaje(this) == p ;
}

problema textoP (this: Pregunta) = result : Texto {
  asegura result == texto(this) ;
}

problema rtaCorrectaP (this: Pregunta) = result : Texto {
  asegura result == rtaCorrecta(this) ;
}

problema puntajeP (this: Pregunta) = result : Puntaje {
  asegura result == puntaje(this) ;
}
```

4. Trivia

```
tipo Trivia {
  observador keywords (t: Trivia) : [Keyword] ;
  observador preguntas (t: Trivia) : [Pregunta] ;
  observador participantes (t: Trivia) : [Usuario] ;
  observador ganadores (t: Trivia) : [Usuario] ;
  observador proxPregunta (t: Trivia, u: Usuario) : Pregunta ;
       requiere u ∈ participantes(t) ∧ u ∉ ganadores(t) ;
  observador puntajeAcumulado (t: Trivia, u: Usuario) : Puntaje ;
       requiere u ∈ participantes(t) ;

  invariante noEsTrivialGanar : |preguntas(t)| > 0 ;
  invariante noHayPreguntasRepetidas : (∀i, j ← [0..|preguntas(t)|), i ≠ j)
     texto(preguntas(t)_i) ≠ texto(preguntas(t)_j) ;
  invariante noHayUsuariosRepetidos : sinRepetidos(participantes(t)) ∧ sinRepetidos(ganadores(t)) ;
  invariante proxPreguntaValida : (∀u ← participantes(t), p ∉ ganadores(t))proxPregunta(t, u) ∈ preguntas(t) ;
  invariante puntajesNoNegativos : (∀u ← participantes(t))puntajeAcumulado(t, u) ≥ 0 ;
  invariante preguntasEntranEnSMS : (∀p ← preguntas(t))0 < |texto(p)| ≤ 139 ;
  invariante maximoPuntaje : ∑[puntaje(p)|p ← preguntas(t)] ≤ 999 ;
  invariante losGanadoresSonParticipantes : (∀g ← ganadores(t))g ∈ participantes(t) ;
}

problema nuevaT (ks: [Keyword], ps: [Pregunta], this: Trivia) {
  requiere |ps| > 0 ;
  requiere (∀i, j ← [0..|ps|), i ≠ j)texto(ps_i) ≠ texto(ps_j) ;
  requiere (∀p ← ps)0 < |texto(p)| ≤ 139 ;
  requiere ∑[puntaje(p)|p ← ps] ≤ 999 ;
  modifica this ;
  asegura mismos(keywords(this), ks) ;
  asegura preguntas(this) == ps ;
  asegura |(participantes(this))| == 0 ;
  asegura |(ganadores(this))| == 0 ;
}

problema preguntasT (this: Trivia) = result : [Pregunta] {
  asegura result == preguntas(this) ;
}

problema keywordsT (this: Trivia) = result : [Keyword] {
  asegura mismos(result, keywords(this)) ;
}

problema participantesT (this: Trivia) = result : [Usuario] {
  asegura mismos(result, participantes(this)) ;
}

problema ganadoresT (this: Trivia) = result : [Usuario] {
  asegura result == ganadores(this) ;
}

problema proxPreguntaParticipanteT (this: Trivia, u: Usuario) = result : Pregunta {
  requiere u ∈ participantes(this) ∧ u ∉ ganadores(this) ;
  asegura result == proxPregunta(this, u) ;
}

problema puntajeAcumuladoT (this: Trivia, u: Usuario) = result : Puntaje {
  requiere u ∈ participantes(this) ;
  asegura result == puntajeAcumulado(this, u) ;
}

problema registrarJugadorT (this: Trivia, u: Usuario, k: Keyword) = result : SMS {
  requiere keywordValida : k ∈ keywords(this) ;
  requiere noEstaRegistrado : u ∉ participantes(this) ;
  modifica this ;
  asegura seMantienenKeywords : mismos(keywords(this), keywords(pre(this))) ;
```

asegura seMantienenPreguntas : $preguntas(this) == preguntas(pre(this))$;
asegura seAgregaParticipante : $mismos(participantes(this), u : participantes(pre(this)))$;
asegura seMantienenGanadores : $ganadores(this) == ganadores(pre(this))$;
asegura otrosParticipantesMantienenEstado :
$(\forall p \leftarrow participantes(pre(this)))puntajeAcumulado(this, p) == puntajeAcumulado(pre(this), p) \land$
$(p \notin ganadores(pre(this)) \rightarrow proxPregunta(this, p) == proxPregunta(pre(this), p))$;
asegura empiezaPorElPrincipio : $proxPregunta(this, u) == preguntas(this)_0$;
asegura empiezaSinPuntos : $puntajeAcumulado(this, u) == 0$;
asegura smsBienvenidaOK : $esMensajePrimerPregunta(res, this, u)$;
}

problema desregistrarJugadorT (this: Trivia, u: Usuario) {
requiere estaRegistrado : $u \in participantes(this)$;
modifica $this$;
asegura seMantienenKeywords : $mismos(keywords(this), keywords(pre(this)))$;
asegura seMantienenPreguntas : $preguntas(this) == preguntas(pre(this))$;
asegura seEliminaParticipante : $mismos(u : participantes(this), participantes(pre(this)))$;
asegura quedanGanadores : $ganadores(this) == [\, g \mid g \leftarrow ganadores(pre(this)), g \neq u \,]$;
asegura otrosParticipantesMantienenEstado :
$(\forall p \leftarrow participantes(this))puntajeAcumulado(this, p) == puntajeAcumulado(pre(this), p) \land$
$(p \notin ganadores(this) \rightarrow proxPregunta(this, p) == proxPregunta(pre(this), p))$;
}

problema procesarRespuestaT (this: Trivia, u: Usuario, r: Texto) = result : SMS {
requiere estaRegistrado : $u \in participantes(this)$;
modifica $this$;
asegura seMantienenKeywords : $mismos(keywords(this), keywords(pre(this)))$;
asegura seMantienenPreguntas : $preguntas(this) == preguntas(pre(this))$;
asegura seMantienenParticipante : $mismos(participantes(this), participantes(pre(this)))$;
asegura siTerminoEsGanador : $leFaltaResponderSoloUna(pre(this), u) \rightarrow$
$ganadores(this) == ganadores(pre(this)) + +[u]$;
asegura siNoTerminoNoEsGanador : $\neg leFaltaResponderSoloUna(pre(this), u) \rightarrow ganadores(this) == ganadores(pre(this))$;
asegura otrosParticipantesMantienenEstado :
$(\forall p \leftarrow participantes(pre(this)), p \neq u)puntajeAcumulado((this, p) == puntajeAcumulado(pre(this), p) \land$
$(p \notin ganadores(pre(this)) \rightarrow proxPregunta((this, p) == proxPregunta(pre(this), p))$;
asegura avanzaUnaPregunta : $leFaltaResponderVarias(pre(this), u) \rightarrow$
$proxPregunta(this, u) == preguntas(this)_{indicePreg(pre(this),u)+1}$;
asegura actualizaPuntaje : $(u \in ganadores(pre(this)) \rightarrow puntajeAcumulado(this, u) == puntajeAcumulado(pre(this), u)) \land$

$(u \notin ganadores(pre(this)) \rightarrow puntajeAcumulado(this, u) == puntajeAcumulado(pre(this), u) + puntosQueSuma(pre(this),$
asegura textoSiTermino : $leFaltaResponderSoloUna(pre(this), u) \rightarrow$
$esMensajeFinal(result, this, u)$;
asegura textoSiNoTermino : $leFaltaResponderVarias(pre(this), u) \rightarrow$
$esMensajeProximaPregunta(pre(this), r, result, this, u)$;
asegura textoSiYaHabiaGanado : $u \in ganadores(pre(this)) \rightarrow$
$esMensajeDeSeguiPagando(result, u)$;
}

problema usuariosEficientesT (this: Trivia) = result : [Usuario] {
asegura $mismos(result, losQueMenosMandaron(this, losQueMasPuntosTienen(this)))$;
}

problema posicionEnT (this: Trivia, u: Usuario) = result : $\mathbb{Z}$ {
requiere estaRegistrado : $u \in participantes(this)$;
asegura $result == posicionEnTrivia(this, u)$;
}

problema palabrasRecurrentesT (this: Trivia, n: $\mathbb{Z}$) = result : [String] {
asegura $mismos(result, sacarRepetidos(soloMayoresAN$
$(n, lasMasRecurrentes(concat([palabras(texto(p)) + +palabras(rtaCorrecta(p)) | p \leftarrow preguntas(this)])))$;
}

5. Gateway

tipo Gateway {
  observador numero (g: Gateway) : Numero ;
  observador trivias (g: Gateway) : [Trivia] ;
  observador comandos (g: Gateway) : [Keyword] ;

  invariante sinTriviasRepetidas : $sinRepetidos(trivias(g))$;
  invariante sinComandosRepetidos : $sinRepetidos(comandos(g))$;
  invariante noSeSuperponenTrivias : $(\forall t_1, t_2 \leftarrow trivias(g), t_1 \neq t_2)\, |interseccion(keywords(t_1), keywords(t_2))| == 0$;
  invariante noSeSolapanComandosConTrivias : $(\forall t \leftarrow trivias(g))\, |interseccion(keywords(t), comandos(g))| == 0$;
}

problema nuevoG (n: Numero, cs: [Keyword], this: Gateway) {
  requiere $sinRepetidos(cs)$;
  asegura $numero(this) == n \land mismos(comandos(this), cs) \land |trivias(this)| == 0$;
}

problema numeroG (this: Gateway) = result : Numero {
  asegura $result == numero(this)$;
}

problema triviasG (this: Gateway) = result : [Trivia] {
  asegura $mismos(result, trivias(this))$;
}

problema comandosG (this: Gateway) = result : [Keyword] {
  asegura $mismos(result, comandos(this))$;
}

problema agregarTriviaG (this: Trivia, g: Gateway) {
  requiere noRepiteKeyword : $(\forall x \leftarrow trivias(this))\, |interseccion(keywords(t), keywords(x))| == 0$;
  requiere noTieneKeywordQueEsComando : $|interseccion(keywords(t), comandos(this))| == 0$;
  modifica $this$;
  asegura mantieneNroYComandos : $numero(this) == numero(pre(this)) \land mismos(comandos(this), comandos(pre(this)))$;
  asegura seAgregaTrivia : $mismos(trivias(this), t : trivias(pre(this)))$;
}

problema procesarComandoG (this: Gateway, u: Usuario, s: SMS) = result : SMS {
  requiere esParticipanteDeAlguna : $(\exists t \leftarrow trivias(this))u \in participantes(t)$;
  requiere esUnMensajeParaMi : $nroDestino(s) == numero(this)$;
  modifica $this$;
  asegura noEntiendoElComando : $\neg esComando(texto(s), pre(this)) \rightarrow pre(this) == this \land esMensajeNoTeEntiendo(result, u)$;
  asegura noEntiendoLaKeyword : $comandoConKeywordInvalida(texto(s), trivias(pre(this)), u) \rightarrow$
    $pre(this) == result \land esMensajeNoTeEntiendo(result, u)$;
  asegura numeroComandosYTriviasSiEsRanking : $esComandoRanking(texto(s)) \rightarrow pre(this) == this$;
  asegura mensajeSiEsRanking : $esComandoRanking(texto(s)) \rightarrow esMensajeRanking(s, result, u, pre(this))$;
  asegura numeroComandosYTriviasSiEsBaja : $nuevoGwPosBaja(pre(this), this, obtenerKeywordDeComando(texto(s)), u)$;
  asegura mensajeSiEsBaja : $esComandoBaja(texto(s)) \rightarrow esMensajeBaja(result, u)$;
}

problema procesarMensajeG (this: Gateway, u: Usuario, s: SMS) = result : SMS {
  requiere noEsUnComando : $\neg primerPalabraPertenece(texto(s), comandos(this))$;
  requiere esUnMensajeParaMi : $nroDestino(s) == numero(this)$;
  modifica $this$;
  asegura noEntiendo : $noEsKeyword(texto(s), trivias(pre(this))) \rightarrow esMensajeNoTeEntiendo(result, u)$;
  asegura nuevoJugador : $\neg esJugador(u, pre(this), s) \rightarrow$
    $(nuevoGwConJugadorNuevo(pre(this), this, extraerKeywordSMS(s), u) \land$
    $esMensajePrimerPregunta(result, triviaXKeyword(trivias(pre(this)), extraerKeywordSMS(s)), u))$;
  asegura yaEraGanador : $esJugadorGanador(u, pre(this), s) \rightarrow$
    $(pre(this) == this \land esMensajeDeSeguiPagando(result, u))$;
  asegura estaPorGanar : $esJugadorYLeFaltaUna(u, pre(this), s) \rightarrow$
    $(nuevoGwConNuevoGanador(pre(this), this, s, u) \land$
    $esMensajeFinal(result, triviaXKeyword(trivias(this), extraerKeywordSMS(s)), u))$;

asegura sigueJugando : $esJugadorLeFaltanVarias(u, pre(this), s) \rightarrow$
$(nuevoGwSinNuevoGanador(pre(this), this, s, u) \wedge$
$esMensajeProximaPregunta(triviaXKeyword(trivias(pre(this)), extraerKeywordSMS(s)), texto(s), result, u,$
$triviaXKeyword(trivias(this), extraerKeywordSMS(s)))$;
}

problema seAnotanEnTodasG (this: Gateway) = result : [Usuario] {
asegura $mismos(result, [u | u \leftarrow todosLosUsuarios(this), seAnotoEnTodas(this, u) \wedge terminoAlguna(this, u)])$;
}

6. Telco

tipo TelCO {
observador usuarios (t: TelCO) : [Usuario] ;
observador gateways (t: TelCO) : [Gateway] ;
observador mensajes (t: TelCO, u: Usuario) : [(SMS, SMS)] ;
requiere $u \in usuarios(t)$;

invariante usuariosDistintos : $sinRepetidos(usuarios(t))$;
invariante noHayGatewaysRepetidos : $sinRepetidos([numero(g) | g \leftarrow gateways(t)])$;
invariante smsEnviadosValidos : $(\forall u \leftarrow usuarios(t))(\forall m \leftarrow mensajes(t, u))$
$hayGatewayConNro(nroDestino(prm(m)), gateways(t))$;
invariante smsRecibidosValidos : $(\forall u \leftarrow usuarios(t))(\forall m \leftarrow mensajes(t, u))nroDestino(sgd(m)) == u$;
}

problema nuevoTE (gs: [Gateway], this: TelCO) {
asegura sinUsuarios : $usuarios(this) == []$;
asegura conGateways : $mismos(gateways(this), gs)$;
}

problema usuariosTE (this: TelCO) = result : [Usuario] {
asegura $mismos(result, usuarios(this))$;
}

problema esNroDeGatewayTE (this: TelCO, n: Numero) = result : Bool {
asegura $result == (\exists g \leftarrow gateways(this))numero(g) == n$;
}

problema gatewayTE (this: TelCO, n: Numero) = result : Gateway {
requiere esNroDeAlgunGateway : $(\exists g \leftarrow gateways(this))numero(g) == n$;
asegura $result == gatewayXNro(this, n)$;
}

problema mensajesTE (this: TelCO, u: Usuario) = result : [(SMS, SMS)] {
requiere esUsuario : $u \in usuarios(this)$;
asegura $result == mensajes(this, u)$;
}

problema agregarUsuarioTE (this: TelCO, u: Usuario) {
requiere noEsUsuario : $u \notin usuarios(this)$;
modifica $this$;
asegura nuevoUsuario : $mismos(u : usuarios(pre(this)), usuarios(this))$;
asegura mismosGateways : $mismos(gateways(pre(this)), gateways(this))$;
asegura mismosMensajes : $(\forall x \leftarrow usuarios(pre(this)))mensajes(pre(this), x) == mensajes(this, x)$;
asegura elNuevoNoTieneMensajes : $mensajes(this, u) == []$;
}

problema procesarMensajeTE (this: TelCO, u: Usuario, sms: SMS) {
requiere esUsuario : $u \in usuarios(this)$;
requiere existeGatewayDestino : $(\exists g \leftarrow gateways(this))numero(g) == nroDestino(sms)$;
modifica $this$;
asegura seMantienenLosUsuarios : $mismos(usuarios(pre(this)), usuarios(this))$;
asegura mismosMensajesDeOtros : $(\forall x \leftarrow usuarios(this), x \neq u)mensajes(pre(this), x) == mensajes(this, u)$;
asegura hayUnMensajeNuevo : $|mensajes(pre(this), u)| == |mensajes(this, u)| - 1$;

asegura mismosMensajesViejos : $mensajes(pre(this), u) == mensajes(this, u)[0..|mensajes(this, u)| - 1)$;
asegura losOtrosGatewaysNoCambian :
$mismos(sacarGateway(gateways(pre(this)), gatewayXNro(pre(this), n)), sacarGateway(gateways(this), gatewayXNro(th$
asegura cambiosEnGateway : $(\exists g \leftarrow gateways(g), numero(g) == nroDestino(sms))$
$(\exists r \leftarrow gateways(pre(this)), numero(g) == nroDestino(sms))$
$(gatewayDespuesDeProcesarMensaje(g, r, sms, u, ultimo(mensajes(this))))$;
}

7. Funciones Auxiliares

8. Auxiliares

8.1. B

`aux bienOMal` (t: Trivia, u: Usuario, r: Texto) : Texto $=$ if $rtaCorrecta(proxPregunta(t, u)) == sacarPrimerPalabra(r)$ then "$Bien$!" else "Mal!" ;

8.2. C

`aux cantAp` (x:T, xs:[T]) : $\mathbb{Z} = |[1|y \leftarrow xs, x == y]|$;

`aux comandoConKeywordInvalida` (s: String, ts: [Trivia], u: Usuario) : Bool $= (esComandoRanking(s) \vee esComandoBaja(s)) \wedge$
$(noEncuentraTrivia(segundaPalabra(s), ts) \vee noEsTriviaDelUsuario(ts, u))$;

`aux cuantosJueganAlgunaTrivia` (t: TelCO, g: Gateway) : $\mathbb{Z} =$
$|[u|u \leftarrow usuarios(t), (\exists x \leftarrow trivias(g))u \in participantes(x)]|$;

8.3. E

`aux esComando` (s: String, g: Gateway) : Bool $=$
$(primerPalabraPertenece(texto(s), comandos(g))) \wedge (esComandoRanking(texto(s)) \vee esComandoBaja(texto(s)))$;

`aux esComandoBaja` (t: String) : Bool $= extraerComando(t) ==$ "$BAJA$" ;

`aux esComandoRanking` (t: String) : Bool $= extraerComando(t) ==$ "$RANKING$" ;

`aux esJugador` (u: Usuario, g: Gateway, s: SMS) : Bool $= u \in participantes(triviaXKeyword(trivias(g), extraerKeywordSMS(s)))$;

`aux esJugadorGanador` (u: Usuario, g: Gateway, s: SMS) : Bool $= u \in participantes(triviaXKeyword(trivias(g), extraerKeywordSMS(s))) \wedge$
$u \in ganadores(triviaXKeyword(trivias(g), extraerKeywordSMS(s)))$;

`aux esJugadorLeFaltanVarias` (u: Usuario, g: Gateway, s: SMS) : Bool $=$
$u \in participantes(triviaXKeyword(trivias(g), extraerKeywordSMS(s))) \wedge$
$leFaltaResponderVarias(triviaXKeyword(trivias(g), extraerKeywordSMS(s)), u)$;

`aux esJugadorYLeFaltaUna` (u: Usuario, g: Gateway, s: SMS) : Bool $= u \in participantes(triviaXKeyword(trivias(g), extraerKeywordSMS(s))) \wedge$
$leFaltaResponderSoloUna(triviaXKeyword(trivias(g), extraerKeywordSMS(s)), u)$;

`aux esMensajeBaja` (s: SMS, u: Usuario) : Bool $= nroDestino(s) == u \wedge texto(s) ==$ "Listo, ya te borramos" ;

`aux esMensajeDeSeguiPagando` (s: SMS, u: Usuario) : Bool $= nroDestino(s) == u \wedge texto(s) ==$ "Gracias por seguir pagando pero el juego termino" ;

`aux esMensajeFinal` (s: SMS, t: Trivia, u: Usuario) : Bool $=$
$nroDestino(s) == u \wedge texto(s) ==$ "Has respondido todas las preguntas. " $++ textoPuntaje(t, u)$;

`aux esMensajeNoTeEntiendo` (s: SMS, u: Usuario) : Bool $= nroDestino(s) == u \wedge texto(s) ==$ "No te entiendo" ;

`aux esMensajePrimerPregunta` (s: SMS, t: Trivia, u: Usuario) : Bool $=$
$nroDestino(s) == u \wedge texto(s) ==$ "Hola! Tenes 0 pts" $++ texto(preguntas(t)_0)$;

`aux esMensajeProximaPregunta` (o: Trivia, r: String, s: SMS, u: Usuario, t: Trivia) : Bool $=$
$nroDestino(s) == u \wedge texto(s) == bienOMal(o, u, r) ++ textoPuntaje(t, u) ++ texto(proxPregunta(t, u))$;

`aux esMensajeRanking` (smsOrig, smsRta: SMS, u: Usuario, g: Gateway) : Bool $= nroDestino(smsRta) == u \wedge$
$(\exists t \leftarrow trivias(g), keywordDeComando(texto(smsOrig)) \in keywords(t))$
$texto(smsRta) ==$ "Tu posicion es: " $++ intToString(posicionEnTrivia(t, u))$;

`aux extraerComando` (t: String) : Keyword $= primerPalabra(quitarEspacios(t))$;

`aux extraerComandoDeSMS` (s: SMS) : Keyword $= primerPalabra(quitarEspacios(texto(s)))$;

`aux extraerKeyword` (s: String) : Keyword $= primerPalabra(quitarEspacios(s))$;

`aux extraerKeywordSMS` (s: SMS) : Keyword $= primerPalabra(quitarEspacios(texto(s)))$;

8.4. G

`aux gatewayDespuesDeBaja` (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS, SMS)) : Bool $=$
$(extraerComandoSMS(s) \in comandos(original) \wedge esComandoBaja(texto(s))) \rightarrow$
$(nuevoGwPosBaja(original, nuevo, obtenerKeywordDeComando(texto(sms)), usr)f \wedge$
$prm(ms) == sms \wedge esMensajeBaja(sgd(ms), usr))$;

`aux gatewayDespuesDeComando` (original: Gateway, nuevo: Gateway, s: SMS, u: Usuario, ms: (SMS, SMS)) : Bool $=$
$gatewayDespuesDeRanking(original, nuevo, s, u, ms) \wedge gatewayDespuesDeBaja(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeOtroComando(original, nuevo, s, u, ms)$;

`aux gatewayDespuesDeKeywordInvalida` (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS,SMS)) : Bool $=$
$(\neg primerPalabraPertenece(texto(sms), comandos(g)) \wedge noEsKeyword(texto(sms), trivias(original)) \rightarrow$
$(original == nuevo \wedge esMensajeNoTeEntiendo(sgd(ms), usr) \wedge prm(ms) == sms)$;

`aux gatewayDespuesDeKeywordYJugadorGanador` (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS,SMS)) : Bool $=$
$(\neg primerPalabraPertenece(texto(sms), comandos(g)) \wedge esJugadorGanador(usr, orginal, sms)) \rightarrow$
$(original == nuevo \wedge prm(ms) == sms \wedge esMensajeDeSeguiPagando(sgd(ms), usr))$;

aux gatewayDespuesDeKeywordYJugadorNuevo (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS,SMS)) : Bool =
$(\neg primerPalabraPertenece(texto(sms), comandos(g)) \wedge \neg esJugador(usr, orginal, sms)) \rightarrow$
$(nuevoGwConJugadorNuevo(original, nuevo, extraerKeywordSMS(sms), usr) \wedge$
$prm(ms) == sms \wedge esMensajePrimerPregunta(sgd(ms), triviaXKeyword(trivias(nuevo), extraerKeyword(s)), usr))$;

aux gatewayDespuesDeKeywordYJugadorNuevoGanador (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS,SMS)) : Bool =
$(\neg primerPalabraPertenece(texto(sms), comandos(g)) \wedge \neg esJugadorYLeFaltaUna(usr, orginal, sms)) \rightarrow$
$(nuevoGwConNuevoGanador(original, nuevo, sms, usr) \wedge$
$prm(ms) == sms \wedge esMensajeFinal(texto(sms), sgd(ms), triviaXKeyword(trivias(nuevo), extraerKeywordSMS(sms)), usr))$;

aux gatewayDespuesDeKeywordSinGanador (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS,SMS)) : Bool =
$(\neg primerPalabraPertenece(texto(sms), comandos(g)) \wedge \neg esJugadorLeFaltanVarias(usr, orginal, sms)) \rightarrow$
$(nuevoGwSinNuevoGanador(original, nuevo, sms, usr) \wedge$
$prm(ms) == sms \wedge esMensajeProximaPregunta(triviaXKeyword(trivias(original), extraerKeywordSMS(sms)), texto(sms), sgd(ms), usr, triviaX$
aux gatewayDespuesDeOtroComando (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS, SMS)) : Bool =
$(\neg esComando(texto(sms), original) \vee comandoConKeywordInvalida(texto(sms), trivias(original), u)) \rightarrow$
$(original == nuevo \wedge esMensajeNoTeEntiendo(sgd(ms), usr) \wedge prm(ms) == sms)$;

aux gatewayDespuesDeProcesarMensaje (original: Gateway, nuevo: Gateway, s: SMS, u: Usuario, ms: (SMS, SMS)) : Bool =
$gatewayDespuesDeComando(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeKeywordInvalida(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeKeywordYJugadorNuevo(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeKeywordYJugadorGanador(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeKeywordYJugadorNuevoGanador(original, nuevo, s, u, ms) \wedge$
$gatewayDespuesDeKeywordSinGanador(original, nuevo, s, u, ms)$;

aux gatewayDespuesDeRanking (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario, ms: (SMS, SMS)) : Bool =
$(extraerComandoSMS(sms) \in comandos(original) \wedge esComandoRanking(texto(sms))) \rightarrow$
$(original == nuevo \wedge prm(ms) == sms \wedge esMensajeRanking(s, sgd(ms), usr, nuevo))$;

aux gatewayXNro (t: TelCO, n: Numero) : Gateway $= [g | g \leftarrow gateways(t), numero(g) == n]_0$;

8.5. H

aux hayGatewayConNro (nro:$\mathbb{Z}$, gs: [Gateway]) : Bool $= (\exists g \leftarrow gs) numero(g) == nro$;

8.6. I

aux igualesConNuevoGanador (t: Trivia, r: Trivia, u: Usuario, x: String) : Bool =
$mismos(keywords(t), keywords(r)) \wedge preguntas(t) == preguntas(r) \wedge$
$mismos(participantes(t), participantes(r)) \wedge ganadores(t) + +[u] == ganadores(r) \wedge$
$(\forall k \leftarrow participantes(r), k \neq u) puntajeAcumulado(r, k) == puntajeAcumulado(t, k) \wedge$
$(\forall k \leftarrow participantes(r), k \neq u \wedge k \notin ganadores(r)) proximaPregunta(r, k) == proximaPregunta(t, k) \wedge$
$puntajeAcumulado(r, u) == puntajeAcumulado(t, u) + puntosQueSuma(t, u, x)$;

aux igualesConNuevoPuntaje (t: Trivia, r: Trivia, u: Usuario, x: String) : Bool =
$mismos(keywords(t), keywords(r)) \wedge preguntas(t) == preguntas(r) \wedge$
$mismos(participantes(t), participantes(r)) \wedge ganadores(t) == ganadores(r) \wedge$
$(\forall k \leftarrow participantes(r), k \neq u) puntajeAcumulado(r, k) == puntajeAcumulado(t, k) \wedge$
$(\forall k \leftarrow participantes(r), k \neq u \wedge k \notin ganadores(r)) proximaPregunta(r, k) == proximaPregunta(t, k) \wedge$
$puntajeAcumulado(r, u) == puntajeAcumulado(t, u) + puntosQueSuma(t, u, x) \wedge proxPregunta(r, u) == preguntas(r))_{indicePreg(t,u)+1}$;

aux igualesSalvoUsuario (t: Trivia, r: Trivia, u: Usuario) : Bool =
$mismos(keywords(t), keywords(r)) \wedge preguntas(t) == preguntas(r) \wedge$
$mismos(participantes(t), u : participantes(r)) \wedge sacarUsuario(ganadores(t), u) == ganadores(r) \wedge$
$(\forall k \leftarrow participantes(r)) puntajeAcumulado(r, k) == puntajeAcumulado(t, k) \wedge$
$(\forall k \leftarrow participantes(r), k \notin ganadores(r)) proximaPregunta(r, k) == proximaPregunta(t, k)$;

aux igualesConUsuario (t: Trivia, r: Trivia, u: Usuario) : Bool $= mismos(keywords(t), keywords(r)) \wedge preguntas(t) == preguntas(r) \wedge$
$mismos(u : participantes(t), participantes(r)) \wedge ganadores(t) == ganadores(r) \wedge$
$(\forall k \leftarrow participantes(t)) puntajeAcumulado(r, k) == puntajeAcumulado(t, k) \wedge$
$(\forall k \leftarrow participantes(t), k \notin ganadores(t)) proximaPregunta(r, k) == proximaPregunta(t, k) \wedge$
$puntajeAcumulado(r, u) == 0 \wedge proximaPregunta(r, u) == preguntas(r)_0$;

aux indiceEnGanadores (u: Usuario, t: Trivia) : $\mathbb{Z} = [i | i \leftarrow [0..|ganadores(t)), ganadores(t)_i == u]_0$;

aux indicePreg (t: Trivia, u: Usuario) : $\mathbb{Z} = [i | i \leftarrow [0..|preguntas(t)|), preguntas(t)_i == proxPregunta(t, u)]_0$;

aux interseccion (a, b: [T]) : [T] $= [x | x \leftarrow a, x \in b]$;

8.7. L

aux lasMasRecurrentes (xs:[String]) : [String] $= [x | x \leftarrow xs, (\forall y \leftarrow xs) cantAp(y, xs) \leq cantAp(x, xs)]$;

aux leFaltaResponderSoloUna (t: Trivia, u: Usuario) : Bool =
$u \notin ganadores(t) \wedge proxPregunta(t, u) == ultimo(preguntas(t))$;

aux leFaltaResponderVarias (t: Trivia, u: Usuario) : Bool =
$u \notin ganadores(t) \wedge proxPregunta(t, u) \neq ultimo(preguntas(t))$;

aux losQueMasPuntosTienen (t: Trivia) : [Usuario] $= [u | u \leftarrow participante(t), tieneMasPuntosQueTodos(t, u)]$;

aux losQueMenosMandaron (t: Trivia, us: [Usuario]) : [Usuario] $= [u | u \leftarrow us, mandoMenosQueLosDemas(t, u, us)]$;

8.8. M

aux mandoMenosQueLosDemas (t: Trivia, u: Usuario, us: [Usuario]) : Bool $= (\forall p \leftarrow us) indicePreg(t,u) \leq indicePreg(t,p)$;

8.9. N

aux noEncuentraTrivia (k: String, ts: [Trivia]) : Bool $= (\forall t \leftarrow ts) k \notin keywords(t)$;

aux noEsKeyword (k: String, ts: [Trivia]) : Bool $= noEncuentraTrivia(primerPalabra(quitarEspacios(s)), ts)$;

aux noEsTriviaDelUsuario (ts: [Trivia], u: Usuario) : Bool $= (\forall t \leftarrow ts) u \notin participantes(t)$;

aux nuevoGwConJugadorNuevo (original: Gateway, nuevo: Gateway, key: Keyword, usr: Usuario) : Bool =
$(|trivias(original)| == |trivias(nuevo)|) \wedge$
$mismos(triviasDeOtrasKeywords(key, original), triviasDeOtrasKeywords(key, trivias(nuevo))) \wedge$
$(\exists t \leftarrow trivias(original), key \in keywords(t))(\exists r \leftarrow trivias(nuevo), key \in keywords(r))(igualesConUsuario(t, r, usr)) \wedge$
$numero(original), numero(nuevo) \wedge mismos(comandos(original), comandos(nuevo))$;

aux nuevoGwConNuevoGanador (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario) : Bool =
$(|trivias(original)| == |trivias(nuevo)|) \wedge$
$mismos(triviasDeOtrasKeywords(extraerKeywordSMS(sms), trivias(original)),$
$triviasDeOtrasKeywords(extraerKeywordSMS(sms), trivias(nuevo))) \wedge$
$(\exists t \leftarrow trivias(original), extraerKeywordSMS(sms) \in keywords(t))(\exists r \leftarrow trivias(nuevo), extraerKeywordSMS(sms) \in keywords(r))$
$(igualesConNuevoGanador(t, r, usr, texto(sms)) \wedge numero(original) == numero(nuevo) \wedge mismos(comandos(original), comandos(nuevo)))$;

aux nuevoGwPosBaja (original: Gateway, nuevo: Gateway, key: Keyword, usr: Usuario) : Bool =
$(|trivias(original)| == |trivias(nuevo)|) \wedge$
$mismos(triviasDeOtrasKeywords(key, original), triviasDeOtrasKeywords(key, trivias(nuevo))) \wedge$
$(\exists t \leftarrow trivias(original), key \in keywords(t))(\exists r \leftarrow trivias(nuevo), key \in keywords(r)) igualesSalvoUsuario(t, r, usr) \wedge$
$numero(original) == numero(nuevo) \wedge mismos(comandos(original), comandos(nuevo))$;

aux nuevoGwSinNuevoGanador (original: Gateway, nuevo: Gateway, sms: SMS, usr: Usuario) : Bool =
$(|trivias(original)| == |trivias(nuevo)|) \wedge$
$mismos(triviasDeOtrasKeywords(extraerKeywordSMS(sms), trivias(original)),$
$triviasDeOtrasKeywords(extraerKeywordSMS(sms), trivias(nuevo))) \wedge$
$(\exists t \leftarrow trivias(original), extraerKeywordSMS(sms) \in keywords(t))(\exists r \leftarrow trivias(nuevo), extraerKeywordSMS(sms) \in keywords(r))$
$(igualesConNuevoPuntaje(t, r, usr, texto(sms)) \wedge numero(original) == numero(nuevo) \wedge mismos(comandos(original), comandos(nuevo)))$;

8.10. P

aux primerPalabra (t: String) : String $= [t_i | i \leftarrow [0..|t|), \text{' '} \notin t[0..i) \wedge t_i \neq \text{' '}]$;

aux primerPalabraPertenece (t: String, cs: [Keyword]) : Bool $= primerPalabra(quitarEspacios(t)) \in cs$;

aux posicionEnTrivia (t: Trivia,u: Usuario) : $\mathbb{Z}$ = if $u \in ganadores(t)$ then $indiceEnGanadores(u,t)$ else $|ganadores(t)| + |tienenMasPuntos(t,u)|$;

aux puntosQueSuma (t: Trivia, u: Usuario, r: Texto) : $\mathbb{Z}$ =
$puntaje(proxPregunta(t,u)) * \beta(rtaCorrecta(proxPregunta(t,u)) == sacarPrimerPalabra(r))$;

8.11. O

aux obtenerKeywordDeComando (t: String) : Keyword $= segundaPalabra(quitarEspacios(t))$;

8.12. Q

aux quitarEspacios (t: String) : String $= [x_i | i \leftarrow [0..|t|), x_i \neq \text{' '}]$;

8.13. S

aux sacarPrimerPalabra (t: String) : String $= [t_i | i \leftarrow (|primerPalabra(t)|..|t|)]$;

aux sacarRepetidos (a: [T]) : [T] $= [a_i | i \leftarrow [0..|a|), a_i \notin a[0..i)]$;

aux sacarUsuario (us: [Usuario], u: Usuario) : [Usuario] $= [x | x \leftarrow us, x \neq u]$;

aux seAnotoEnTodas (g: Gateway, u: Usuario) : Bool $= (\forall t \leftarrow trivias(g)) u \in participantes(t)$;

aux segundaPalabra (t: String) : String $= primerPalabra(sacarPrimerPalabra(t))$;

aux soloMayoresAN (n:$\mathbb{Z}$, xs: [String]) : [String] $= [x | x \leftarrow xs, |x| \geq n]$;

aux sinRepetidos (a: [T]) : Bool $= (\forall i, j \leftarrow [0..|a|), i \neq j) a_i \neq a_j$;

8.14. T

aux terminoAlguna (g: Gateway, u: Usuario) : Bool $= (\exists t \leftarrow trivias(g)) u \in ganadores(t)$;

aux textoPuntaje (t: Trivia, u: Usuario) : Texto $= \text{"}Tenes\text{"} + + puntajeAcumulado(t,u) + + \text{"}pts.\text{"}$;

aux tieneMasPuntosQueTodos (t: Trivia, u: Usuario) : Bool =
$(\forall p \leftarrow participantes(t)) puntajeAcumulado(t,u) \geq puntajeAcumulado(t,p)$;

`aux tienenMasPuntos` (t: Trivia, u: Usuario) : [Usuario] $= [p | p \leftarrow participantes(t), p \notin ganadores(t) \wedge puntajeAcumulado(t, p) > puntajeAcumulado(t, u)]$;

`aux todosLosUsuarios` (g: Gateway) : [Usuario] $= sacarRepetidos([u | t \leftarrow trivias(g), u \leftarrow participantes(t)])$;

`aux triviasDeOtrasKeywords` (k: String, ts: [Trivia]) : [Trivia] $= [t | t \leftarrow ts, k \notin keywords(t)]$;

`aux triviaXKeyword` (ts: [Trivia], k: String) : Trivia $= [x | x \leftarrow ts, k \in keywords(x)]_0$;

8.15. U

`aux ultimo` (a: [T]) : T $= a_{|a|-1}$;