Notations MODIFS EN COURS : ASSERTIONS A FAIRE

* Eléments syntaxiques pris du méta-modèle
* Δ : environnement contenant les types déclarés
  + Forme indéterminée pour l’instant
  + Construction indéterminée pour l’instant
* Γ : environnement contenant les types des variables
* Φ : environnement contenant les fonctions déclarées
* Σ : environnement contenant les systèmes déclarés
* Μ : environnement contenant les médiateurs déclarés
* Κ : environnement contenant les noms des constituants
* P : listes des types des paramètres formels d'un behavior
* φ : type d’une fonction
* σ : type d’un système
* μ : type d’un médiateur
* α : type d’une architecture
* τ : type d’une expression
* τ1 <: τ2 : τ1 sous-type de τ2 (inclus le cas τ1 même type que τ2)
* V : liste de Valuing

Jugements

* Δ,Φⱶd bien typé : dans l’environnement Δ,Φ le type d est bien typé
* Δ,Φⱶf:φ : dans l’environnement Δ,Φ la fonction f a le type φ
* Δ,Φⱶs:σ : dans l’environnement Δ,Φ le système s est de type σ
* Δ,Φⱶm:μ : dans l’environnement Δ,Φ le médiateur m est de type μ
* Δ,Φ,Σ,Μⱶa:α : dans l’environnement Δ,Φ,Σ,Μ l’architecture a est de type α
* Δ,Φⱶe:τ : dans l’environnement Δ,Φ l’expression e est de type τ

D'après le diagramme SoSADL

D'après le diagramme EntityBlock

D'après le diagramme DataType

→ ajouter la condition *vmin*<=*vmax*? Non parce que ce n'est pas décidable.

D'après le diagramme FunctionDecl

→ c'est la règle lorsque *type* ET *expression* sont donnés.

→ Valuing avec type non donné et expression donnée.

Note : l'expression est toujours donnée (imposée par la grammaire concrète).

→ Valuing vide (tous les Valuing ont été traités, [] = liste vide)

D'après le diagramme SystemDecl

D'après le diagramme MediatorDecl

D'après le diagramme GateAndDutyDecl

D'après le diagramme ArchitectureDecl

D'après le diagramme ArchBehaviorDecl

→ cette règle est à revoir

D'après le diagramme Behavior

→ le repeat ne peut pas être suivi d'un statement

→ le choose ne peut pas être suivi d'un statement

→ IfThenElse sans else

→ IfThenElse ne peut pas être suivi d'un statement.

→ IfThenElse avec else

→ IfThenElse ne peut pas être suivi d'un statement.

→ RecursiveCall ne peut pas être suivi d'un statement.

→ est-ce que name intervient dans le typage ? Quelle est sa sémantique ?

→ est-ce que AskAssertion lie des variables, et si oui lesquelles ?

→pour l'instant E n'est pas défini :

* faut-il typer l' « l'environnement » qui contient les contraintes ? Auquel cas cela pourrait être E
* sinon, il faut introduire une variable de type pour chaque nom. Mais dans la pratique cela sera-t-il vraiment utilisable ?
  + Par exemple « ask toto is (x=y) » ne permettra de conclure rien d'autre que le fait que x et y sont du même type. Cependant les autres instructions pourraient fournir suffisamment de contraintes pour déterminer ce type.
  + Par exemple « ask toto is (x=y) value z = x+1 » : rien ne permettrait de s'assurer que les types correspondent bien au tell correspondant « value x = sequence{1} tell toto is x=y ». Cet exemple contient un problème de types.

→ il manque TellAssertion

D'après le diagramme Expression

→ connLeft (resp. connRight) peuvent être : soit une connexion de gate, soit une connexion de constituant, soit une connexion d'un ensemble de constituants... et dans chaque cas : in, out ou inout...

Q1 : les multiplicités ont-elles un sens dans tous les cas d'unification possibles ?

Q2 : relations de sous-typage dans chaque cas possible ?

Q3 : unification possible entre in et inout ? Ou et inout ?

→ unifications autorisées?

- gate-gate , gate-système(s), gate-médiateur(s), système(s)-médiateur(s),

- système(s)-système(s), médiateur(s)-médiateur(s).

→ on a choisi de séparer l'espace de nom des constituants (Κ) de l'espace de nom des variables (Γ).

→ on a choisi de conserver la quantité (un ou plusieurs) dans le type des constituants, et de n'autoriser la quantification que lorsqu'il y en a plusieurs.

→ question : intérêt de « div » par rapport à « / » ? supprimer « / » (ou « div » ?) de la grammaire ?

→ \*, / et mod à faire.

→ A REVOIR ! object a un type déclaré, method existe dans l'environnement des fonctions (= méthodes) du type de object, les paramètres (effectifs) sont bien typés, le type de l'expression MethodCall() est le type de cette méthode.

→ object est de type Sequence(τ) qui est déclaré, expression est du type τe, le type de l'expression Map(object,expression) est donc de type Sequence(τe ).

→ object a un type déclaré, field est un nom qui existe dans l'environnement des variables et des fonctions du type de l'objet, le type de ce field est τ, le type de l'expression Field(object,field) est donc le type du field.

D'après le diagramme Assertion

→ les actions « via complexName send/receive any » sont autorisées dans les assertions :

- Q1 : les actions sont-elles ordonnées dans une assertion (doit-on en tenir compte) ?

- Q2 : complexName peut venir d'un import (cf. wsnMonitoringSos).

→ faire \*, / et mod.

→ question : intérêt de « div » par rapport à « / » ? supprimer « div » ?

D'après le diagramme Protocol

→ contrairement à un behavior, un protocol ne déclare pas de paramètres formels.

→ contrairement au « repeat » du Behavior, le « repeat » du Protocol peut être suivi d'autres statements.

→ IfThenElseProtocol sans else

→ IfThenElseProtocol avec else

→ le Assert de Protocol est le même que dans Behavior donc même question : est-ce que name intervient dans le typage de Assert ? Quelle est sa sémantique ?

→ AskAssertion est le même que dans Behavior, donc mêmes questions.