

Summary of Project 2

10/15/2015

1. Implementation Descriptions And Explanations:

(a) [1.8 Definition]: Naming convention for attributes used for both synthesized and inherited process is to separate them as two different attributes, where the one starting with letter “s” means synthesized and the other starting with “i” inherited. For example: rt becomes \uparrow srt and \downarrow irt, ids becomes \uparrow sids and \downarrow iids.

(b) Scheme containing substring “Add” means adding the inherited attribute to some sort. Usually in the form of “S_Add_A”, where “S” means some kind of sort and “A” means the inherited attribute.

For example, DeclarationAddTe, StatementsAddIrt.

(c) [1.8 Definition]: Attribute ids is defined using new syntactic sort, which is a list of IDENTIFIER. Implementing detail below:

```
// use new syntactic sort to define list of member names [2]1.8
sort IdList | [ (IDENTIFIER) (IdListTail) ] | [];

sort IdListTail | [ , (IDENTIFIER) (IdListTail) ] | [];

attribute  $\uparrow$ sids(IdListTail);
```

(d) Schemes in the form of “Convert_A_to_B” usually means turning one syntactic sort to another, where “A” denotes the input sort and B denotes the result sort. For example, ConvertNTLTtoNTL, ConvertTLTtoTL.

(e) [1.11 Semantic Operations]: Implementation Detail of Eq where Type contains a NameTypeList. Eq will invoke a EqHelper functions taking the two NameTypeLists. The way to judge whether they are equal are using idea similar to set. If $A \subseteq B$ and $B \subseteq A$, we know that $A = B$. So EqHelper runs twice to and use IsInList helper function to traverse both two NameTypeList and returns the result.

(f) Schemes starting with TestXXXX means they are used for testing code snippets. For example, the one used to test the attribute \uparrow ds of Units. All tests are at the bottom of the .hx file.

```
// Testing Unit $\uparrow$ ds
sort NameTypeListTail | scheme TestUnitDs(Unit);
TestUnitDs(#U  $\uparrow$ ds(#ds))  $\rightarrow$  #ds;
```

(g) There are new syntactic sort defined at the bottom of the .hx file, in order to test the helper schemes which takes more than two parameters. Example show below:

```
// define new syntactic sort to test Eq
sort TypeConsType | [ (Type): (Type) ];

sort Bool | scheme TestEq(TypeConsType);
TestEq([ (Type#1): (Type#2) ]) → Eq(#1, #2);
```

(h) Scheme MergeTTL and Scheme ConvertToTLT are used to derived the attribute ↑ts. Scheme ConvertToTLT is used to convert TypeList to TypeListTail. Then MergeTTL merge a Type and TypeList as a new TypeList, which is the value of attribute ↑ts.

```
sort TypeList | scheme MergeTTL(Type, TypeList);
MergeTTL(#1, []) → [ (Type#1) ];
MergeTTL(#1, #2) → [ (Type#1) (TypeListTail ConvertToTLT(#2)) ];

sort TypeListTail | scheme ConvertToTLT(TypeList);
ConvertToTLT([]) → [];
ConvertToTLT([ (Type#1) (TypeListTail#2)]) → [ , (Type#1) (TypeListTail#2) ] ;
```

(g) About synthesizing attribute ok. Since ok is synthesized after te immersed so ok is also evaluated using helper schemes, which invoke relevant Sort_AddTe scheme and then compute ok and propagate it up to the root. Idea similar to hacs manual page 25, where Se scheme uses SeB scheme as a helper. So for BAddTe(B) → X will then be wrapped using ok scheme. So it becomes BAddTe(B) → BSynOk(X). Because of time limitation, the one implemented in the .hx file is directly setting to true, which leads to success of Check scheme.

(h) Scheme helper Eq(Type1, Type2). It is implemented by first enumerating all the cases of Type1. Then using [data #], it evaluates Type2, which represents by #, then calling helper to see if Type1 and Type1 are exactly the same. This avoids the situation where Type2 is a function and its return type is Type1.

2. Problems Encountered:

(a) Semantic data sort vs Syntactic sort: Decomposing Type and rebuild it using the semantic data sort. Show below.

```

sort T
| Boolean
| String
| Number
| Void
| Interface(IDENTIFIER)
| Callable(ListOfT, T)
| NTList(NTMaps)
;

sort T | scheme ApplyType(Type) ;
ApplyType( [boolean] ) → Boolean ;
ApplyType( [string] ) → String ;
ApplyType( [number] ) → Number ;
ApplyType( [void] ) → Void ;
ApplyType( [(IDENTIFIER#)] ) → Interface(#);
ApplyType( [ ( {TypeList #1} ) => {Type #2} ] ) → Callable(CollectTL(#1), ApplyType(#2)) ;
ApplyType( [ { (NameTypeList#1)} ] ) → NTList(CollectNTL(#1)) ;

sort NTMap | NT(IDENTIFIER, T) ;

sort NTMap | scheme CollectNT(NameType);
CollectNT( [ { IDENTIFIER#1 } : {Type#2} ] ) → NT(#1, ApplyType(#2)) ;

sort NTMaps
| Con(NTMap, NTMaps)
| None
;

sort NTMaps | scheme CollectNTT(NameTypeListTail) ;
CollectNTT( [ , (NameType#1) (NameTypeListTail#2) ] ) → Con(CollectNT(#1), CollectNTT(#2)) ;
CollectNTT( [ ] ) → None;

sort NTMaps | scheme CollectNTL(NameTypeList) ;
CollectNTL( [ {NameType#1} (NameTypeListTail#2) ] ) → Con(CollectNT(#1), CollectNTT(#2)) ;
CollectNTL([ ] ) → None ;

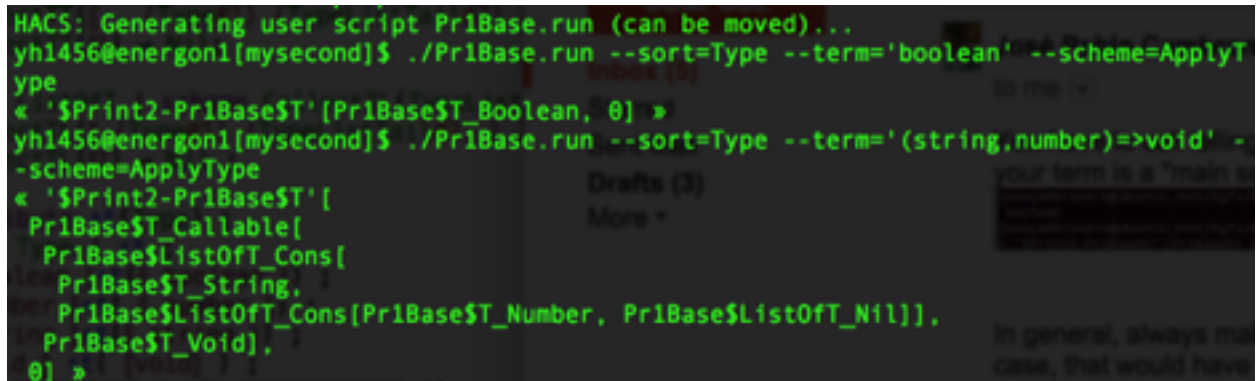
sort ListOfT
| Cons(T, ListOfT)
| Nil
;

sort ListOfT | scheme CollectTT(TypeListTail) ;
CollectTT( [ , (Type#1) (TypeListTail#2) ] ) → Cons(ApplyType(#1), CollectTT(#2)) ;
CollectTT([ ] ) → Nil;

```

Then the output is not printed properly. And the semantic data become more and more complicated, wrapping many layers outside one single value.

The solution is to use the original syntactic sort as the value of the attributes. Then apply something similar to pattern matching to the sort.



```
HACS: Generating user script Pr1Base.run (can be moved)...
yh1456@energon1[mysecond]$ ./Pr1Base.run --sort=Type --term='boolean' --scheme=ApplyType
< '$Print2-Pr1Base$T'[Pr1Base$T_Boolean, 0] >
yh1456@energon1[mysecond]$ ./Pr1Base.run --sort=Type --term='(string,number)=>void' --scheme=ApplyType
< '$Print2-Pr1Base$T'[
  Pr1Base$T_Callable[
    Pr1Base$ListOfT_Cons[
      Pr1Base$T_String,
      Pr1Base$ListOfT_Cons[Pr1Base$T_Number, Pr1Base$ListOfT_Nil]],
      Pr1Base$T_Void],
  0] >
```

(b) Recursion gets chained using helper scheme. See below. UnitsAddTeHelper tries to extend the attribute uds into te then after adding all ads it tries to pass the added attribute te to the subtree by calling UnitsAddTe on itself again. Then it got chained to itself by calling helper function again.

```
// U1.te = Us.te | Us2.te = Extend(Us.te, U1.uds)
UnitsAddTe([ (Unit#1) (Units#2) ] t#syn)
  → UnitsAddTeHelper([ (Unit UnitAddTe(#1)) (Units UnitsAddTe(#2)) ] t#syn);
// U1.te = Us.te
UnitsAddTe([ (Unit#1) ]) → [ (Unit UnitAddTe(#1))];

sort Units | scheme UnitsAddTeHelper(Units) tte;
UnitsAddTeHelper([ (Unit#1 tuds(#uds)) (Units#2) ] t#syn)
  → [ (Unit#1) (Units ExtendUUds(#2, #uds)) ] t#syn;

sort Units | scheme ExtendUUds(Units, NameTypeListTail) tte;
ExtendUUds(#Us t#syn, []) → [ (Units UnitsAddTe(#Us)) ] t#syn; // ?????
ExtendUUds([#Us t#syn, [ (IDENTIFIER#21):(Type#22) (NameTypeListTail#23)]]
  → ExtendUUds(#Us, #23) tte{#21:#22} t#syn;
```

Solution: Do it without helper function. Attribute uds is synthesized and doesn't have any dependencies upon others, so directly invoking ExtendUUds will be fine

```
UnitsAddTe([ (Unit#1 tuds(#uds)) (Units#2) ] t#syn)
  → [ (Unit UnitAddTe(#1)) (Units ExtendUUdsUs(#2, #uds)) ] t#syn;
// U1.te = Us.te
UnitsAddTe([ (Unit#1) ] t#syn) → [ (Unit UnitAddTe(#1)) ] t#syn;

sort Units | scheme ExtendUUdsUs(Units, NameTypeListTail) tte;
ExtendUUdsUs(#Us, [ , (IDENTIFIER#1) : (Type#2) (NameTypeListTail#3) ])
  → ExtendUUdsUs(#Us, #3) tte{#1:#2};
ExtendUUdsUs(#Us, []) → UnitsAddTe(#Us);
```

(c) Problems of adding GlobalDefs

GlobalDefs is a scheme that outputs a NameTypeListTail with constant inside. Show below:

```
sort NameTypeListTail | scheme GlobalDefs;
GlobalDefs→ [ , document: {body: {innerHTML:string}} , string : (string)⇒number, charAt: (string ⇒
, number)⇒ number, substr:(string, number, number) ⇒ string];[]
```

It somehow generates compiler error, illustrating segmentation fault. And because there is no specific line and column specified where it generates the compiler error, so now the .hx file comments the GlobalDef scheme.

Solution: As mentioned in the email, update the version to 1.1.9 and uncomment the GlobalDefs.

But then it turned out that 1.1.9 breaks something else, so downgraded to 1.1.6 and comment the GlobalDefs scheme. Then 1.1.10 comes out and upgraded again to add it into the initial environment.

(d) Recursion stops. DeclarationAddTe calls another helper scheme ExtendSsDs by passing all the needed value. But then the helper scheme only able to add the first the into the Statements#Ss.

```
sort Declaration | scheme DeclarationAddTe(Declaration) ite;
DeclarationAddTe([ interface (IDENTIFIER#1) { (Members#2) } ])
→ [interface (IDENTIFIER#1) { (Members MembersAddTe(#2)) }];
DeclarationAddTe([ function (IDENTIFIER#1) (CallSignature#2 rds(#ds)) { (Statements#3) } ])
→ ExtendSsDs(#3,#ds,#1,#2);

sort Declaration | scheme ExtendSsDs(Statements, NameTypeListTail, IDENTIFIER, CallSignature) ite;
ExtendSsDs(#Ss, [], #Id, #Cs) → [];
ExtendSsDs(#Ss, [ , (IDENTIFIER#21):(Type#22) (NameTypeListTail#23)], #Id, #Cs)
→ [ function (IDENTIFIER#Id) (CallSignature#Cs) (Statements#Ss ite{#21:#22})]]
```

Solution: Extend the helper scheme directly inside Statements. After adding all the te then passing it down to the subtree using the scheme StatementAddTe.

(e) attribute ok causing semantic data sort. Because ok is not fully implemented in the first pass. So it outputs that it needs ok attribute.

```
[yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitUds --sort=Unit --term]
='var a:string; '
« '$Print2-Pr2Base$NameTypeListTail'[
Pr2Base$NameTypeListTail_Delayed_TestUnitUds[
Pr2Base$Unit_Collect_uds[
Pr2Base$Unit__Statement_[
Pr2Base$Statement_Need_ds[
Pr2Base$Statement_Need_ok[
'Pr2Base$Statement_var_NameType_:'[
'Pr2Base$NameType__IDENTIFIER__Type_'[
{"$LineLocation" : STRING[1]; "$ColumnLocation" : STRING[5]} "a",
Pr2Base$Type_string]]]]]]].
0] »
```


So in order to test the ds attribute of Unit, just comments the part with attribute ok, and then the test scheme works fine.

(f) IsMember Doesn't work properly. Show below with old version code:

```
sort Bool | scheme IsMember(Type, IDENTIFIER) ite; // ite{IDENTIFIER : Type}
IsMember([IDENTIFIER#]), IDENTIFIER# ) → True;
IsMember([ { (IDENTIFIER#1) : (Type#2) (NameTypeListTail#3) } ], IDENTIFIER#1) // ite{#1:#t}
→ True;
IsMember([ { (IDENTIFIER#1) : (Type#2) (NameTypeListTail#3) } ], IDENTIFIER#4)//ite{#1:#t}
→ IsMember([ { (NameTypeList ConvertNTLTtoNTL(#3)) } ], IDENTIFIER#4);
IsMember(#1, IDENTIFIER#2) → False;
```

```
(yh1456@energon1[mysecond])$ ./Pr2Yu.run --scheme=TestIsMember --sort=TypeConsIdentifier --term='{
ac:number, bb:void, c:boolean, aa:string):c'
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] »
(yh1456@energon1[mysecond])$ ./Pr2Yu.run --scheme=TestIsMember --sort=TypeConsIdentifier --term='{
ac:number, bb:void, c:boolean, aa:string):bb'
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] »
```

Reason: Because hacs has it own way of rearranging rules, so it has a preference to pick the last rule instead of others. Set the result of the third rule as an error will see the result instead of just False.

```
yh1456@energon1[mysecond])$ ./Pr2Yu.run --scheme=TestIsMember --sort=TypeConsIdentifier --term='{
ac:number, bb:void, c:boolean, aa:string):bb'
Exception in thread "main" java.lang.RuntimeException: Error?: Undefined Type
    at net.sf.crsx.generic.GenericEvaluator.error(Unknown Source)
    at net.sf.crsx.generic.GenericEvaluator.compute(Unknown Source)
    at net.sf.crsx.generic.GenericEvaluator.contract(Unknown Source)
    at net.sf.crsx.generic.GenericRule.contract(Unknown Source)
    at net.sf.crsx.generic.GenericValuation.contract(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.contraction(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.normalizeChildren(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.normalize(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.normalize(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.loadDirective(Unknown Source)
    at net.sf.crsx.generic.GenericCRS.load(Unknown Source)
    at net.sf.crsx.run.Crsx.realMain(Unknown Source)
    at net.sf.crsx.run.Crsx.realMain(Unknown Source)
    at net.sf.crsx.run.Crsx.main(Unknown Source)
```

Solution: Adding all cases type has and set the recursion rule as the default one when the rest doesn't match.

```
sort Bool | scheme IsMember(Type, IDENTIFIER) ite; // ite{IDENTIFIER : Type}
IsMember([IDENTIFIER#]), IDENTIFIER# ) ite{#1: #t} → True;
IsMember([ { (IDENTIFIER#1) : (Type#2) (NameTypeListTail#3) } ], IDENTIFIER#1)
→ True;
IsMember([ boolean ], IDENTIFIER#) → False;
IsMember([ number ], IDENTIFIER#) → False;
IsMember([ string ], IDENTIFIER#) → False;
IsMember([ void ], IDENTIFIER#) → False;
IsMember([ (TypeList#1) => (Type#2) ], IDENTIFIER#3) → False;
IsMember([ { } ], IDENTIFIER#4)
→ False;
default IsMember([ { (IDENTIFIER#1) : (Type#2) (NameTypeListTail#3) } ], IDENTIFIER#4)
→ IsMember([ { (NameTypeList ConvertNTLTtoNTL(#3)) } ], IDENTIFIER#4);
```

(g) Eq picks wrong rule when it is called from scheme IsInList.

```
sort Bool | scheme Eq{Type, Type} <te>;
Eq{ [ boolean ], [ boolean ] } → True;
Eq{ [ number ], [ number ] } → True;
Eq{ [ string ], [ string ] } → True;
Eq{ [ void ], [ void ] } → True;
Eq{ [ { IDENTIFIER#1 }, [ { IDENTIFIER#2 } ] ] <te>{#1:#t1} <te>{#2:#t2} } → Eq{#t1,#t2};
Eq{ [ { (Type#11) (TypeListTail#12) } => (Type#13) ], [ { (Type#21) (TypeListTail#22) } => (Type#23) ] }
→ And(Eq{#13, #23}, And(Eq{#11,#21}, Eq{[(TypeList ConvertTLToTL(#12)) => (Type#13)], [(TypeList ConvertTLToTL(#22)) => (Type#23)] }));
Eq{ [ { (NameTypeList#1) }, [ { (NameTypeList#2) } ] }
→ And(EqHelper{#1,#2}, EqHelper{#2,#1});
default Eq{#, #} → error [for debug purpose];

yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsInList --sort=NameTypeConsNameTypeList --term='aa:
number:cc:void, bb:boolean,aa:string'
* '$Print2-Pr2Yu$Bool' [{"te" : A_te}Pr2Yu$Bool_Eq[Pr2Yu$Type_number, Pr2Yu$Type_string], 0] *
```

It regards string and number as IDENTIFIERS, then it tries to look up in the environment and then outputs the wrong message.

So the solution to it is to change hacs preference towards rule selections by adding other cases into the scheme.

3. Some Revisions on the Reference SDD

(a) [1.9 Definition]: Adding production $E \rightarrow E_1 \wedge E_2$ to the original SDD, where the relevant semantic rules are $E.ok =$

$E_1.ok \wedge E_2.ok \wedge Eq(E.te, \text{number}, E_1.t) \wedge Eq(E.te, \text{number}, E_2.t)$, $E_1.te = E_2.te = E.te$, $E.t = \text{number}$

(b) [1.8 Definition]: Adding synthesized attribute ts on ELT.

(c) [1.8 Definition]: Attribute rt are passed down from Declaration, then inherited into both Ss and S.

4. Some Test Result

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestDeclarationDs --sort=Declaration --term='function Id1 (a:number, b:string, c:number):void {2+3;}'  
  , Id1 : (  number   , string   , number   ) => void
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeListTailDs --sort=NameTypeListTail --term="  , Asss : string, B:string"  
  , Asss : string   , B : string  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeListTailDs --sort=NameTypeListTail --term=""
```

```
HALS: Generating user script Pr2Base.run (can be moved)...  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestCallSignatureDs --sort=CallSignature --term="(Asss:string, Bss:number):boolean"  
  , Asss : string   , Bss : number
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeDs --sort=NameType --term="Asss : number"  
  , Asss : number
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestMembersDs --sort=Members --term="Asss : string;"  
  , Asss : string  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestMembersDs --sort=Members --term="Asss : string;  
Bsss : number;"  
  , Asss : string   , Bsss : number
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="{ }"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="var A:number;"  
  , A : number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="2+2;"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term=";"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="if (0 < 1) {} else {}"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="while (4*7) ;"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="yyyh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="return aa+bb;"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementDs --sort=Statement --term="return;"
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitDs --sort=Unit --term="interface Id {aa:number;}"  
  , Id : { aa : number   }  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitDs --sort=Unit --term="{}"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitsDs --sort=Units --term="{}"  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitsDs --sort=Units --term="{} interface Id {cc:number;}"  
  , Id : { cc : number   }
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestMemberDs --sort=Member --term='Aa (a:number, b:string) : boolean {}'  
  , Aa : (  number   , string   ) => boolean
```


Test for t

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeT --sort=NameType --term="Asf : number"  
number
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestCallSignatureT --sort=CallSignature --term='(a:number, b:string) : boolean'  
( number , string ) => boolean
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='aabccc'  
string  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='55'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='!55'  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='~"werw"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='-"werw"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='+443'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='5*5*3'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='5/3'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe"%wer2'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe"-"wer2"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe"<="wer2"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe">=3'  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='33<"wer2"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe">"sd33"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"arwe"=="sd33"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='234!="sd33"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='234&"sd33"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='66 ^ 88'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='234 | "sd33"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='(3 | 4) && "sd33"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"aa" || "sd33"  
boolean  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"ab"?0:999'  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='"ab"?99'  
string  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='88=="aa"  
number  
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionT --sort=Expression --term='8={}'  
number  
yh1456@energon1[mysecond]$
```

Test for ts

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeListTailTs --sort=NameTypeListTail --term=' a:number, b:string, c:void'
number . string . void
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestNameTypeListTs --sort=NameTypeList --term=' aa:number, b:string, c:void'
number . string . void
```

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionListTs --sort=ExpressionList --term='4,5,6,"aa","ae3"'
number . number . number . string . string
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestExpressionListTailTs --sort=ExpressionListTail --term='5,"aaa",33'
number . string . number
```

Test for ok

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestStatementsOk --sort=Statements --term='{}'
« '$Print2-Pr2Base$Bool' [Pr2Base$Bool_True, 0] »
```

Test for uds

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitUds --sort=Unit --term='var a:string: '
. a : string
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestUnitUds --sort=Unit --term='interface Id { }'
```

Test for srt:

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=TestCallSignatureSrt --sort=CallSignature --term='(a:number):boolean'
boolean
```

Test for helpers:

Test for IsLValue:

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --tl
term="abw"
v '$Print2-Pr2Base$Bool' [Pr2Base$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --tl
term="3.ab"
v '$Print2-Pr2Base$Bool' [Pr2Base$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --tl
term="3+2"
```

Test for IsLValue:

```
... 19 more
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --term='abew'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --term='3.ab'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --term='3+2'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_False, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=IsLValue --sort=Expression --term='~"a"+2'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_False, 0] »
```

Test for DistinctFirst:

```
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=DistinctFirst --sort=NameTypeListTail --term=', a:number, b:string, c:boolean'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Base.run --scheme=DistinctFirst --sort=NameTypeListTail --term=', a:number, b:string, c:boolean, a:number'
« '$Print2-Pr2Base$Bool'[Pr2Base$Bool_False, 0] »
```

Test for IsEmpty:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=IsEmpty --sort=IdListTail --term=''
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=IsEmpty --sort=IdListTail --term=', aa,bb'
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] »
```

Test for IsMember:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsMember --sort=TypeConsIdentifier --term='{ac:number, bb:void, c:boolean, aa:string):c'
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] »
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsMember --sort=TypeConsIdentifier --term='{ac:number, bb:void, c:boolean, aa:string):cc'
« '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] »
```

Test for MemberType:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestMemberType --sort=TypeConsIdentifier --term='{ac:number, bb:void, c:boolean, aa:string):c'
boolean
```

Test for ReturnType:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestMemberType --sort=TypeConsIdentifier --term='{ac:number, bb:void, c:boolean, aa:string):c'
boolean
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=ReturnType --sort=Type --term='(number, string, void)> boolean'
boolean
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=ReturnType --sort=Type --term='number'
Exception in thread "main" java.lang.RuntimeException: Error?: No return type found
```

Test for IsArgument:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsArguments --sort=TypeConsTypeList --term='(number, string, boolean, boolean)=>void:number, string, boolean'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsArguments --sort=TypeConsTypeList --term='(number, string, boolean, boolean)=>void:number, string, boolean'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
```

Test for Eq:

```
RACS: Generating user script Pr2Yu.run (can be moved)...
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEq --sort=TypeConsType --term='(number, string)=>void:(number, string)=>void'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEq --sort=TypeConsType --term='(number, string)=>void:(number, string, boolean)=>void'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEq --sort=TypeConsType --term='(a:number, b:string):(a:number, b:string, c:void)'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEq --sort=TypeConsType --term='(a:number, b:string):(a:number, b:string)'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
```

Test for IsInList:

```
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsInList --sort=NameTypeConsNameTypeList --term='aa:number:cc:void, bb:boolean, aa:string'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestIsInList --sort=NameTypeConsNameTypeList --term='aa:number:cc:void, bb:boolean, aa:number'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
```

Test for EqHelper:

```
... 19 more
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEqHelper --sort=NTLConsNTL --term='a:number, b:string-a:number, b:string, c:void'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEqHelper --sort=NTLConsNTL --term='b:string, c:void, a:number-a:number, b:string, c:void'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_True, 0] x
yh1456@energon1[mysecond]$ ./Pr2Yu.run --scheme=TestEqHelper --sort=NTLConsNTL --term='b:string, c:void, a:number-a:number, b:string, d:void'
x '$Print2-Pr2Yu$Bool'[Pr2Yu$Bool_False, 0] x
```

Notes:

Changes to the version submitted on Friday morning.

- 1) Adding more test cases to the scheme helper and fixed some potential bugging helper schemes.
- 2) Adding the process of synthesizing ok attribute.

Current Problem of the .hx file.

There are still issues dealing with type which contains a identifier, because the ! prefix on it. So this somehow affects Eq helper scheme, some synthesized attributes on Expression and when doing the map constraint using te.

So when run the input sample, when there is an IDENTIFIER used as type, the program will have trouble and will dump all the information saying some sort need ds, ok or some other stuff. If the ! problem can be solved here, the type checker will work fine for most samples.