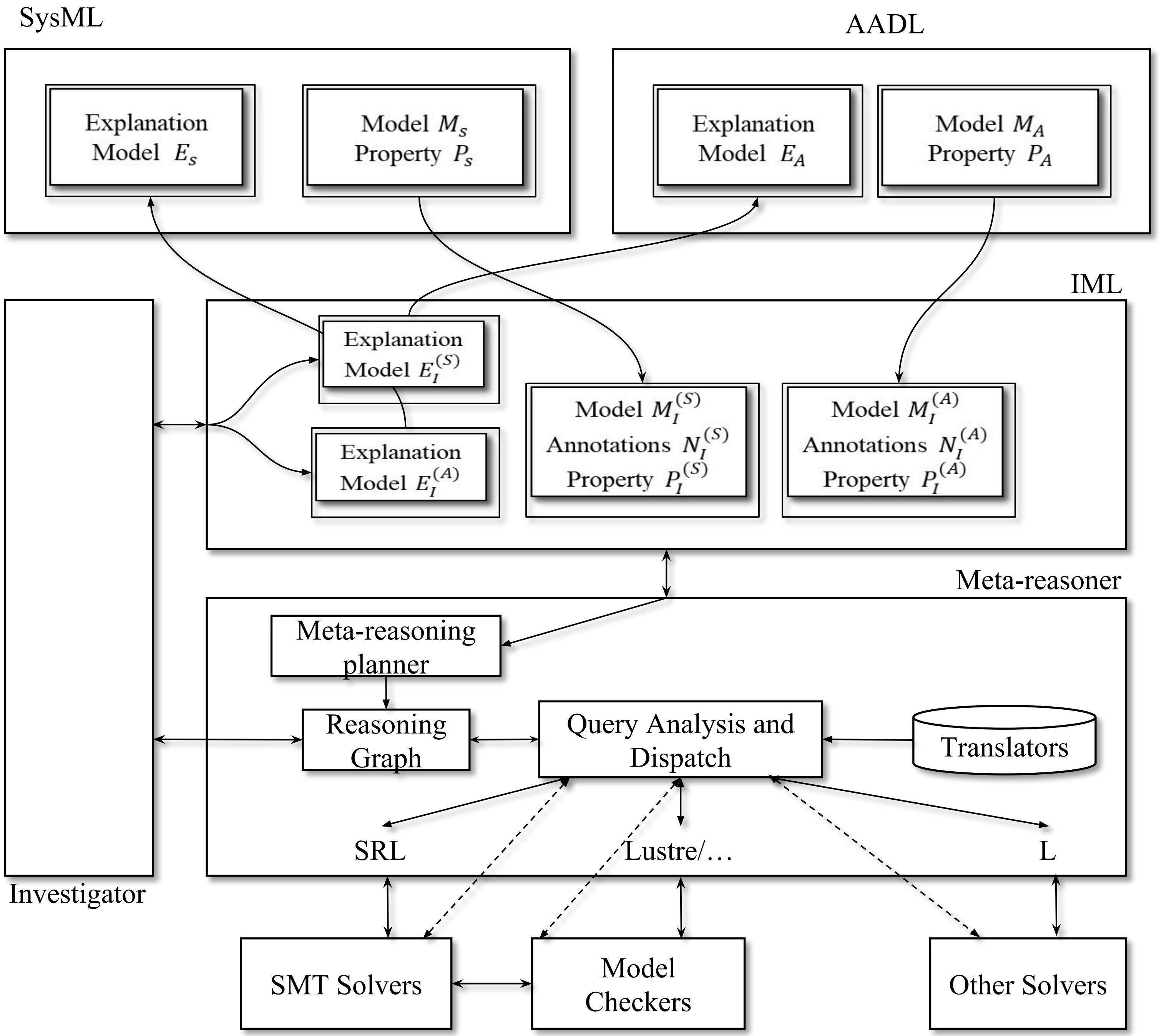
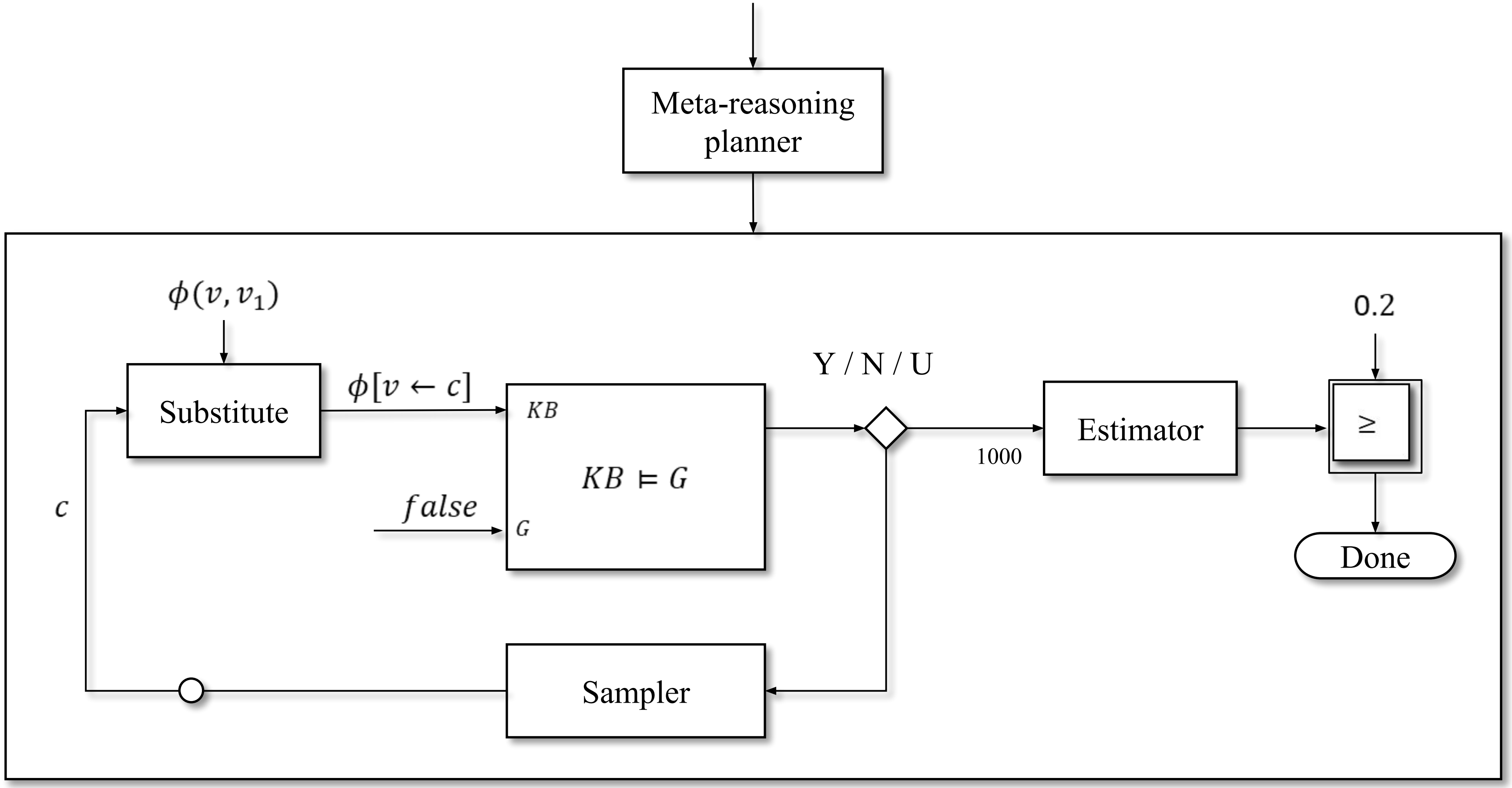
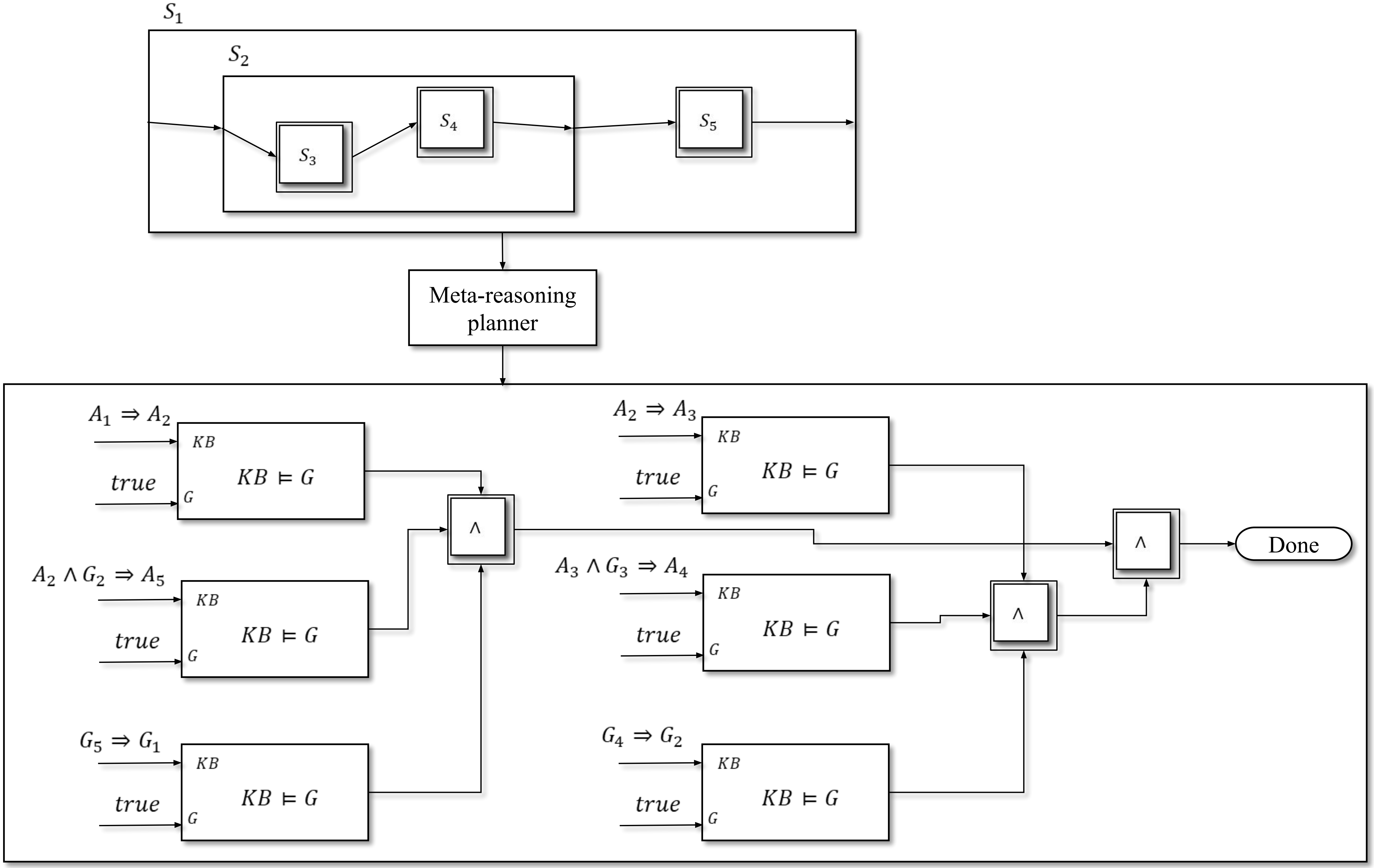
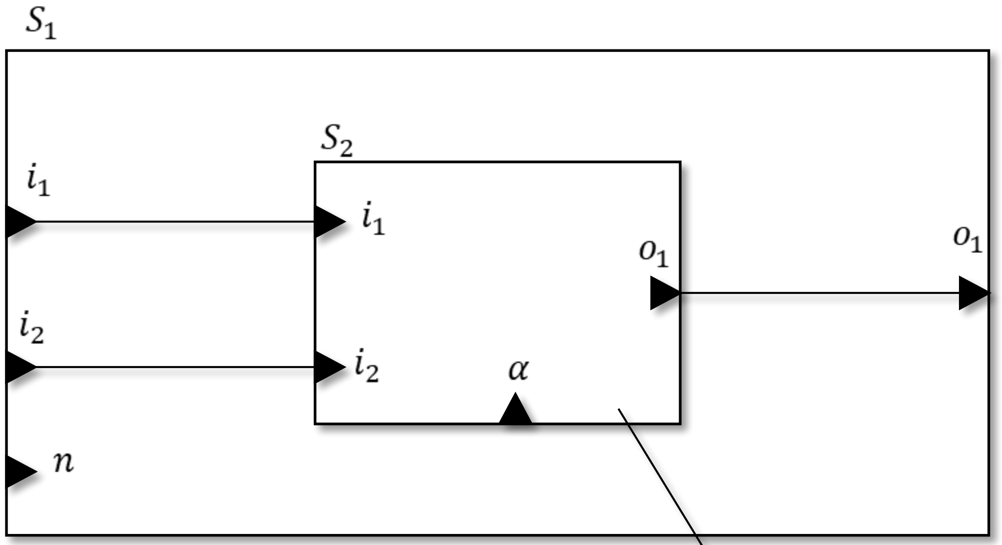

SysML
Explanation Model E_s
Model M_s Property P_s
AADL
Explanation Model E_A
Model M_A Property P_A
IML
Explanation Model $E_I^{(S)}$
Explanation Model $E_I^{(A)}$
Model $M_I^{(S)}$ Annotations $N_I^{(S)}$ Property $P_I^{(S)}$
Model $M_I^{(A)}$ Annotations $N_I^{(A)}$ Property $P_I^{(A)}$
Meta-reasoner
Meta-reasoning planner
Reasoning Graph
Query Analysis and Dispatch
Translators
SRL
Lustre/…
L
Investigator
SMT Solvers
Model Checkers
Other Solvers

```
v : Uniform<Int>              ;
v1 : Real                     ;
<<g:Goal>> P(ϕ(v,v1)) ≤ 0.2 ;
```




S_1
S_2
S_3
S_4
S_5
Meta-reasoning planner
$A_1 \Rightarrow A_2$
KB
$true$
G
$KB \vDash G$
$A_2 \wedge G_2 \Rightarrow A_5$
KB
$true$
G
$KB \vDash G$
$G_5 \Rightarrow G_1$
KB
$true$
G
$KB \vDash G$
$\wedge$
$A_2 \Rightarrow A_3$
KB
$true$
G
$KB \vDash G$
$A_3 \wedge G_3 \Rightarrow A_4$
KB
$true$
G
$KB \vDash G$
$G_4 \Rightarrow G_2$
KB
$true$
G
$KB \vDash G$
$\wedge$
$\wedge$
Done



Assumption of S_1

$n \geq 0 \wedge \left(\exists x, y.\ (0 \leq x \leq n \wedge 1 \leq y \leq n) \Rightarrow \left(|i_1| = \frac{x}{n} \wedge |i_2| = \frac{y}{n}\right)\right)$

Guarantee of S_1

$|o_1| \leq 1$

Simple spec for QAM modulation

Assumption of S_2

$|i_1| = 1 \wedge (|i_2| = 1 \vee i_2 = 0)$

Guarantee of S_1

$o_1 = i_1\sqrt{2}\sin(\alpha) + i_2\sqrt{2}\cos(\alpha)$

Simple spec for BPSK or QPSK modulation

S_1 can accept QAM constellations while S_2 only accepts BPSK and QPSK. The output must be between 1 and -1.

Notice that for n=1, S_2 is in fact a refinement of S_1.

```
package utrc::test1
public
with Base_Types;

system S1
features
i1: in data port Base_Types::Float;
i2: in data port Base_Types::Float;
o1: out data port Base_Types::Float;
n : in data port Base_Types::Integer ;
annex iml {**
 a1 <<a:Assume>> : Bool := exists x:Int, y:Int { (y >= 0 && y <= n && x>=1 && x
<= 0) => ( (i1 = x/n || i1 = -1 * x/n) && ( i2 = y/n || i2 =  -1 *y/n)) } ;
    g1 <<g:Guarantee>>: Bool := o1 <=1 && o1 >=-1;
**};
end S1;

system S2
features
i1: in data port Base_Types::Float;
i2: in data port Base_Types::Float;
o1: out data port Base_Types::Float;
alpha : in data port Base_Types::Float;
annex iml {**
a1 <<a:Assume>> : Bool := (i1 =1 || i1=-1) && (i2=0 || i2 =1 || i2 = -1) ;
g1 <<g:Guarantee>>: Bool := o1 = i1 * sqrt(2) * sin(alpha) + i2 * sqrt(2) *
cos(alpha);
**};
end S2;

system implementation S1.Impl
subcomponents
S2_sub : system S2 ;
connections
i1_TO_A : port i1 -> S2_sub.i1;
i2_TO_A : port i2 -> S2_sub.i2;
S2_TO_o1 : port S2_sub.o1 -> o1 ;
end S1.Impl;


end utrc::test1;
```

AADL MODEL

AADL package

```
package hermes.iml.aadl ;
import iml.lang.* ;
type Integer sameas Int;
type Float sameas Real ;
type Boolean sameas Bool ;
meta type system ;
meta type implementation ;
meta type in ;
meta type out;
meta type port;
meta type connection;
meta type subcomponent;
type Connection<type T> {
source : T ;
target : T;
}
```

Contract package

```
package hermes.iml.contracts;
meta type Assume ;
meta type Guarantee;
```

Language package

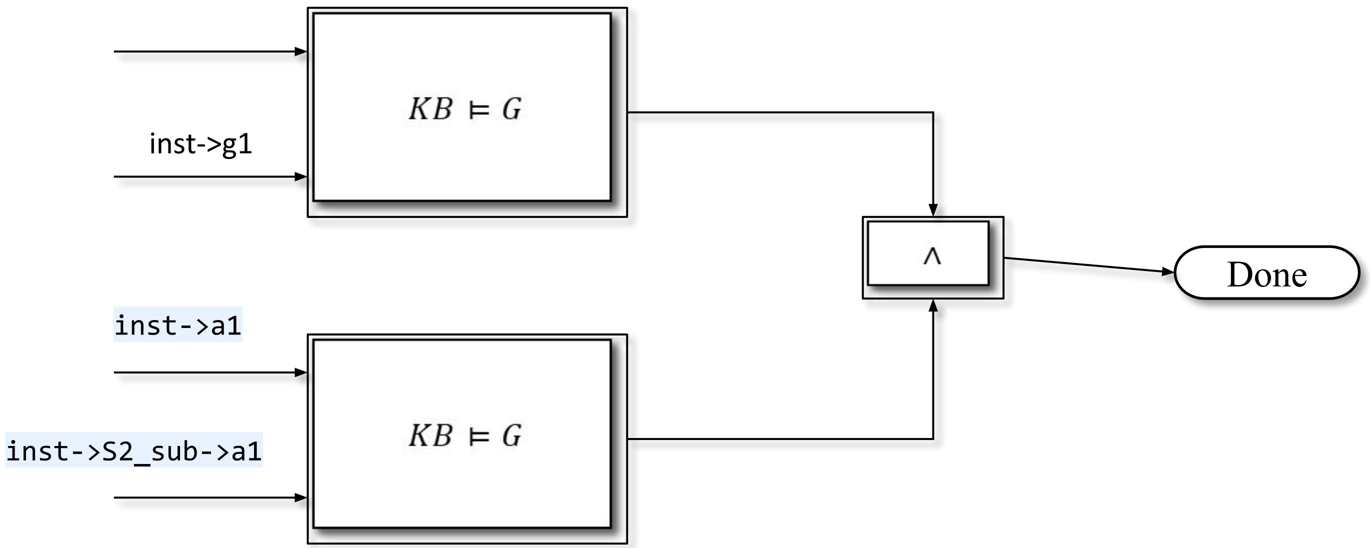
```
package iml.lang;
type Int ;
type Real ;
type Bool ;
meta type Assert;
meta type Goal;
meta type Modality;

sqrt : Real ~> Real ;
sin : Real ~> Real ;
cos : Real ~> Real ;
```

IML model

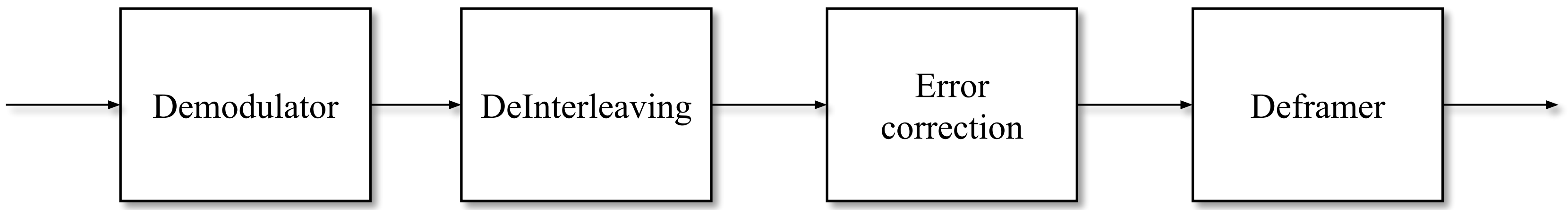
```
package utrc.test1 ;
import iml.lang.*;
import hermes.iml.aadl.* ;
import hermes.iml.contracts.* ;


type <<s:system>> S1 {
i1 <<i:in,p:port>>: Float;
i2 <<i:in,p:port>>: Float ;
o1 <<o:out,p:port>>: Float ;
n <<i:in,p:port>>: Integer;
    a1 <<a:Assume>> : Bool := exists x:Int, y:Int { (y >= 0 && y <= n && x>=1
&& x <= 0) => ( (i1 = x/n || i1 = -1 * x/n) && ( i2 = y/n || i2 =  -1 *y/n))
} ;
g1 <<g:Guarantee>>: Bool := o1 <=1 && o1 >=-1;
}


type <<s:system>> S2 {
i1 <<i:in,p:port>>: Float;
i2 <<i:in,p:port>>: Float ;
o1 <<o:out,p:port>>: Float ;
alpha <<i:in,p:port>>: Float;
    a1 <<a:Assume>> : Bool := (i1 =1 || i1=-1) && (i2=0 || i2 =1 || i2 = -1)
;
g1 <<g:Guarantee>>: Bool := o1 = i1 * sqrt(2) * sin(alpha) + i2 * sqrt(2) *
cos(alpha);
}

type <<s:system,i:implementation>> S1__impl extends S1 {
S2_sub <<c:subcomponent>>: S2 ;
i1_TO_A : Connection := new Connection {source=i1; target = S2_sub->i1;};
i2_TO_A : Connection := new Connection {source=i2 ; target = S2_sub->i2;};
S2_TO_o1 : Connection := new Connection {source=S2_sub->o1 ; target =o1 ;} ;

//i1_TO_A <<c:connection>>: Bool := i1 = S2_sub->i1;
//i2_TO_A <<c:connection>>: Bool := i2 = S2_sub->i2;
//S2_TO_o1 <<c:connection>>: Bool := S2_sub->o1 = o1 ;
}
```

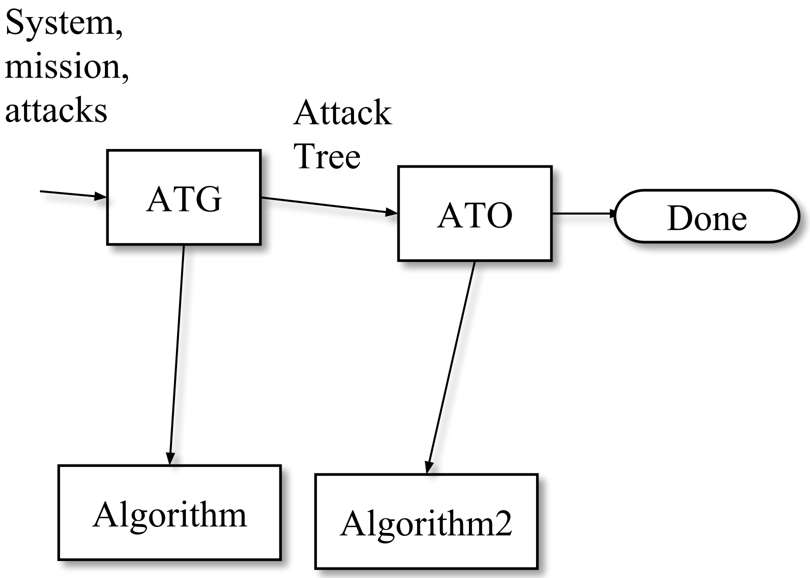
Reasoning graph (written in IML) where inst is a generic instance of S1__iml

`(inst->S2_sub->a1 => inst->S2_sub->g1) && inst->a1`



BACKUP


Demodulator
DeInterleaving
Error correction
Deframer


System, mission, attacks
ATG
Attack Tree
ATO
Done
Algorithm
Algorithm2