α-lang — A Simply Typed Lambda Calculus with Reliability Assertion and Conservative Requests Optimization under High Reliability Assurance

Yifan Chen, Haoze Wu, Ruyi Ji

June 11, 2017

Abstract

We design and elaborate α-lang, a simply typed lambda calculus armed with a conservative static analysis and an independent optimization algorithm, capable of solving a meaningful problem. Definitions, implementation, algorithm design & proofs and tests are all implemented.

1 Introduction

The statistic analysis in big data usually requires distributed computing, where master process gathers the parameters computed by slave processes/nodes based on massive data and appropriate algorithms (e.g. stream algorithms). Suppose we program based on these parameters and pursue high reliability assurance and accuracy with high probability, repeated sampling/computation requests need to be invoked and thus might result in overwhelming computation cost.

However, many randomized algorithms/stream algorithms/filters produce good-enough solutions with high probability, so repeated sampling/computation requests produce statistics such as mode and median, which augments the accuracy exponentially. We believe that we abstract a considerable set of these programs whose semantic can be expressed by a simply typed lambda calculus we design and elaborate there-in-after, armed with a conservative requests optimization, which enables programmers to declare correctness assertions and to enjoy a reasonable and distinguished requests arrangement that satisfies those assertions with high probability.

2 Language Definition

2.1 Syntax Rules

`t ::=`		(terms)
	$\langle$`int`$\rangle$	(int value)
	`x`	(variable)
	λ`x:T.t`	(abstraction)
	`t t`	(application)
	`t` $\langle$`op`$\rangle$ `t`	(algebraic operation)
	`if t` $\langle$`cmp`$\rangle$ `t then t else t`	(comparasion conditional)
	`assert` $\langle$`prob`$\rangle$ `t`	(reliability assertion)

`v ::=`		(values)
	$\langle$`int`$\rangle$	(int value)
	λ`x:T.t`	(abstraction value)

`T ::=`		(types)
	`Int`	(int type)
	`T` $\rightarrow$ `T`	(function type)

Γ `::=`		(contexts)
	$\emptyset$	(empty context)
	Γ, `x:T`	(term variable binding)

where

$$\langle\texttt{int}\rangle \in \mathcal{Z}$$
$$\langle\texttt{prob}\rangle \in (\frac{1}{2}, 1)$$
$$\langle\texttt{op}\rangle \in \{+, -, \times\}$$
$$\langle\texttt{cmp}\rangle \in \{=, \neq, <, >, \leq, \geq\}$$

2.2 Evaluation Rules

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

$$(\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-App-Abs)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \langle \mathtt{op} \rangle\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \langle \mathtt{op} \rangle\ \mathtt{t}_2} \quad \text{(E-Op1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \langle \mathtt{op} \rangle\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \langle \mathtt{op} \rangle\ \mathtt{t}_2'} \quad \text{(E-Op2)}$$

$$\langle \mathtt{int} \rangle\ \langle \mathtt{op} \rangle\ \langle \mathtt{int} \rangle \longrightarrow \langle \mathtt{int} \rangle \quad \text{(E-Prim-Op)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{if}\ \mathtt{t}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{t}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4 \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \langle \mathtt{cmp} \rangle\ \mathtt{t}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4} \quad \text{(E-Cnd1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{if}\ \mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{t}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4 \longrightarrow \mathtt{if}\ \mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{t}_2'\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4} \quad \text{(E-Cnd2)}$$

$$\frac{\mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{v}_2 \text{ is true}}{\mathtt{if}\ \mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{v}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4 \longrightarrow \mathtt{t}_3} \quad \text{(E-Cnd-True)}$$

$$\frac{\mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{v}_2 \text{ is false}}{\mathtt{if}\ \mathtt{v}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{v}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4 \longrightarrow \mathtt{t}_4} \quad \text{(E-Cnd-False)}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t} \longrightarrow \mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t}'} \quad \text{(E-Ass-Term)}$$

$$\mathtt{assert}\ \langle \mathtt{prob} \rangle\ \langle \mathtt{int} \rangle \longrightarrow \langle \mathtt{int} \rangle \quad \text{(E-Ass-Int)}$$

2.3 Typing Rules

$$\vdash \langle \mathtt{int} \rangle : \mathtt{Int} \quad \text{(T-Int)}$$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x}{:}\mathtt{T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2{:}\mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2{:}\mathtt{T}_1 \to \mathtt{T}_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1{:}\mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2{:}\mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2{:}\mathtt{T}_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1{:}\mathtt{Int} \qquad \Gamma \vdash \mathtt{t}_2{:}\mathtt{Int}}{\Gamma \vdash \mathtt{t}_1\ \langle \mathtt{op} \rangle\ \mathtt{t}_2{:}\mathtt{Int}} \quad \text{(T-Op)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1{:}\mathtt{Int} \qquad \Gamma \vdash \mathtt{t}_2{:}\mathtt{Int} \qquad \Gamma \vdash \mathtt{t}_3{:}\mathtt{T} \qquad \Gamma \vdash \mathtt{t}_4{:}\mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \langle \mathtt{cmp} \rangle\ \mathtt{t}_2\ \mathtt{then}\ \mathtt{t}_3\ \mathtt{else}\ \mathtt{t}_4{:}\mathtt{T}} \quad \text{(T-Cnd)}$$

$$\frac{\Gamma \vdash \mathtt{t}{:}\mathtt{T}}{\Gamma \vdash \mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t}{:}\mathtt{T}} \quad \text{(T-Ass)}$$

2.4 Notes

The α-lang we designed is similar to the simply typed lambda calculus we have learned. We just specify the `if` sentences so that we can avoid dealing with `Bool` type, and `assert` sentence is defined, which is equivalent to `id` function on `Int` type in semantics.

In addition, the type of the whole program must be `Int` and we are not allowed to `assert` term which is arrow type, because it might result in some troubles in our analysis and algorithm design.

It is worth noting that the real number of assertions might be more than the literal number of assertions in the program, because we might define functors encoded with `assert` sentences and these functors might propagate many assertions.

3 Analysis of the Problem

3.1 Problem Formalization

Intuitively, we would like to minimize the number of all data requests and guarantee that the program probably produces the correct answer that we expect. Formally, given n variable $x_i, i = 1, \cdots, n$, with accuracy α_i, a program of α-lang is applied with them, where m constraints (S_i, l_i) is specified by assertions. Each variable will be sampled for $t_i > 0$ times, which is to be calculated such that the sum of them minimizes. S_i is specifically determined by a conservative data-flow analysis there-in-after which we name *Key Variable Analysis*, representing the set constituted by elements which occurs in the computation in terms of this fixed assertion, and l_i is given in the assertion statement by programmer. The goal is to satisfy the probability constraint of every assertion with the minimum number of data requests.

In conclusion, the input is a program of α-lang with free `Int` variable $x_1, \cdots, x_n$, plus $\alpha_1, \cdots, \alpha_n$ which separately denote the accuracy of each variable. The output is $t_1, \cdots, t_n$ which separately denotes the number of data requests of each variable so that all assertion constraints are satisfied.

3.2 Problem Transformation

For each assertion, the result is related to some variables and the others do not matter. Hence, for each assertion, intuitively we can perform a may-analysis to compute all variables that might influence the result, which we name *Key Variable Analysis* and elaborate there-in-after.

Suppose we finish the key analysis and attain S_i, then α_i, the probability of satisfying this assertion, is $\prod_{k \in S_i} \alpha_k$, which should $\geq l_i$.

Then we analysis the upper bound as follow.

Intuitively, repeated sampling of variable x_k produces a mode, whose accuracy improves exponentially, as long as it is strictly greater than $\frac{1}{2}$. More precisely, suppose for a fixed variable x_k, we sample for t_t times, then the accuracy α_k is improved to

$$\begin{aligned}\sum_{i=\lceil \frac{t_k}{2} \rceil}^{t_k} \binom{t_k}{i} \alpha_k^i (1-\alpha_k)^{t_k-i} &= 1 - \sum_{i=0}^{\lceil \frac{t_k}{2} \rceil - 1} \binom{t_k}{i} \alpha_k^i (1-\alpha_k)^{t_k-i} \\ &\geq 1 - 2^{t_k-1} \alpha_k^{\frac{t_k}{2}} (1-\alpha_k)^{\frac{t_k}{2}} \\ &= 1 - \frac{W_k^{\frac{t_k}{2}}}{2}\end{aligned}$$

where $W_k = 4\alpha_k(1-\alpha_k) < 1$ is a constant in terms of the variable x_k.

The condition $1 - \frac{W_k^{\frac{t_k}{2}}}{2} \geq R_k$ guarantee that the improved accuracy of variable x_k is at least R_k, and we rewrite as $t_k \geq 2\log_{W_k}(2-2R_k)$, which means that for any precision $\epsilon > 0$, the constraint $t_k \geq 2\log_{W_k}(2\epsilon)$ serves as a upper bound of a valid solution. (With fixed $\alpha_1, \cdots, \alpha_n$, this upper bound is polynomial, NOT pseudo-polynomial.)

The upper bound convinces us of the existence of not bad solutions. However, this problem seems to be a difficult programming problem, so we just designed a naive greedy algorithm as the programming solver.

3.3 Programming Solver

For the i-th assertion, the constraint $\prod_{k \in S_i} R_k \geq l_i$ is equivalent to

$$\sum_{k \in S_i} \ln R_k \geq \ln l_i.$$

At the very beginning each inequality is yet satisfied and we try to increase their left head side. Notice that the condition

$$\ln(1 - \frac{W_k^{t_k+1}}{2}) \geq \ln R_i$$

guarantees the R_i constraint for variable x_i, through increasing t_k to $t_k + 1$, we increase the left head side of the equality of the i-th assertion in which the variable x_k exists, by

$$\ln(1 - \frac{W_k^{t_k+1}}{2}) - \ln(1 - \frac{W_k^{t_k}}{2}).$$

In our greedy solver, each time we simply increase t_k by 1, and the k is determined such that

$$N_k(\ln(1 - \frac{W_k^{t_k+1}}{2}) - \ln(1 - \frac{W_k^{t_k}}{2}))$$

maximizes, where N_k denotes the number of x_k's occurrence over all S_i.

4 Key Variable Analysis

4.1 The Simplified Version

It might be a bit difficult to directly tackle the problem, so we consider the simplified version first. The simplification simply erases the `if ... then ... else ...` sentence from the syntactic rule, so that branches temporarily won't bother us.

The remaining syntax rules are:

`t ::=`	(terms)
⟨`int`⟩	(int value)
`x`	(variable)
λ`x:T.t`	(abstraction)
`t t`	(application)
`t` ⟨`op`⟩ `t`	(algebraic operation)
`assert` ⟨`prob`⟩ `t`	(reliability assertion)

Therefore, although the specified values of each free variable is unknown in the program, we can "pretend to evaluate" the program without those value and perform the *Key Variable Analysis* by the way. The spirit of this analysis is that if we give the algebraic formula of the term we assert all variables constituting this formula serve as a straight-forward may-analysis. We name this special evaluation with *abstract α-interpretation*, which deals with `Set` elements instead of `Int`. The `Set` type denotes a set of the free variable $x_1, \cdots, x_n$ of the whole program of α-lang.

According to the syntax rules, evaluation rules and typing rules of α-lang, we step by step present and explain the syntax rules, abstract interpretation rules and typing rules of α-interpreter.

`t ::=`	(terms)
⟨`set`⟩	(set value)
`x`	(variable)
λ`x:T.t`	(abstraction)
`t t`	(application)
`t` ∪ `t`	(set union)
`assert` ⟨`prob`⟩ `t`	(reliability assertion)

`v ::=`	(values)
⟨`set`⟩	(set value)
λ`x:T.t`	(abstraction value)

$$
\begin{array}{lll}
\mathtt{T} ::= & & \text{(types)} \\
& \mathtt{Set} & \text{(set type)} \\
& \mathtt{T} \to \mathtt{T} & \text{(function type)} \\
\\
\Gamma ::= & & \text{(contexts)} \\
& \emptyset & \text{(empty context)} \\
& \Gamma, \mathtt{x{:}T} & \text{(term variable binding)}
\end{array}
$$

The conversion is straight-forward. We simply replace $\langle \mathtt{op} \rangle$ by $\cup$, $\langle \mathtt{int} \rangle$ by $\emptyset$, and x_i by $\{x_i\}$.

The typing rules are also straight-forward.

$$\vdash \langle \mathtt{set} \rangle \mathtt{:Set} \qquad \text{(T-Set)}$$

$$\frac{\mathtt{x{:}T} \in \Gamma}{\Gamma \vdash \mathtt{x{:}T}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x{:}T_1} \vdash \mathtt{t_2{:}T_2}}{\Gamma \vdash \lambda \mathtt{x{:}T_1}\,.\,\mathtt{t_2{:}T_1} \to \mathtt{T_2}} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t_1{:}T_{11}} \to \mathtt{T_{12}} \qquad \Gamma \vdash \mathtt{t_2{:}T_{11}}}{\Gamma \vdash \mathtt{t_1\ t_2{:}T_{12}}} \qquad \text{(T-App)}$$

$$\frac{\Gamma \vdash \mathtt{t_1{:}Set} \qquad \Gamma \vdash \mathtt{t_2{:}Set}}{\Gamma \vdash \mathtt{t_1} \cup \mathtt{t_2{:}Set}} \qquad \text{(T-Union)}$$

$$\frac{\Gamma \vdash \mathtt{t{:}T}}{\Gamma \vdash \mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t{:}T}} \qquad \text{(T-Ass)}$$

Abstract interpretation rules:

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ t_2} \longrightarrow \mathtt{t_1'\ t_2}} \qquad \text{(I-App1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{v_1\ t_2} \longrightarrow \mathtt{v_1\ t_2'}} \qquad \text{(I-App2)}$$

$$(\lambda \mathtt{x{:}T_{11}}\,.\,\mathtt{t_{12}})\mathtt{v_2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v_2}]\mathtt{t_{12}} \qquad \text{(I-App-Abs)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1} \cup \mathtt{t_2} \longrightarrow \mathtt{t_1'} \cup \mathtt{t_2}} \qquad \text{(I-Union1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{v_1} \cup \mathtt{t_2} \longrightarrow \mathtt{v_1} \cup \mathtt{t_2'}} \qquad \text{(I-Union2)}$$

$$\langle \mathtt{set} \rangle \cup \langle \mathtt{set} \rangle \longrightarrow \langle \mathtt{set} \rangle \qquad \text{(I-Prim-Union)}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t'}}{\mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t} \longrightarrow \mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t'}} \qquad \text{(I-Ass-Term)}$$

$$\mathtt{assert}\ \langle \mathtt{prob} \rangle\ \langle \mathtt{set} \rangle \longrightarrow \langle \mathtt{set} \rangle \qquad \text{(I-Ass-Set)}$$

Now consider the last abstract interpretation rule. We can by the way add a piece of constraint to the global assertion constraints set, which will be eventually solved by the programming solver we designed earlier.

However, from the foregoing we mention that the real number of assertions might be more than the literal number of assertions in the program, because we might define functors encoded with `assert` sentences and these functors might propagate many assertions. In this case, we are simply confronted with some term like $\mathtt{t}_1\ \mathtt{t}_2$ and $\mathtt{t}_2$ contains a lambda abstraction inside which `assert` sentences exist. By

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(I-App2)}$$

and

$$(\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \qquad \text{(I-App-Abs)}$$

the propagation of `assert` functors realize. We just follow the abstract interpretation rules and now they suffice.

4.2 The Complete Version

Now we try to introduce `if` sentence and complement an additional syntax rule — `if t` $\cup$ `t then t else t`.

$\langle$`cmp`$\rangle$ is replaced by $\cup$, because we don't care about the whether the predicate is true or false. We simply would like to know, (1) what branch this conditional term falls into (which depends on the predicate), and (2) what variables may have something to do with the evaluation result of the predicate and the two branches.

Hence the additional syntax rule is:

```
t ::=                                              (terms)
      if t then t else t                           (conditional)
```

And the additional typing rule is straight-forward:

$$\frac{\Gamma \vdash \mathtt{t}_1{:}\mathtt{Set} \qquad \Gamma \vdash \mathtt{t}_2{:}\mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3{:}\mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3{:}\mathtt{T}} \qquad \text{(T-Cnd)}$$

However, the additional abstract interpretation rules are non-trivial, because we are dealing with branches. We cannot simply choose one branch and drop the other.

Therefore we treat the term `if v then v else v` as value:

```
v ::=                                              (values)
      if v then v else v                           (conditional value)
```

and continue our analysis to see if we are able to figure out some approach to deal with such term.

Notice that what we actually would like to analyze from the problem is simply the variable set for all assertions, we just need to focus on those essential `if` sentences.

For brief, we temporarily omit the $\langle$`prob`$\rangle$ component of `assert` terms.

For example, in

$$\texttt{if } t_1 \texttt{ then assert } t_2 \texttt{ else } t_3$$

the assertion has nothing to do with what branch the interpretation falls in and thus $\texttt{t}_1$ is meaningless, but in

$$\texttt{assert if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$$

the term $\texttt{t}_1$ matters because it determines which branch the interpretation chooses and thereby influences the value of the whole term.

Our key idea is to simplify or transform our program such that `if` sentences float to the outer the the others float inward. Obviously we have simplification rules:

$$\texttt{(if } v_1 \texttt{ then } v_2 \texttt{ else } v_3\texttt{) } v_4 \longrightarrow \texttt{if } v_1 \texttt{ then (} v_2 \texttt{ } v_4\texttt{) else (} v_3 \texttt{ } v_4\texttt{)} \quad \text{(I-Cnd-App-Simp1)}$$

$$v_1 \texttt{ (if } v_2 \texttt{ then } v_3 \texttt{ else } v_4\texttt{)} \longrightarrow \texttt{if } v_2 \texttt{ then (} v_1 \texttt{ } v_3\texttt{) else (} v_1 \texttt{ } v_4\texttt{)} \quad \text{(I-Cnd-App-Simp2)}$$

$$\texttt{(if } v_1 \texttt{ then } v_2 \texttt{ else } v_3\texttt{)} \cup v_4 \longrightarrow \texttt{if } v_1 \texttt{ then (}v_2 \cup v_4\texttt{) else (}v_3 \cup v_4\texttt{)} \quad \text{(I-Cnd-Union-Simp1)}$$

$$v_1 \cup \texttt{(if } v_2 \texttt{ then } v_3 \texttt{ else } v_4\texttt{)} \longrightarrow \texttt{if } v_2 \texttt{ then (}v_1 \cup v_3\texttt{) else (}v_1 \cup v_4\texttt{)} \quad \text{(I-Cnd-Union-Simp2)}$$

Since we have abstract interpretation rule

$$\frac{\texttt{t} \longrightarrow \texttt{t}'}{\texttt{assert } \langle\texttt{prob}\rangle \texttt{ t} \longrightarrow \texttt{assert } \langle\texttt{prob}\rangle \texttt{ t}'} \quad \text{(I-Ass-Term)}$$

and

$$\texttt{assert } \langle\texttt{prob}\rangle \langle\texttt{set}\rangle \longrightarrow \langle\texttt{set}\rangle \quad \text{(I-Ass-Set)}$$

we simply add three abstract interpretation rules

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \quad \text{(I-Cnd1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\texttt{if } v_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } v_1 \texttt{ then } t_2' \texttt{ else } t_3} \quad \text{(I-Cnd2)}$$

$$\frac{t_3 \longrightarrow t_3'}{\texttt{if } v_1 \texttt{ then } v_2 \texttt{ else } t_3 \longrightarrow \texttt{if } v_1 \texttt{ then } v_2 \texttt{ else } t_3'} \quad \text{(I-Cnd3)}$$

and then the interpreter is able to avoid dealing with `if` sentence until the interpretation is exhaustive and falls into kind of normal form, such as

$$\texttt{assert if } v_1 \texttt{ then } v_2 \texttt{ else } v_3.$$

Again, we consider the complete syntax

`t ::=`	(terms)
$\langle$`set`$\rangle$	(set value)
`x`	(variable)
λ`x:T.t`	(abstraction)
`t t`	(application)
`t` $\cup$ `t`	(set union)
`if t then t else t`	(conditional)
`assert` $\langle$`prob`$\rangle$ `t`	(reliability assertion)

and we conclude that currently all unmanageable cases are listed:

`assert if` v_1 `then` v_2 `else` v_3	(Ass-Cnd)
`assert (`v_1 `(if` v_2 `then` v_3 `else` v_4`))`	(Ass-Cnd-App1)
`assert ((if` v_1 `then` v_2 `else` v_3`)` v_4`)`	(Ass-Cnd-App2)
`assert (`$v_1 \cup$`(if` v_2 `then` v_3 `else` v_4`))`	(Ass-Cnd-Union1)
`assert ((if` v_1 `then` v_2 `else` v_3`)`$\cup v_4$`)`	(Ass-Cnd-Union2)

We attempt to recognize these patterns and consider the appropriate transformations. It isn't simplification because the semantics has subtle change.

Firstly we consider

`assert if` v_1 `then` v_2 `else` $v_3 \longrightarrow$

`if assert` v_1 `then assert` v_2 `else assert` v_3 (I-Ass-Cnd-Trans)

and the comprehension is that if we guarantee that the term

`if` v_1 `then` v_2 `else` v_3

is correct, we simply provide a kind of weaker condition which guarantees that terms v_2 and v_3 are correct anyway and term v_1 is also correct (so that the correct branch is chosen).

However, the three assertions that propagate from the original assertion should be treated as a whole, so the sets they eventually separately figure out should be a union and serves as ONE assertion constraint but not three.

Our solution is to specify each assertion with a unique index number and we modify one syntax rule

`t ::=`	(terms)
`assert` $\langle$`index`$\rangle$ $\langle$`prob`$\rangle$ `t`	(reliability assertion)

and correct the corresponding abstract interpretation.

Again, for brief, we omit the $\langle$`index`$\rangle$ component of `assert` terms, so now the modified `assert` terms look just the same as before.

Similarly, we have

$$\texttt{assert (v}_1\ \texttt{(if v}_2\ \texttt{then v}_3\ \texttt{else v}_4\texttt{))} \longrightarrow$$

$$\texttt{(if assert v}_2\ \texttt{then assert (v}_1\ \texttt{v}_3\texttt{) else assert (v}_1\ \texttt{v}_4\texttt{))} \quad \text{(I-Ass-Cnd-App-Trans1)}$$

$$\texttt{assert ((if v}_1\ \texttt{then v}_2\ \texttt{else v}_3\texttt{) v}_4\texttt{)} \longrightarrow$$

$$\texttt{(if assert v}_1\ \texttt{then assert (v}_2\ \texttt{v}_4\texttt{) else assert (v}_3\ \texttt{v}_4\texttt{))} \quad \text{(I-Ass-Cnd-App-Trans2)}$$

$$\texttt{assert (v}_1\cup\texttt{(if v}_2\ \texttt{then v}_3\ \texttt{else v}_4\texttt{))} \longrightarrow$$

$$\texttt{(if assert v}_2\ \texttt{then assert (v}_1\cup\texttt{v}_3\texttt{) else assert (v}_1\cup\texttt{v}_4\texttt{))} \quad \text{(I-Ass-Cnd-Union-Trans1)}$$

$$\texttt{assert ((if v}_1\ \texttt{then v}_2\ \texttt{else v}_3\texttt{)}\cup\texttt{v}_4\texttt{)} \longrightarrow$$

$$\texttt{(if assert v}_1\ \texttt{then assert (v}_2\cup\texttt{v}_4\texttt{) else assert (v}_3\cup\texttt{v}_4\texttt{))} \quad \text{(I-Ass-Cnd-Union-Trans2)}$$

It must be emphasized that the $\langle$`index`$\rangle$ of the `assert` term holds along such kind of Ass-Cnd transformation.

However, from the foregoing we mention that the real number of assertions might be more than the literal number of assertions in the program, because we might define functors encoded with `assert` sentences and these functors might propagate many assertions. In this case, we are simply confronted with some term like $\texttt{t}_1\ \ \texttt{t}_2$ and $\texttt{t}_2$ contains a lambda abstraction inside which `assert` sentences exist, so when we perform the application we need to "deep copy" but not simply "substitute" ("shallow copy") the term. The I-App-Abs should be rewrite as

$$(\lambda\texttt{x:T}_{11}\texttt{.t}_{12})\texttt{v}_2 \longrightarrow [\texttt{x} \mapsto \texttt{copy}\ \ \texttt{v}_2]\texttt{t}_{12} \qquad \text{(I-App-Abs)}$$

where `copy` function simply replace all indices of the `assert` sentences such that all assertions propagated have unique index.

A natural question is whether or not the `copy` operations conflicts with Ass-Cnd transformations. Actually `copy` operations (assertion propagations) and Ass-Cnd transformations will not interfere with each other because (1) assertion propagations before Ass-Cnd transformations do not influence the consistency of the `index` after Ass-Cnd transformations and (2) we do not allow the assertion of terms except `Int` type and the interpretation rule

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1\ \texttt{t}_2 \longrightarrow \texttt{t}_1'\ \texttt{t}_2} \qquad \text{(I-App1)}$$

and

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(I-App2)}$$

and

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t} \longrightarrow \mathtt{assert}\ \langle \mathtt{prob} \rangle\ \mathtt{t}'} \qquad \text{(I-Ass-Term)}$$

and

$$\mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ \langle \mathtt{set} \rangle \longrightarrow \langle \mathtt{set} \rangle \qquad \text{(I-Ass-Set)}$$

guarantees that the problematic cases can only be among terms of

$$\mathtt{v}_1\ \ (\mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ \langle \mathtt{set} \rangle) \qquad \text{(App-Ass-Set)}$$

and

$$\mathtt{v}_1\ \ (\mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ (\mathtt{if}\ \mathtt{v}_2\ \mathtt{then}\ \mathtt{v}_3\ \mathtt{else}\ \mathtt{v}_4)) \qquad \text{(App-Ass-Cnd)}$$

and other App-Ass-Cnd cases. In these cases, the assertion propagation won't happen because the next interpretation will be

$$\mathtt{v}_1\ \langle \mathtt{set} \rangle \qquad \text{(App-Set)}$$

and

$$\mathtt{v}_1\ (\mathtt{if}\ \mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ \mathtt{v}_2\ \mathtt{then}\ \mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ \mathtt{v}_3\ \mathtt{else}\ \mathtt{assert}\ \langle \mathtt{index} \rangle\ \langle \mathtt{prob} \rangle\ \mathtt{v}_4) \qquad \text{(App-Cnd-Ass)}$$

and other App-Cnd-Ass cases, followed by

$$(\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \langle \mathtt{set} \rangle \longrightarrow [\mathtt{x} \mapsto \mathtt{copy}\ \langle \mathtt{set} \rangle]\mathtt{t}_{12} \qquad \text{(I-App-Abs-Set)}$$

and

$$(\mathtt{if}\ \mathtt{v}_1\ \mathtt{then}\ \mathtt{v}_2\ \mathtt{else}\ \mathtt{v}_3)\ \mathtt{v}_4 \longrightarrow \mathtt{if}\ \mathtt{v}_1\ \mathtt{then}\ (\mathtt{v}_2\ \mathtt{v}_4)\ \mathtt{else}\ (\mathtt{v}_3\ \mathtt{v}_4) \qquad \text{(I-Cnd-App-Simp1)}$$

and other Cnd-Simp simplifications.

Therefore by induction we can prove that Ass-Cnd transformations imply the absence of assertion propagations. Also we can tell it by intuition that assertion propagations exist only in the case of lambda abstractions inside which `assert` sentences exist, which cannot be the case of Ass-Cnd transformations because Ass-Cnd transformations deal with `Set` type instead of lambda abstractions.

The complete abstract α-interpreter is finished. The interpretation result is a redundant stuff with `if` sentences and $\langle \mathtt{set} \rangle$ values. Since all `assert` sentences disappear, we can simply drop this stuff.

5 Complete Abstract α-interpreter

5.1 Abstract α-interpreter Syntax Rules

`t ::=`		(terms)
	$\langle$`set`$\rangle$	(set value)
	`x`	(variable)
	λ`x:T.t`	(abstraction)
	`t t`	(application)
	`t` $\cup$ `t`	(union operation)
	`if t then t else t`	(conditional)
	`assert` $\langle$`index`$\rangle$ $\langle$`prob`$\rangle$ `t`	(reliability assertion)
	`copy t`	(deep copy)

`v ::=`		(values)
	$\langle$`set`$\rangle$	(set value)
	λ`x:T.t`	(abstraction value)
	`if v then v else v`	(conditional value)

`T ::=`		(types)
	`Set`	(set type)
	`T` $\rightarrow$ `T`	(function type)

Γ `::=`		(contexts)
	$\emptyset$	(empty context)
	Γ, `x:T`	(term variable binding)

where

$$\langle \texttt{set} \rangle \subset \{x_1, \cdots, x_n\}$$
$$\langle \texttt{prob} \rangle \in (\frac{1}{2}, 1)$$
$$\langle \texttt{index} \rangle \in \mathcal{N}$$

5.2 Abstract α-interpreter Interpretation Rules

$$\texttt{copy}\ \langle \texttt{set} \rangle \longrightarrow \langle \texttt{set} \rangle \qquad \text{(I-Copy-Set)}$$

$$\texttt{copy}\ \texttt{x} \longrightarrow \texttt{x} \qquad \text{(I-Copy-Var)}$$

$$\texttt{copy}\ \lambda \texttt{x:T.t} \longrightarrow \lambda \texttt{x:T.(copy t)} \qquad \text{(I-Copy-Abs)}$$

$$\texttt{copy}\ (\texttt{t}_1\ \texttt{t}_2) \longrightarrow (\texttt{copy}\ \texttt{t}_1)\ (\texttt{copy}\ \texttt{t}_2) \qquad \text{(I-Copy-App)}$$

$$\texttt{copy}\ (\texttt{t}_1 \cup \texttt{t}_2) \longrightarrow (\texttt{copy}\ \texttt{t}_1) \cup (\texttt{copy}\ \texttt{t}_2) \qquad \text{(I-Copy-Union)}$$

$$\texttt{copy}\ (\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3) \longrightarrow \texttt{if}\ (\texttt{copy}\ \texttt{t}_1)\ \texttt{then}\ (\texttt{copy}\ \texttt{t}_2)\ \texttt{else}\ (\texttt{copy}\ \texttt{t}_3) \qquad \text{(I-Copy-Cnd)}$$

$$\texttt{copy}\ (\texttt{copy}\ \texttt{t}) \longrightarrow \texttt{copy}\ \texttt{t} \qquad \text{(I-Copy-Copy)}$$

$$\texttt{copy}\ (\texttt{assert}\ \langle \texttt{index} \rangle\ \langle \texttt{prob} \rangle\ \texttt{t}) \longrightarrow \texttt{assert}\ \langle \texttt{index} \rangle\ \langle \texttt{prob} \rangle\ (\texttt{copy}\ \texttt{t}) \qquad \text{(I-Copy-Ass)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1\ \texttt{t}_2 \longrightarrow \texttt{t}_1'\ \texttt{t}_2} \qquad \text{(I-App1)}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{v}_1\ \texttt{t}_2 \longrightarrow \texttt{v}_1\ \texttt{t}_2'} \qquad \text{(I-App2)}$$

$$(\lambda \texttt{x:T}_{11}.\texttt{t}_{12})\texttt{v}_2 \longrightarrow [\texttt{x} \mapsto \texttt{copy}\ \texttt{v}_2]\texttt{t}_{12} \qquad \text{(I-App-Abs)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{t}_1 \cup \texttt{t}_2 \longrightarrow \texttt{t}_1' \cup \texttt{t}_2} \qquad \text{(I-Union1)}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{v}_1 \cup \texttt{t}_2 \longrightarrow \texttt{v}_1 \cup \texttt{t}_2'} \qquad \text{(I-Union2)}$$

$$\langle \texttt{set} \rangle \cup \langle \texttt{set} \rangle \longrightarrow \langle \texttt{set} \rangle \qquad \text{(I-Prim-Union)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \longrightarrow \texttt{if}\ \texttt{t}_1'\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3} \qquad \text{(I-Cnd1)}$$

$$\frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \longrightarrow \texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{t}_2'\ \texttt{else}\ \texttt{t}_3} \qquad \text{(I-Cnd2)}$$

$$\frac{\texttt{t}_3 \longrightarrow \texttt{t}_3'}{\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{t}_3 \longrightarrow \texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{t}_3'} \qquad \text{(I-Cnd3)}$$

$$(\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{v}_3)\ \texttt{v}_4 \longrightarrow \texttt{if}\ \texttt{v}_1\ \texttt{then}\ (\texttt{v}_2\ \texttt{v}_4)\ \texttt{else}\ (\texttt{v}_3\ \texttt{v}_4) \qquad \text{(I-Cnd-App-Simp1)}$$

$$\texttt{v}_1\ (\texttt{if}\ \texttt{v}_2\ \texttt{then}\ \texttt{v}_3\ \texttt{else}\ \texttt{v}_4) \longrightarrow \texttt{if}\ \texttt{v}_2\ \texttt{then}\ (\texttt{v}_1\ \texttt{v}_3)\ \texttt{else}\ (\texttt{v}_1\ \texttt{v}_4) \qquad \text{(I-Cnd-App-Simp2)}$$

$$(\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{v}_3) \cup \texttt{v}_4 \longrightarrow \texttt{if}\ \texttt{v}_1\ \texttt{then}\ (\texttt{v}_2 \cup \texttt{v}_4)\ \texttt{else}\ (\texttt{v}_3 \cup \texttt{v}_4) \qquad \text{(I-Cnd-Union-Simp1)}$$

$$\texttt{v}_1 \cup (\texttt{if}\ \texttt{v}_2\ \texttt{then}\ \texttt{v}_3\ \texttt{else}\ \texttt{v}_4) \longrightarrow \texttt{if}\ \texttt{v}_2\ \texttt{then}\ (\texttt{v}_1 \cup \texttt{v}_3)\ \texttt{else}\ (\texttt{v}_1 \cup \texttt{v}_4) \qquad \text{(I-Cnd-Union-Simp2)}$$

$$\frac{\texttt{t} \longrightarrow \texttt{t}'}{\texttt{assert}\ \langle \texttt{index} \rangle\ \langle \texttt{prob} \rangle\ \texttt{t} \longrightarrow \texttt{assert}\ \langle \texttt{index} \rangle\ \langle \texttt{prob} \rangle\ \texttt{t}'} \qquad \text{(I-Ass-Term)}$$

$$\texttt{assert}\ \langle \texttt{index} \rangle\ \langle \texttt{prob} \rangle\ \langle \texttt{set} \rangle \longrightarrow \langle \texttt{set} \rangle \qquad \text{(I-Ass-Set)}$$

$$\texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{v}_3 \longrightarrow \texttt{if}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_1\ \texttt{then}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_2\ \texttt{else}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_3 \quad \text{(I-Ass-Cnd-Trans)}$$

$$\texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1\ (\texttt{if}\ \texttt{v}_2\ \texttt{then}\ \texttt{v}_3\ \texttt{else}\ \texttt{v}_4)) \longrightarrow (\texttt{if}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_2\ \texttt{then}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1\ \texttt{v}_3)\ \texttt{else}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1\ \texttt{v}_4)) \quad \text{(I-Ass-Cnd-App-Trans1)}$$

$$\texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ ((\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{v}_3)\ \texttt{v}_4) \longrightarrow (\texttt{if}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_1\ \texttt{then}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_2\ \texttt{v}_4)\ \texttt{else}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_3\ \texttt{v}_4)) \quad \text{(I-Ass-Cnd-App-Trans2)}$$

$$\texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1 \cup (\texttt{if}\ \texttt{v}_2\ \texttt{then}\ \texttt{v}_3\ \texttt{else}\ \texttt{v}_4)) \longrightarrow (\texttt{if}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_2\ \texttt{then}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1 \cup \texttt{v}_3)\ \texttt{else}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_1 \cup \texttt{v}_4)) \quad \text{(I-Ass-Cnd-Union-Trans1)}$$

$$\texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ ((\texttt{if}\ \texttt{v}_1\ \texttt{then}\ \texttt{v}_2\ \texttt{else}\ \texttt{v}_3) \cup \texttt{v}_4) \longrightarrow (\texttt{if}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{v}_1\ \texttt{then}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_2 \cup \texttt{v}_4)\ \texttt{else}\ \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ (\texttt{v}_3 \cup \texttt{v}_4)) \quad \text{(I-Ass-Cnd-Union-Trans2)}$$

5.3 Abstract α-interpreter Typing Rules

$$\vdash \langle\texttt{set}\rangle\texttt{:Set} \quad \text{(T-Set)}$$

$$\frac{\texttt{x:T} \in \Gamma}{\Gamma \vdash \texttt{x:T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2\texttt{:T}_2}{\Gamma \vdash \lambda\texttt{x:T}_1\texttt{.t}_2\texttt{:T}_1 \to \texttt{T}_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \texttt{t}_1\texttt{:T}_{11} \to \texttt{T}_{12} \qquad \Gamma \vdash \texttt{t}_2\texttt{:T}_{11}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2\texttt{:T}_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash \texttt{t}_1\texttt{:Set} \qquad \Gamma \vdash \texttt{t}_2\texttt{:Set}}{\Gamma \vdash \texttt{t}_1 \cup \texttt{t}_2\texttt{:Set}} \quad \text{(T-Union)}$$

$$\frac{\Gamma \vdash \texttt{t}_1\texttt{:Set} \qquad \Gamma \vdash \texttt{t}_2\texttt{:T} \qquad \Gamma \vdash \texttt{t}_3\texttt{:T}}{\Gamma \vdash \texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3\texttt{:T}} \quad \text{(T-Cnd)}$$

$$\frac{\Gamma \vdash \texttt{t:T}}{\Gamma \vdash \texttt{assert}\ \langle\texttt{index}\rangle\ \langle\texttt{prob}\rangle\ \texttt{t:T}} \quad \text{(T-Ass)}$$

$$\frac{\Gamma \vdash \texttt{t:T}}{\Gamma \vdash \texttt{copy}\ \texttt{t:T}} \quad \text{(T-Copy)}$$

6 Implementation

Armed with Haskell we successfully implement the α-lang compiler.

To put it simple, we let $\langle \texttt{op} \rangle \in \{+\}$ and $\langle \texttt{cmp} \rangle \in \{<\}$.

There are also some other (not very large) changes in the implementation. We use a transformation to convert input language to the execution language, where require are given executing iteration numbers and assertion are directly removed.

Another change is that we use direct require command in the format "[<require-name><given-reliablity>]" rather than free variable lists, for more convenient usage and possible further extensions.

7 Test

There are some test examples in the "examples" directory. Just use "require-test <file-name>" to see the result.

8 Conclusion

In conclusion, we have defined a language that can automatically handle the unstability of data source. This analysis can provide a example of method for constructing a highly reliable software in an environment full of unstable data source, which is quite similar to our real life.

9 Division of Labour

The Haskell Implementation is implemented by Yifan Chen. Haoze Wu and Ruyi Ji finish the language design, the algorithm design & analysis, the report, the slides and the test of the compiler implementation.

The *Key Variable Analysis* consumes a lot of time of design, analysis and discussion, and should be credited to all of us.