8217 COPY 2

The Mathematical Semantics of Algol 60

by

Peter Mosses

Technical Monograph PRG-12

January, 1974.

Oxford University Computing Laboratory,
Programming Research Group,
45 Banbury Road,
Oxford.

Oxford University Computing Laboratory,
Programming Research Group,
45 Banbury Road,
Oxford.

ABSTRACT

This paper describes the programming language ALGOL 60 (omitting own declarations) by using the Scott-Strachey mathematical semantics. A separate commentary on this description is provided, including an indication of the correspondence between the semantic description language and the λ-calculus.

Familiarity with previous publications on mathematical semantics, e.g. [6,8,10,13], and with the λ-calculus, is assumed.

CONTENTS

[The commentary is bound separately.]

REFERENCES

[1] Allen, C.D.; Chapman, D.N.; & Jones, C.B., *A Formal Definition of ALGOL 60*, IBM U.K. Technical Report TR12.105.

[2] Knuth, D., *The Remaining Trouble Spots in ALGOL 60*, Comm. ACM 10 (1967), pp. 611-618.

[3] Landin, P.J., *A Correspondence Between ALGOL 60 and Church's Lambda-Notation*, Comm. ACM 8 (1965), pp. 89-101, 158-165.

[4] Lauer, P., *Formal Definition of ALGOL 60*, IBM Lab. Vienna, Technical Report TR.25.088 (1968)

[5] Naur, P. (Ed.) *Revised Report on the Algorithmic Language ALGOL 60*, Comm. ACM 6 (1963), pp.1-17.

*[6] Scott, D., *Outline of a Mathematical Theory of Computation*, Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems (1970), pp. 169-176.

[7] Scott, D., *Lambda Calculus and Recursion Theory*, Private Communication.

*[8] Scott, D; Strachey, C. *Towards a Mathematical Semantics for Computer Languages*, Proc. Symposium on Computers and Automata. Microwave Institute Symposia Series 21, Polytechnic Institute of Brooklyn.

[9] Scott, D; Strachey, C. *Data Types as Lattices*, (in preparation).

*[10] Strachey, C. *Varieties of Programming Language*, Proc. International Computing Symposium, Cini Foundation, Venice (1972), pp.222-233.

[11] Strachey, C. *An Abstract Model for Storage*, (in preparation).

[12] Strachey, C.; Wadsworth, C.P. *Continuations: A Mathematical Semantics with Full Jumps*, Programming Research Group Technical Monograph PRG-11.

[13] Tennent, R.D., *Mathematical Semantics of Programming Languages* Technical Report 73-15 (May 1973) Dept. Computing & Information Science, Queen's University, Kingston, Ontario.

[14] Utman, R.E. (Chairman ASA X3.4.2) *Suggestions on ALGOL 60 (Rome) Issues*, Comm. ACM 6 (1963) pp.20-23.

* also available as a Programming Research Group Technical Monograph.

INTRODUCTION

This paper presents the 'semantic clauses' of ALGOL 60, using the methods developed at Oxford by Professor C. Strachey and others. The language described is that specified in the Revised Report on ALGOL 60 [5] (referred to below as "the Report"), except that 'own' declarations have been omitted - this will be discussed below.

The dividing lines between syntax and semantics, and semantics and implementation, are rather hazy - especially those between the latter two. The policy taken here has been to define primitive operations, such as *ApplyFn* and *Jump*, in a minimal fashion, and to give only axioms about the store-management functions. An implementation of this semantics could stipulate new definitions of these operations, but should preserve any theorems deducible from the original definitions and axioms (i.e. under some suitable formalism, e.g. that of the language LAMBDA [7]).

The mathematics and the comments upon it are presented separately, with the aim of exhibiting the structure of the semantic functions more clearly. In the commentary, ¶... refers to a section of the Report. The commentary on a function is headed by the name of that function, and an index is given to all functions, together with an indication of their types.

As in any large program before 'debugging', there will probably be several syntactical and semantical errors in this description. However, the author hopes soon to have a 'compiler' for semantic descriptions, the use of which should increase one's degree of belief in their correctness - this project is to form part of the author's thesis, to be submitted in supplication for the degree of D.Phil.

For the mathematical justification of the approach used here, see [6, 8, 9, 10, 11]. Also of interest as tutorial papers, in using and understanding semantic clauses, are [12, 13].

In connection with the omission of 'own' declarations, see [2, 14]. The doubts expressed in [14], about the lack of initialisation of 'own' identifiers, seem well-founded, as the semantics of the ALGOL 60 construction is very untidy. A more natural construction might be to allow initialised definitions in procedure headings, so that the scope of the definition is the body of the procedure, whilst its extent is the same as that of the procedure identifier. This suggestion was made by Landin in [3], and can be incorporated into the given syntax and semantics at almost no cost.

This report is here put forward less as 'the last word' on ALGOL 60 semantics, than as an experiment in using the Scott-Strachey semantic method to describe practical programming languages.

Any comments on the report, or suggestions for its improvement, will be very welcome.

ACKNOWLEDGEMENTS

The original inspiration for this report came from reading [1] and [3], as it was felt that a shorter and less algorithmic description of ALGOL 60 could be formulated in the Scott-Strachey semantics.

Many thanks are due to the members of the Programming Research Group, Oxford University, who studied earlier versions of this report and made many helpful comments.

This report was written whilst the author was being supported by an SRC Research Studentship.

SYNTAX

```
Prog → Sta

Sta  → begin DecL DefL StaL end
     → begin StaL end
     → if Exp then Sta1 else Sta2
     → Ide :  Sta
     → goto Exp
     → Var := AssL
     → for Var := ForL do Sta
     → Ide(ExpL)
     → Λ

StaL → Sta ; StaL
     → Sta

DecL → Dec {; Dec}*| Λ

Dec  → Type IdeL
     → Type IdeL[BdsL]

IdeL → Ide {, Ide}*

BdsL → Bds {, Bds}*

Bds  → Exp1 : Exp2

DefL → Def {; Def}* | Λ

Def  → switch Ide := ExpL
     → Type Ide(ParL); Sta

ParL → Par {,Par}* | Λ

Par  → Type Ide name
     → Type Ide value
```

```
Type → real | integer | boolean
     → array | Type array
     → procedure | Type procedure
     → label | string | switch

AssL → Var := AssL
     → Exp

ForL → For {, For}*

For  → Exp
     → Exp1 while Exp2
     → Exp1 step Exp2 until Exp3

ExpL → Exp {, Exp}* | Λ

Exp  → if Exp1 then Exp2 else Exp3
     → Exp1 Op Exp2
     → Op Exp
     → Ide(ExpL)
     → Ide[ExpL]
     → Ide
     → Const
     → Str
     → (Exp)

Var  → Ide[ExpL]
     → Ide

Op   → LogOp
     → RelOp
     → NumOp
```

LogOp → ≡ | ⊃ | ∨ | ∧ | ¬

RelOp → < | ≤ | = | ≠ | ≥ | >

NumOp → + | - | × | / | ÷ | ↑

Const → true | false
→ P INT
→ P REAL

Str → P STRING

Ide → P IDE

DOMAINS

(i) Standard Domains:

I (identifiers)
N (integers)
O (empty domain)
Q (strings)
T {true,false}

(ii) Syntactic Domains:

AssL
Bds
BdsL
Const
Dec
DecL
Def
DefL
El = Bds + Dec + Def + Exp + Ide + Par
Exp
ExpL
For
ForL
IDE (undefined)
Ide
IdeL
INT (undefined)
List = BdsL + DecL + DefL + ExpL + IdeL + ParL
LogOp
NumOp
Op
Par
ParL
Prog

REAL (undefined)
RelOp
Sta
StaL
Str
STRING (undefined)
Type
Var

(iii) Semantic Domains:

ActiveFn = *MakeActiveFn*(*ResLocn*:Locn, *Fn*:Fn)
Area (indicating locations in use)
Array = *MakeArray*(*BdsL*:Bds*, *LocnL*:Locn*)
Bds = *MakeBds*(*LBd*:N, *UBd*:N)
C = S → S
D = Locn + Array + Switch + Fn + ActiveFn + Rt + Label + String + Name
Den = ⟨D, Typ⟩
E = D + V + Bds
Fn = Param* → W
G = C → C
K = E → C
Label = *MakeLabel*(*ProperArea*:Area, *Code*:C)
Locn (addresses of real,integer and boolean values)
M = {"ev", "jv", "lv", "rv"}
Map (associating locations with values)
Name = M → W
Param = Tyn → M → W
R (real numbers)
Rt = Param* → G
S = *MakeS*(*SArea*:Area, *SMap*:Map)
String = (ALGOL 60 strings)
Switch = N → W

$Typ = Typ_1 + Typ_2 + \ldots + Typ_7$

$Typ_1 = \mathit{MakeTyp}(\mathit{Main}{:}X_1,\ \mathit{Qual}{:}0)$

$Typ_2 = \mathit{MakeTyp}(\mathit{Main}{:}X_2,\ \mathit{Qual}{:}Typ_1)$

$Typ_3 = \mathit{MakeTyp}(\mathit{Main}{:}X_3,\ \mathit{Qual}{:}0)$

$Typ_4 = \mathit{MakeTyp}(\mathit{Main}{:}X_4,\ \mathit{Qual}{:}Typ_1)$

$Typ_5 = \mathit{MakeTyp}(\mathit{Main}{:}X_5,\ \mathit{Qual}{:}0)$

$Typ_6 = \mathit{MakeTyp}(\mathit{Main}{:}X_6,\ \mathit{Qual}{:}Typ_1+Typ_2+Typ_3+Typ_4+Typ_5)$

$Typ_7 = \mathit{MakeTyp}(\mathit{Main}{:}X_7,\ \mathit{Qual}{:}Typ_4)$

$U = I \to Den$

$V = N + R + T$

$W = K + C$

$X = X_1 + X_2 + \ldots + X_7$

X_1 = {"real", "integer", "boolean", "num"}

X_2 = {"array"}

X_3 = {"label"}

X_4 = {"fn"}

X_5 = {"rt", "string", "switch"}

X_6 = {"name"}

X_7 = {"active"}

(iv) Denotation Domains of Bound Variables:

α : Locn

β : T

γ : G

δ : D

ε : Basic

ζ - untyped

η : Area

θ : C

ι : I

κ : K + [E* → C]

(λ)

μ : M

ν : N

ξ : N + R

(o)

π : Param

ρ : U

σ : S

τ : Typ

υ : M → W

φ - untyped

χ : X

ψ : Bds

ω : W

t denotes a "deduction tree" belonging to a syntactic domain.

SEMANTIC FUNCTIONS

```
compiler λt:Prog. λρ0. λθ0.
          let τ1 = MakeTyp("fn", MakeTyp("real", ?)) in
          let τ2 = MakeTyp("fn", MakeTyp("integer", ?)) in
          let ρ1 = ρ0[Abs/τ1/id"abs"]
                     [Sign/τ2/id"sign"]
                     [Sqrt/τ1/id"sqrt"]
                     [Sin/τ1/id"sin"]
                     [Cos/τ1/id"cos"]
                     [Arctan/ τ1 / id"arctan"]
                     [Ln/τ1/id"ln"]
                     [Exp/τ1/id"exp"]
                     [Entier/τ2/id"entier"]
          in
          switch labelof t in
          §
          case"Sta":𝔇⟦Sta⟧ρ1θ0
          §

def 𝔇⟦t:Sta⟧ρθ =
     let (ι*,τ*) = (𝔍*lab⟦t⟧, 𝔗*lab⟦t⟧) in
        Area ‖
          λη. ℭ⟦t⟧ρ[ (fix δ*. 𝔊⟦t⟧ρ[δ*/τ*/ι*]ηθ) /τ*/ι*] ‖ θ

def ℭ*⟦t:StaL⟧ρθ = switch labelof t in
§
case"Sta ; StaL": ℭ⟦Sta⟧ρ ‖ ℭ*⟦StaL⟧ρ ‖ θ
case"Sta":        ℭ⟦Sta⟧ρθ
§
```

```
def 𝔈⟦t:Sta⟧ρθ = switch labelof t in
§
case"begin DecL DefL StaL end":
    let (ι*₁,τ*₁) = (𝔍*dec⟦DecL⟧, 𝔗*dec⟦DecL⟧) in
    let (ι*₂,τ*₂) = (𝔍*def⟦DefL⟧, 𝔗*def⟦DefL⟧) in
    let (ι*₃,τ*₃) = (𝔍*lab⟦StaL⟧, 𝔗*lab⟦StaL⟧) in
    Indistinct(ι*₁ cat ι*₂ cat ι*₃) → ?,
         Area ‖
          λη₁.𝔇*⟦DecL⟧ρ[?/ ?/ ι*₁cat ι*₂ cat ι*₃] ‖
           λδ*₁. Area ‖
            λη₂. let ρ₁ = ρ[δ*₁/τ*₁/ι*₁] in
                 let θ₁ = SetArea(η₁){θ} in
                𝔈*⟦StaL⟧ρ₁[(fix δ*. let ρ₂ = ρ₁[δ*/τ*₂catτ*₃/ι*₂catι*₃] in
                                     ℋ*⟦DefL⟧ρ₂ cat 𝔊*⟦StaL⟧ρ₂η₂θ₁)
                             / τ*₂ cat τ*₃ / ι*₂ cat ι*₃] ‖ θ₁

case"begin StaL end": 𝔈*⟦StaL⟧ρθ

case"if Exp then Sta₁ else Sta₂":
   ℛ⟦Exp⟧ρ"boolean" {λβ. β → 𝔈⟦Sta₁⟧ρθ, 𝔈⟦Sta₂⟧ρθ}

case"Ide: Sta":  let (δ,τ) = ρ⟦Ide⟧ in Hop(δ)

case"goto Exp": 𝔍⟦Exp⟧ρ"label" ‖ λδ. Jump(δ)

case"Var := AssL":
    let χ = Main(𝔗var⟦Var⟧ρ) in 𝔄⟦t⟧ρχ() ‖ θ

case"for Var := ForL do Sta":
    let τ = 𝔗var⟦Var⟧ρ in Mainτ = "boolean" → ?,
    𝔉*⟦ForL⟧ρ(Mainτ)(𝒱⟦Var⟧ρτ)(𝔓⟦Sta⟧ρ) ‖ θ

case"Ide(ExpL)":
    Coerce(ρ⟦Ide⟧)(MakeTyp("rt",?))"ev" ‖
     λδ. ApplyRt(δ)(𝔘*⟦ExpL⟧ρ){θ}

case"Λ":  θ
§
```

```
def 𝒟*⟦t:DecL⟧ρκ = ∏(𝔛₂⟦t⟧(λt₁.𝒟⟦t₁⟧ρ)) ∥ κ

def 𝒟⟦t:Dec⟧ρκ = switch labelof t in
§
case"Type IdeL":
      let τ = 𝒯⟦Type⟧ in ∏(𝔛₁⟦IdeL⟧(λt₁. Newτ)) ∥ κ
case"Type IdeL[BdsL]":
      let τ = 𝒯⟦Type⟧ in
      ℬ⟦BdsL⟧ρ ∥ λψ*. ∏(𝔛₁⟦IdeL⟧(λt₁. NewArrayτψ*)) ∥ κ
§

def ℋ*⟦t:DefL⟧ρ = 𝔛₁⟦t⟧(λt₁.ℋ⟦t₁⟧ρ)

def ℋ⟦t:Def⟧ρ = switch labelof t in
§
case"switch Ide := ExpL":
      let ω* = 𝔛₁⟦ExpL⟧(λt₁. 𝒥⟦t₁⟧ρ"label") in λν. ω*↓ν
case"Type Ide(ParL); Sta":
      switch labelof "Type"of t in
      §
      case"procedure":
      λπ*. λθ.
       Area ∥
        λη. 𝒬*⟦ParL⟧π* ∥
         λδ*. 𝒫⟦Sta⟧ρ[δ*/𝒯*_par⟦ParL⟧/𝒥*_par⟦ParL⟧] ∥
          SetArea(η) ∥ θ
      case"Type procedure":
          let ⟨δ,τ⟩ = ρ⟦Ide⟧ in
          λπ*. λκ.
           Area ∥
            λη. New(Qualτ) ∥
             λα. let ⟨δ₁,τ₁⟩ = ⟨MakeActivePn(α,δ), MakeTyp("active",τ)⟩ in
                 𝒬*⟦ParL⟧π* ∥
                  λξ*. 𝒫⟦Sta⟧ρ[δ₁preδ*/ τ₁pre𝒯*_par⟦ParL⟧/ 𝒥⟦Ide⟧pre𝒥*_par⟦ParL⟧] ∥
                   Contents(α) ∥
                    λβ. SetArea(η) ∥ κ(β)
      §
§
```

```
def Q*⟦t:ParL⟧ π*κ = ∏(𝔛₃⟦t⟧(π*)(Q)) ‖ κ

def Q⟦t:Par⟧πκ = switch labelof t in
§
case"Type Ide  name":  κ(π(𝔍⟦Type⟧))
case"Type Ide  value":
      let τ = 𝔍⟦Type⟧ in
      Mainτ =  "label" → π(τ)"jv" ‖ κ ,
      Mainτ =  "array" → π(τ)"rv" ‖ λδ. CopyArrayδτ ‖ κ ,
                        π(τ)"rv" ‖ λε. Newτ ‖ λα. Setαε ‖ κ(α)
§

def 𝔖*⟦t:StaL⟧ ρηθ = switch labelof t in
§
case"Sta ; StaL":  𝔖⟦Sta⟧ρη(ℭ*⟦StaL⟧ρθ) cat 𝔖*⟦StaL⟧ρηθ
case"Sta":  𝔖⟦Sta⟧ρηθ
§

def 𝔖⟦t:Sta⟧ρηθ = switch labelof t in
§
case"begin DecL DefL StaL end":  ()
case"begin StaL end":  𝔖*⟦StaL⟧ρηθ
case"if Exp then Sta₁ else Sta₂":  𝔖⟦Sta₁⟧ρηθ cat 𝔖⟦Sta₂⟧ρηθ
case"Ide: Sta":  MakeLabel(η, ℭ⟦Sta⟧ρθ) pre 𝔖⟦Sta⟧ρηθ
case"goto Exp":
case"Var := AssL":
case"for Var := ForL do Sta":
case"Ide(ExpL)":
case"Λ":            ()
§

def 𝔄⟦t:AssL⟧ ρχα*θ = switch labelof t in
§
case"Var := AssL":  𝔏⟦Var⟧ρχ ‖ λα.𝔄⟦AssL⟧ρχ(α pre α*) ‖ θ
case"Exp":        ℜ⟦Exp⟧ρχ ‖ λε. SetMany(α)(ε) ‖ θ
§
```

```
def 𝓕*⟦t:ForL⟧ρχυγθ = 𝔛₄⟦t⟧(λt₁. 𝓕⟦t₁⟧ρχυγ) ∥ θ

def 𝓕⟦t:For⟧ρχυγθ = switch labelof t in
§
case"Exp":  υ"lv" ∥
              λα. ℛ⟦Exp⟧ρχ ∥
               λξ. Setαξ ∥ θ

case"Exp₁ while Exp₂":
      fix θ'. υ"lv" ∥
                λα. ℛ⟦Exp₁⟧ρχ ∥
                 λξ. Setαξ ∥
                 ℛ⟦Exp₂⟧ ρχ ∥
                   λβ. β → γ{θ'}, θ
case"Exp₁ step Exp₂ until Exp₃":
      υ"lv" ∥
       λα.ℛ⟦Exp₁⟧ρχ ∥
        λξ₁. Setαξ₁ ∥
         fix θ'. ∏(υ"rv", ℛ⟦Exp₂⟧ρχ, ℛ⟦Exp₃⟧ρχ) ∥
                λ⟨ξ,ξ₂,ξ₃⟩ . Finished(ξ,ξ₂,ξ₃) → θ,
                            γ{υ"lv" ∥
                               λα'. ∏(υ"rv", ℛ⟦Exp₂⟧ρχ) ∥
                                λ⟨ξ',ξ₂'⟩ . Set(α')(Plus(ξ',ξ₂')) ∥ θ'}
§

def 𝔍⟦t:Ide⟧ = IdeVal("IDE"of t)
def 𝔍*dec⟦t:DecL⟧ = 𝔛₂⟦t⟧(𝔍dec)
def 𝔍dec⟦t:Dec⟧ = 𝔛₁⟦IdeL⟧(𝔍)
def 𝔍*def⟦t:DefL⟧ = 𝔛₁⟦t⟧(𝔍def)
def 𝔍def⟦t:Def⟧ = 𝔍⟦Ide⟧
def 𝔍*par⟦t:ParL⟧ = 𝔛₁⟦t⟧(𝔍par)
def 𝔍par⟦t:Par⟧ = 𝔍⟦Ide⟧

def 𝔍*lab⟦t:StaL⟧ = switch labelof t in
§
case"Sta ; StaL":  𝔍lab⟦Sta⟧ cat 𝔍*lab⟦StaL⟧
case"Sta":         𝔍lab⟦Sta⟧
§
```

```
def 𝔍_lab[t:Sta] = switch labelof t in
§
case"begin DecL DefL StaL end":  ()
case"begin StaL end":             𝔍*_lab[StaL]
case"if Exp then Sta_1 else Sta_2":  𝔍_lab[Sta_1] cat 𝔍_lab[Sta_2]
case"Ide: Sta":                   𝔍[Ide] pre 𝔍_lab[Sta]
case"goto Exp":
case"Var := AssL":
case"for Var := ForL do Sta":
case"Ide(ExpL)":
case"Λ":                          ()
§

def 𝔗[t:Type] = switch labelof t in
§
case"real":
case"integer":
case"boolean":  MakeTyp(labelof t, ?)
case"array":    MakeTyp("array", MakeTyp("real",?))
case"Type array":  MakeTyp("array",𝔗[Type])
case"procedure":   MakeTyp("rt", ?)
case"Type procedure":  MakeTyp("fn", 𝔗[Type])
case"label":
case"string":
case"switch": MakeTyp (labelof t, ?)
§

def 𝔗*_dec[t:DecL] = 𝔛_2[t](𝔗_dec)
def 𝔗_dec[t:Dec] = let τ = 𝔗[Type] in 𝔛_1[IdeL](λt'. τ)
def 𝔗*_def[t:DefL] = 𝔛_1[t](𝔗_def)
def 𝔗_def[t:Def] = 𝔗[Type]
def 𝔗*_par[t:ParL] = 𝔛_1[t](𝔗_par)

def 𝔗_par[t:Par] = switch labelof t in
§
case"Type Ide name":  MakeTyp("name", 𝔗[Type])
case"Type Ide value": 𝔗[Type]
§
```

```
def 𝔍*_lab⟦t:StaL⟧ = switch labelof t in
§
case"Sta ; StaL":  𝔍_lab⟦Sta⟧ cat 𝔍*_lab⟦StaL⟧
case"Sta":         𝔍_lab⟦Sta⟧
§

def 𝔍_lab⟦t:Sta⟧ = switch labelof t in
§
case"begin DecL DefL StaL end":  ⟨⟩
case"begin StaL end":            𝔍*_lab⟦StaL⟧
case"if Exp then Sta_1 else Sta_2":𝔍_lab⟦Sta_1⟧ cat 𝔍_lab⟦Sta_2⟧
case"Ide: Sta":  MakeTyp("label", ?) pre 𝔍_lab⟦Sta⟧
case"goto Exp":
case"Var := AssL":
case"for Var := ForL do Sta":
case"Ide(ExpL)":
case"Λ":                         ⟨⟩
§

def 𝔍_var⟦t:Var⟧ρ = let⟨δ,τ⟩ = ρ⟦Ide⟧ in BasicTyp(τ)

def 𝔍_res⟦t:Op⟧ = switch labelof t in
§
case"LogOp":
case"RelOp":  MakeTyp("boolean", ?)
case"NumOp":  MakeTyp("num", ?)
§

def 𝔍_arg⟦t:Op⟧ = switch labelof t in
§
case"LogOp":  MakeTyp("boolean", ?)
case"RelOp":
case"NumOp":  MakeTyp("num", ?)
§
```

```
def 𝒯const⟦t:Const⟧ = switch labelof t in

case"P REAL":  MakeTyp("real", ?)
case"P INT":   MakeTyp("integer", ?)
case"true"
case"false":   MakeTyp("boolean", ?)

def 𝒱⟦t:Exp⟧ρτ1μκ =
    let χ1 = Main τ1 in
    switch μ in
    §
    case"ev":
         switch labelof t in
         §
         case"Ide":  Coerce(ρ⟦Ide⟧)τ1μκ
         case"Str":  χ1≠"string" → ? , κ(𝒮⟦Str⟧)
         $
    case"jv" :
         switch labelof t in
         §
         case"if Exp1 then Exp2 else Exp3":
             ℛ⟦Exp1⟧ρ"boolean"{λβ.β → 𝒱⟦Exp2⟧ρτ1μκ, 𝒱⟦Exp3⟧ρτ1μκ}
         case"Ide[ExpL]":
             χ1≠"label" → ?, Coerce(ρ⟦Ide⟧)(MakeTyp("switch",?))"ev" ‖
                                  λδ 𝒩1⟦ExpL⟧ρ ‖ λν. δ(ν){κ}
         case"Ide":  Coerce(ρ⟦Ide⟧)τ1μκ
         $
    case"lv" :
         switch labelof t in
         §
         case"Ide[ExpL]":  Coerce(ρ⟦Ide⟧)(MakeTyp("array",τ1))"ev" ‖
                                  λδ. 𝒩*⟦ExpL⟧ρ ‖ λν*. κ(Accessδν*)
         case"Ide":  Coerce(ρ⟦Ide⟧)τ1μκ
         $
```

```
case"rv":
    let κ₁ = (χ₁="real"∨ χ₁="integer") → κ∘Transferχ₁, κ in
    switch labelof t in
    §
    case"if Exp₁ then Exp₂ else Exp₃":
       ℛ⟦Exp₁⟧ρ"boolean"{λβ. β → 𝒱⟦Exp₂⟧ρτ₁μκ, 𝒱⟦Exp₃⟧ρτ₁μκ}
    case"Exp₁ Op Exp₂":
        let (χ,χ′) = (Main(𝒯_res⟦Op⟧), Main(𝒯_arg⟦Op⟧)) in
        ~Goodχχ₁μ → ? ,
           𝒯(ℛ⟦Exp₁⟧ρχ′, ℛ⟦Exp₂⟧ρχ′) ∥ κ₁∘𝒲₂⟦Op⟧
    case"Op Exp":
        let (χ,χ′) = (Main(𝒯_res⟦Op⟧), Main(𝒯_arg⟦Op⟧))in
        ~Goodχχ₁μ → ? ,
           ℛ⟦Exp⟧ρχ′ ∥ κ₁∘𝒲₁⟦Op⟧
    case"Ide(ExpL)":
        Coerce(ρ⟦Ide⟧)(MakeTyp("fn",τ₁))μ ∥
            λδ. ApplyFn(δ)(𝒰*⟦ExpL⟧ρ){κ₁}
    case"Ide[ExpL]":
        Coerce(ρ⟦Ide⟧)(MakeTyp("array",τ₁))μ ∥
            λδ.𝒩*⟦ExpL⟧ρ ∥ λν*. Contents(Accessδν*) ∥ κ₁
    case"Ide" :
        Coerce(ρ⟦Ide⟧)τ₁μκ₁
    case"Const":
        ~Good(Main(𝒯_const⟦Const⟧))χ₁μ → ?, κ₁(𝒦⟦Const⟧)
    case"(Exp)":
       𝒱⟦Exp⟧ρτ₁μκ
    §
§
```

```
def 𝒥⟦t:Exp⟧ρχκ = 𝒱⟦t⟧ρ(MakeTyp(χ,?))"jv"κ
def ℒ⟦t:Var⟧ρχκ = 𝒱⟦t⟧ρ(MakeTyp(χ,?))"lv"κ
def ℛ⟦t:Exp⟧ρχκ = 𝒱⟦t⟧ρ(MakeTyp(χ,?))"rv"κ
def ℬ*⟦t:BdsL⟧ρκ = Π_n(𝔛_1⟦t⟧(λt_1.ℬ⟦t_1⟧ρ)) ∥ κ
def ℬ⟦t:Bds⟧ρκ = Π(𝒩⟦Exp_1⟧ρ 𝒩⟦Exp_2⟧ρ) ∥ κ∘MakeBds
def 𝒩*⟦t:ExpL⟧ρκ = Π_0(𝔛_1⟦t⟧(λt_1.𝒩⟦t_1⟧ρ)) ∥ κ
def 𝒩⟦t:Exp⟧ρκ = ℛ⟦t⟧ρ"int"κ
def 𝒩_1⟦t:ExpL⟧ρκ = dimof t ≠ 1 → ?,𝒩⟦1 of t⟧ρκ
def 𝒰*⟦t:ExpL⟧ρ = 𝔛_1⟦t⟧(λt_1.𝒱⟦t_1⟧ρ)
def 𝒮⟦t:Str⟧ = StringVal("STRING"of t)

def 𝒦⟦t:Const⟧ = switch labelof t in
§
case"P REAL":  RealVal("REAL"of t)
case"P INT":   IntVal("INT"of t)
case"true":    true
case"false":   false
§

def 𝒲_1⟦t:Op⟧ε = switch labelof(1 of t) in
§
case"¬":  Not ε
case"+":  ε
case"-":  Negate ε
§

def 𝒲_2⟦t:Op⟧(ε,ε_1) = switch labelof(1 of t) in
§
case"≡":  Eqv(ε,ε_1)
case"⊃":  Imp(ε,ε_1)
case"∨":  Or(ε,ε_1)
case"∧":  And(ε,ε_1)
case"<":  Lt(ε,ε_1)
case"≤":  Le(ε,ε_1)
case"=":  Eq(ε,ε_1)
case"≠":  Ne(ε,ε_1)
case"≥":  Ge(ε,ε_1)
case">":  Gt(ε,ε_1)
```

```
case"+":  Plus(ε,ε1)
case"-":  Minus(ε,ε1)
case"×":  Mult(ε,ε1)
case"/":  RDiv(ε,ε1)
case"÷":  IsIntε ∧ IsIntε1 →
               let ε' = RDiv(ε,ε1) in
               Mult(Signε', Entier(Absε')),
          ?
case"↑":  IsIntε1 →
               Eq(Zero,ε1) →
                    {Ne(Zero,ε) → One, ? },
                    {let ε' = Iter(Int(Absε1))(λε2. Mult(ε2,ε))(One) in
                     Gt(Zero,ε1) → ε', RDiv(One,ε')},
          IsRealε1 →
               Eq(Zero,ε) →
                    {Gt(Zero,ε1) → Real(Zero),?},
               Gt(Zero,ε) →
                     Exp(Mult(ε1,Lnε)),
               ?,
          ?
∮

def 𝔛1⟦t:List⟧φ = 𝔛2⟦t⟧(λt1. ⟨φ⟦t1⟧⟩)
def 𝔛2⟦t:List⟧φ = CatMap(dimof t)(λν. φ⟦ν of t⟧)
def 𝔛3⟦t:ParL⟧π*φ = dimof t ≠ dimof π* → ?,
                       CatMap(dimof t)(λν. (φ⟦ν of t⟧(π*↓ν )))
def 𝔛4⟦t:ForL⟧φθ = Compound(dimof t)(λν. φ⟦ν of t⟧)(θ)
```

AUXILIARY FUNCTIONS

(i) Defined:

```
def ApplyFn(δ:Fn)π*κ = δπ*κ
def ApplyPt(δ:Rt)π*θ = δπ*θ
def Areaκσ = κ(SArea(σ))(σ)
def BasicTyp(τ) = switch Mainτ in
§
case"name":
case"active":
case"fn":
case"array":  BasicTyp(Qualτ)
case"real":
case"integer":
case"boolean":  τ
default:  ?
§
def Coerce(δ,τ)τ₁μκ =
    let (χ,τ') = (Mainτ, Qualτ) in
    let(χ₁,τ₁') = (Mainτ₁, Qualτ₁) in
    switch χ in
    §
    case"name":   δ(μ){λδ'. Coerce(δ',τ')τ₁μκ
    case"active":  μ="ev" ∨ μ="rv" → Coerce(Fnδ,τ')τ₁κ,
                   μ="lv" → Coerce(Locnδ,Qualτ')τ₁κ,?
    case"fn":      μ="ev" ∧ χ₁="fn" ∧ Good(Mainτ')(Mainτ₁')(μ) → κ(δ),
                   μ="ev" ∧ χ₁="rt" → κ(λπ*.λθ. δπ*{λε.θ}),
                   μ="rv" ∧ Good(Mainτ')(χ₁)μ → ApplyFn(δ)() {κ}, ?
    case"array":  (μ="ev"∨ μ="rv")∧ χ₁="array" ∧ Good(Mainτ')(Mainτ₁')(μ)
                                                    → κ(δ), ?

    case"real":
    case"integer":
    case"boolean":  μ="lv" ∧ Goodχχ₁μ → κ(δ),
                    μ="rv" ∧ Goodχχ₁μ → Contentsδ{κ}, ?
```

```
    case"label":  μ="jv" ∧ χ1="label" → κ(δ), ?
    case"rt":
    case"string":
    case"switch":  μ="ev" ∧ χ1=χ → κ(δ), ?
    $

def Finished(ξ1,ξ2,ξ3) = Lt(Mult(Minus(ξ3,ξ1), Sign(ξ2)), Zero)
def Goodχχ1μ = switch μ in
§
case"ev":
case"lv":  χ=χ1
case"rv":  χ="boolean" → χ1=χ,
           χ="integer" ∨ χ="real" → (χ1="integer"∨χ1="real"∨χ1="num"),
                                     false
default:  false
$

def Hop(δ:Label) = Code(δ)
def Int(ξ) = Entier(Plus(ξ,Half))
def Jump(δ:Label) = SetArea(ProperAreaδ) ∥ Code(δ)
def SetAreaηθσ = θ(MakeS(η,SMapσ))
def SetManyα*εθ = Compound(dimof α*)(λν. Set(α*↓ν)(ε))(θ)
def Transferχξ =
    χ="real" → Realξ
    χ="integer" → Intξ, ?
```

(ii) Informally defined:

```
def CatMap(ν)(φ) = φ(1) cat φ(2) cat ... cat φ(ν)
def Compound(ν)(φ)(θ) = φ(1) ∥ φ(2) ∥ ... ∥ φ(ν) ∥ θ
def Indistinct(ι*) = let ν = dimof ι* in
                     (ι*↓1=ι*↓2) ∨ (ι*↓1=ι*↓3) ∨ ... ∨ (ι*↓1=ι*↓ν)
                     ∨             (ι*↓2=ι*↓3) ∨ ... ∨ (ι*↓2=ι*↓ν)
                                          . . . .
                                                     ∨ (ι*↓(ν-1)=ι*↓ν)
```

$$\text{def } \mathit{Inside}\,\psi^*\nu^* = \text{let } \nu' = \text{dimof } \nu^* \text{ in}$$
$$LBd(\psi^*\downarrow 1) \le \nu^*\downarrow 1 \le UBd(\psi^*\downarrow 1)$$
$$LBd(\psi^*\downarrow 2) \le \nu^*\downarrow 2 \le UBd(\psi^*\downarrow 2)$$
$$\cdots\cdots$$
$$LBd(\psi^*\downarrow\nu') \le \nu^*\downarrow\nu' \le UBd(\psi^*\downarrow\nu')$$

$$\text{def } \mathit{Iter}(\nu)(\phi{:}\text{Basic}\to\text{Basic})(\varepsilon) = \underbrace{\phi(\phi(\ldots\phi}_{\nu \text{ occurrences of } \phi.}(\varepsilon)\ldots))$$

$$\text{def } \Pi\,\omega^*\kappa = \text{let } \nu = \text{dimof } \omega^* \text{ in}$$
$$\text{let } p = \mathit{SomePermof1to}(\nu) \text{ in}$$
$$\omega^*\downarrow p(1) \parallel \lambda\zeta_{p(1)}.\ \omega^*\downarrow p(2) \parallel \lambda\zeta_{p(2)}.\ \ldots\ \omega^*\downarrow p(\nu). \parallel \lambda\zeta_{p(\nu)}.\ \kappa(\zeta_1,\zeta_2,\ldots,\zeta_\nu)$$

$$\text{def } \Pi_0\,\omega^*\kappa = \text{let } \nu = \text{dimof } \omega^* \text{ in}$$
$$\omega^*\downarrow 1 \parallel \lambda\zeta_1.\ \omega^*\downarrow 2 \parallel \lambda\zeta_2.\ \ldots\ \omega^*\downarrow\nu \parallel \lambda\zeta_\nu.\ \kappa(\zeta_1,\zeta_2,\ldots,\zeta_\nu)$$

(iii) Restricting axioms:

We abbreviate as follows.

(a) Φ-eq E asserts that the argument of Φ is true, i.e. that axiom $E[T/\Phi] = E[K/\Phi]$
where
$T = \lambda\beta.\ \beta\to 1,\ ?$
$K = \lambda\beta.\ I$
$I = \lambda\sigma.\ \sigma$

(b) axiom $E_1 \leftrightarrow E_2$ denotes
axiom $\Pi(E_1,E_2) = \Pi_0(E_1,E_2)$ (i.e. E_1 and E_2 commute).

(c) Free variables are universally quantified over their domains.

Φ-eq $New\tau \parallel \lambda\alpha.\ InArea\alpha \parallel \lambda\beta.\ \Phi(\beta)$

Φ-eq $New\tau \parallel \lambda\alpha.\ Contents\alpha \parallel \lambda\varepsilon.\ \Phi(\varepsilon=?)$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ New\tau \parallel \lambda\alpha_1.\ \Phi(\beta \supset \alpha\neq\alpha_1)$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ Set\alpha\varepsilon \parallel Contents\alpha \parallel \lambda\varepsilon_1.\ \Phi(\varepsilon=\varepsilon_1)$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ Contents\alpha \parallel \lambda\varepsilon.\ Contents\alpha \parallel \lambda\varepsilon_1.\ \Phi(\varepsilon=\varepsilon_1)$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ NewArray\tau\psi^\star \parallel \lambda\delta.\ \Phi(\beta \supset (Inside\psi^\star\nu^\star \supset Access\delta\nu^\star\neq\alpha))$

Φ-eq $NewArray\tau\psi^\star \parallel \lambda\delta.\ \Phi(BdsL(\delta)=\psi^\star)$

Φ-eq $NewArray\tau\psi^\star \parallel \lambda\delta.\ \Phi((Inside\psi^\star\nu^\star \wedge Inside\psi^\star\nu_1^\star \wedge Access\delta\nu^\star=Access\delta\nu_1^\star) \supset \nu^\star=\nu_1^\star)$

Φ-eq $NewArray\tau\psi^\star \parallel \lambda\delta.\ InArea(Access\delta\nu^\star) \parallel \lambda\beta.\ \Phi(Inside\psi^\star\nu^\star \supset \beta)$

Φ-eq $NewArray\tau\psi^\star \parallel \lambda\delta.\ Contents(Access\delta\nu^\star) \parallel \lambda\varepsilon.\ \Phi(Inside\psi^\star\nu^\star \supset \varepsilon=?)$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ NewArray\tau\psi^\star \parallel \lambda\delta.\ \Phi(\beta \supset (Inside\psi^\star\nu^\star \supset Access\delta\nu^\star\neq\alpha))$

Φ-eq $CopyArray\delta_1\tau \parallel \lambda\delta.\ \Phi(BdsL(\delta)=BdsL(\delta_1))$

Φ-eq $CopyArray\delta_1\tau \parallel \lambda\delta.\ InArea(Access\delta\nu^\star) \parallel \lambda\beta.\ \Phi(Inside\psi^\star\nu^\star \supset \beta)$

Φ-eq $CopyArray\delta_1\tau \parallel \lambda\delta.\ Contents(Access\delta\nu^\star) \parallel \lambda\varepsilon.\ Contents(Access\delta_1\nu^\star) \parallel \lambda\varepsilon_1.\ \Phi(Inside(BdsL(\delta))(\nu^\star) \supset \varepsilon=Transfer(BasicTyp\tau)(\varepsilon_1))$

Φ-eq $InArea\alpha \parallel \lambda\beta.\ CopyArray\delta_1\tau \parallel \lambda\delta.\ \Phi(\beta \supset (Inside(BdsL(\delta))(\nu^\star) \supset Access\delta\nu^\star\neq\alpha))$

axiom $\alpha\neq\alpha_1 \supset Contents\alpha \leftrightarrow \lambda\kappa.\ Set\alpha_1\varepsilon\{\kappa(?)\}$

axiom $\alpha\neq\alpha_1 \supset Contents\alpha \leftrightarrow InArea\alpha_1$

axiom $\alpha\neq\alpha_1 \supset Contents\alpha \leftrightarrow Contents\alpha_1$

INDEX

$\mathfrak{T}_{const}[\![$ t:Const $]\!] = \tau$	19,	C15
$\mathfrak{T}_{dec}[\![$ t:Dec $]\!] = \tau^*$	17,	C15
$\mathfrak{T}^*_{dec}[\![$ t:DecL $]\!] = \tau^*$	17,	C15
$\mathfrak{T}_{def}[\![$ t:Def $]\!] = \tau$	17,	C15
$\mathfrak{T}^*_{def}[\![$ t:DefL $]\!] = \tau^*$	17,	C15
$\mathfrak{T}_{lab}[\![$ t:Sta $]\!] = \tau^*$	18,	C15
$\mathfrak{T}^*_{lab}[\![$ t:StaL $]\!] = \tau^*$	18,	C15
$\mathfrak{T}_{par}[\![$ t:Par $]\!] = \tau$	17,	C15
$\mathfrak{T}^*_{par}[\![$ t:ParL $]\!] = \tau^*$	17,	C15
$\mathfrak{T}_{res}[\![$ t:Op $]\!] = \tau$	18,	C15
$\mathfrak{T}_{var}[\![$ t:Var $]\!]\rho = \tau$	18,	C15
$\mathfrak{U}^*[\![$ t:ExpL $]\!]\rho = \pi^*$	21,	C17
$\mathfrak{V}[\![$ t:Exp $]\!]\rho\tau\mu\kappa = \theta$	19,	C15
$\mathfrak{W}_1[\![$ t:Op $]\!]\varepsilon = \varepsilon'$	21,	C17
$\mathfrak{W}_2[\![$ t:Op $]\!](\varepsilon,\varepsilon_1) = \varepsilon'$	21,	C17
$\mathfrak{X}_1[\![$ t:List $]\!]\phi = \omega^*$	22,	C17
$\mathfrak{X}_2[\![$ t:List $]\!]\phi = \omega^*$	22,	C17
$\mathfrak{X}_3[\![$ t:ParL $]\!]\pi^*\phi = \omega^*$	22,	C17
$\mathfrak{X}_4[\![$ t:ForL $]\!]\phi\theta = \theta'$	22,	C17

$Abs\ \xi = \xi'$	-	-
$Access\ \delta\nu^* = \sigma$	-	C20
$And\ (\beta,\beta_1) = \beta'$	-	-
$ApplyFn\ (\delta$:Fn$)\pi^*\kappa = \theta$	23,	C18
$ApplyPt\ (\delta$:Rt$)\pi^*\theta = \theta'$	23,	C18
$Arctan\ \xi = \xi'$	-	-
$Area\ \kappa = \theta$	23,	C18
$BasicType = \tau'$	23,	C18
$BdsL\ (\delta$:Array$) = \psi^*$	9,	-
$CatMap\ \nu\phi = \zeta^*$	24,	-
$Code\ (\delta$:Label$) = \theta$	9,	-
$Coerce\ (\delta,\tau)\tau_1\mu\kappa = \theta$	23,	C18
$Compound\ \nu\phi\theta = \theta'$	24,	-
$Content\ \sigma\kappa = \theta$	-	C20
$CopyArray\ \delta\tau\kappa = \theta$	-	C20
$Cos\ \xi = \xi'$	-	-

Entierξ	$= \nu$	-	-
Eq(ξ,ξ_1)	$= \beta$	-	-
Eqv(β,β_1)	$= \beta'$	-	-
Expξ	$= \xi'$	-	-
Finished(ξ_1,ξ_2,ξ_3)	$= \beta$	24,	C18
Fn$(\delta$:ActiveFn$)$	$= \delta$:Fn	9,	-
Ge(ξ,ξ_1)	$= \beta$	-	-
Good$\chi\chi_1\nu$	$= \beta$	24,	C18
Gt(ξ,ξ_1)	$= \beta$	-	-
Half	$= \xi$	-	-
Hop$(\delta$:Label$)$	$= \theta$	24,	C18
IdeVal(t:IDE)	$= \iota$	-	C3
Imp(β,β_1)	$= \beta'$	-	-
InArea$\alpha\kappa$	$= \theta$	-	C20
Indistinct(ι^*)	$= \beta$	24	-
Inside$\psi^*\nu^*$	$= \beta$	25,	C20
Intξ	$= \nu$	24,	C19
IntVal(t:INT)	$= \nu$	-	C3
IsIntξ	$= \beta$	-	-
IsRealξ	$= \beta$	-	-
Iter$\nu\phi\epsilon$	$= \epsilon'$	25	-
Jump$(\delta$:Label$)$	$= \theta$	24,	C19
LBd(ψ)	$= \nu$	9	-
Le(ξ,ξ_1)	$= \beta$	-	-
Lnξ	$= \xi'$	-	-
LocnL$(\delta$:Array$)$	$= \alpha^*$	9	-
Lt(ξ,ξ_1)	$= \beta$	-	-
Mainτ	$= \chi$	10	-
MakeActiveFn$(\delta$:Fn$,\alpha)$	$= \delta$	9	-
MakeArray(ψ^*,α^*)	$= \delta$	9	-
MakeBds(ν,ν_1)	$= \psi$	9	-
MakeLabel(η,γ)	$= \delta$	9	-
MakeS$(\eta,\phi$:Map$)$	$= \sigma$	9	-
MakeTyp(χ,τ)	$= \tau$	10	-
Minus(ξ,ξ_1)	$= \xi'$	-	-
Mult(ξ,ξ_1)	$= \xi'$	-	-

$Ne(\xi,\xi_1)$	$= \beta$	-	-
$Negate\,\xi$	$= \xi'$	-	-
$New\,\tau\kappa$	$= \theta$	-	C20
$NewArray\,\tau\psi^*\kappa$	$= \theta$	-	C20
$Not\,\beta$	$= \beta'$	-	-
One	$= \nu$	-	-
$Or(\beta,\beta_1)$	$= \beta'$	-	-
$Plus(\xi,\xi_1)$	$= \xi'$	-	-
$ProperArea(\delta$:Label)	$= \eta$	9	-
$Qual\,\tau$	$= \tau'$	10	-
$RDiv(\xi,\xi_1)$	$= \xi'$	-	-
$Real\,\xi$	$= \xi'$	-	-
$RealVal$(t:REAL)	$= \xi$	-	C3
$ResLocn(\delta$:ActiveFn)	$= \alpha$	9	-
$SomePermof1to(\nu)$	$= \zeta$:N→N	-	-
$SArea\,\sigma$	$= \eta$	9	-
$Set\,\alpha\varepsilon\theta$	$= \theta'$	-	-
$SetArea\,\eta\theta$	$= \theta'$	24,	C19
$SetMany\,\alpha^*\varepsilon\theta$	$= \theta'$	24,	C19
$Sign\,\xi$	$= \nu$	-	-
$Sin\,\xi$	$= \xi'$	-	-
$SMap\,\sigma$	$= \phi$:Map	9	-
$Sqrt\,\xi$	$= \xi'$	-	-
$StringVal$(t:STRING)	$= \delta$	-	C3
$Transfer\,\chi\xi$	$= \xi'$	24,	C19
$UBd\,\psi$	$= \nu$	9	-
$Zero$	$= \nu$	-	-
$\prod \omega^*\kappa$	$= \theta$	25,	C19
$\prod_0 \omega^*\kappa$	$= \theta$	25,	C19