This document defines the typing rules for our language. Typing rules are paired with grammar productions.

1. Syntax: S' -> S

Semantics: S'.type := S.type

1. Syntax: S -> D' C .

Semantics: if D'.type != type\_error and

C.type != type\_error then

S.type := program

else

S.type := type\_error

1. Syntax: S -> C .

Semantics: S.type := C.type

1. Syntax: D'1 -> D D'2

Semantics: if D.type == declaration and

D'2.type == declarations then

D'1.type := declarations

else

D'1.type := type\_error

1. Syntax: D' -> D

Semantics: if D.type == declaration then

D'.type := declarations

else

D'.type := type\_error

(Note: declaration vs declaration**s**)

1. Syntax: D -> let T id be E .

Semantics: if E.type != type\_error then

add\_type(id.entry, T.type);

D.type := declaration

else

D.type := type\_error

1. Syntax: T -> int

Semantics: T.type := integer

1. Syntax: T -> set

Semantics: T.type := set

1. Syntax: E -> E'

Semantics: E.type := E'.type

1. Syntax: E1 -> E2 U E'

Semantics: if E2.type == set and E'.type == set then

E1.type := set

else

E1.type := type\_error

1. Syntax: E1 -> E2 + E'

Semantics: if E2.type == integer and E'.type == integer then

E1.type := integer

else

E1.type := type\_error

1. Syntax: E1 -> E2 - E'

Semantics: if E2.type == integer and E'.type == integer then

E1.type := integer

else

E1.type := type\_error

1. Syntax: E' -> E''

Semantics: E'.type := E''.type

1. Syntax: E'1 -> E'2 I E''

Semantics: if E'2.type == set and E''.type == set then

E'1.type := set

else

E'1.type := type\_error

1. Syntax: E'1 -> E'2 \* E''

Semantics: if E'2.type == integer and E''.type == integer then

E'1.type := integer

else

E'1.type := type\_error

1. Syntax: E'' -> num

Semantics: E''.type := integer

1. Syntax: E'' -> id

Semantics: E''.type := lookup\_type(id.entry)

1. Syntax: E'' -> ( E )

Semantics: E''.type := E.type

1. Syntax: E'' -> { Z P }

Semantics: if P.type == predicate then

E''.type := set

else

E''.type := type\_error

1. Syntax: Z -> id :

Semantics: add\_type(id.entry, integer);

Z.type := void

(Note: In our language, sets are of only one type: set of integers. Thus, all representatives are of type integer.)

1. Syntax: P1 -> P2 | P'

Semantics: if P2.type == predicate and

P'.type == predicate then

P1.type := predicate

else

P1.type := type\_error

1. Syntax: P -> P'

Semantics: P.type := P'.type

1. Syntax: P'1 -> P'2 & P''

Semantics: if P'2.type == predicate and

P''.type == predicate then

P'1.type := predicate

else

P'1.type := type\_error

1. Syntax: P' -> P''

Semantics: P'.type := P''.type

1. Syntax: P'' -> R

Semantics: if R.type == relation and

P''.type := predicate

else

P''.type := type\_error

1. Syntax: P'' -> ( P )

Semantics: P''.type := P.type

1. Syntax: P'' -> ! R

Semantics: if R.type == relation and

P''.type := predicate

else

P''.type := type\_error

1. Syntax: R -> E1 < E2

Semantics: if E1.type == integer and E2.type == integer then

R.type := relation

else

R.type := type\_error

1. Syntax: R -> E1 > E2

Semantics: if E1.type == integer and E2.type == integer then

R.type := relation

else

R.type := type\_error

1. Syntax: R -> E1 = E2

Semantics: if E1.type == integer and E2.type == integer then

R.type := relation

else

R.type := type\_error

1. Syntax: R -> E1 @ E2

Semantics: if E1.type == integer and E2.type == set then

R.type := relation

else

R.type := type\_error

1. Syntax: C -> show A

Semantics: C.type := A.type

1. Syntax: A -> E

Semantics: if E.type != type\_error then

A.type := calculation

else

A.type := type\_error

1. Syntax: A -> P

Semantics: if P.type != type\_error then

A.type := calculation

else

A.type := type\_error