OUTPUT

Agregamos el comando

```
<comm> ::= !<intexp>
```

donde !e es el comando que escribe el valor de e. Claramente este comando puede encontrarse dentro de un ciclo o en cualquier fragmento de programa, eventualmente un programa puede generar una cantidad finita o infinita de output.

Los comportamientos posibles de un programa en un estado dado son ahora los siguientes:
1) se genera una cantidad finita de output y luego "se cuelga"
2) se genera una cantidad finita de output y luego termina
3) se genera una cantidad finita de output y luego falla
4) se genera una cantidad infinita de output

Sea Ω = conjunto de estos comportamientos.

```
Ω = {<n1,..,nk>} ∪ {<n1,..,nk,s>} ∪ {<n1,..,nk,<abort,s>>} ∪ {<n1,..,nk,...>}
```

En Ω se define la relación w ≤ w' cuando w es segmento inicial de w'. Con esta relación Ω es un dominio, donde el mínimo es la secuencia vacía <>. Las cadenas interesantes tienen supremo de la forma <n1,..,nk,...>.

```
[[]] ∈ <comm> -> Σ -> Ω

[[skip]]s = <s>
[[fail]]s = <<abort, s>>
[[v:= e]]s = <[s | v : [[e]]s]>
                            ⎧ [[c0]]s        si [[b]]s
[[if b then c0 else c1]]s = ⎨
                            ⎩ [[c1]]s        c.c.
[[c0;c1]]s = [[c1]]_* ([[c0]]s)
```

donde, dada una f ∈ Σ -> Ω, denotamos por f_* la siguiente extensión de f a Ω. Así, f_* ∈ Ω -> Ω definida por:

```
        ⎧ <n1,..,nk>              si x = <n1,..,nk>
f_* x = ⎨ <n1,..,nk> ++ f s       si x = <n1,..,nk,s>
        | <n1,..,nk,<abort,s>>    si x = <n1,..,nk,<abort,s>>
        ⎩ <n1,..,nk,...>          si x = <n1,..,nk,...>
```

o sea

```
        ⎧ <n1,..,nk> ++ f s       si x = <n1,..,nk,s>
f_* x = ⎨
        ⎩ x                       c.c.
```

También se puede definir así, f_* continua tal que:

```
        ⎧ <>                 si x = <>
f_* x = ⎨ f s                si x = <s>
        | <<abort,s>>        si x = <<abort,s>>
        ⎩ <n> ++ f_* w       si x = <n> ++ w
```

Continuidad es importante porque garantiza que f_* <n1,..,nk,...> = <n1,..,nk,...>.

```
[[catchin c0 with c1]]s = [[c1]]_♣ ([[c0]]s)
```

donde, dada una f ∈ Σ -> Ω, denotamos por f_♣ la siguiente extensión de f a Ω. Así, f_♣ ∈ Ω -> Ω definida por:

```
        ⎧ <n1,..,nk>              si x = <n1,..,nk>
f_♣ x = ⎨ <n1,..,nk,s>            si x = <n1,..,nk,s>
        | <n1,..,nk> ++ f s       si x = <n1,..,nk,<abort,s>>
        ⎩ <n1,..,nk,...>          si x = <n1,..,nk,...>
```

o sea

```
        ⎧ <n1,..,nk> ++ f s       si x = <n1,..,nk,<abort,s>>
f_♣ x = ⎨
        ⎩ x                       c.c.
```

También se puede definir así, f_♣ continua tal que:

```
        ⎧ <>                 si x = <>
f_♣ x = ⎨ <s>                si x = <s>
        | f s                si x = <<abort,s>>
        ⎩ <n> ++ f_♣ w       si x = <n> ++ w
```

```
[[while b do c]] = sup'(F^i ⊥')

              ⎧ w_* ([[c]]s)     si [[b]]s
donde F w s = ⎨
              ⎩ <s>              c.c.

[[newvar v:= e in c]]s = (λs'∈Σ. [s' | v : s v])_† ([[c]][s | v : [[e]]s])
```

donde, dada una f ∈ Σ -> Σ, denotamos por f_† la siguiente extensión de f a Ω. Así, f_† ∈ Ω -> Ω definida por:

```
        ⎧ <abort, f s>      si x = <abort, s>
f_† x = ⎨ f x               si x ∈ Σ
        ⎩ ⊥                 si x = ⊥

        ⎧ <n1,..,nk>                si x = <n1,..,nk>
f_† x = ⎨ <n1,..,nk,f s>            si x = <n1,..,nk,s>
        ⎪ <n1,..,nk,<abort,f s>>    si x = <n1,..,nk,<abort,s>>
        ⎩ <n1,..,nk,...>            si x = <n1,..,nk,...>
```

También se puede definir así, f_† continua tal que:

```
        ⎧ <>                si x = <>
f_† x = ⎨ <f s>             si x = <s>
        ⎪ <<abort,f s>>     si x = <<abort,s>>
        ⎩ <n> ++ f_† w      si x = <n> ++ w

[[!e]]s = <[[e]]s,s>
```

Ejercicio: reformular el teorema de coincidencia y el de sustitución para que tenga en cuenta los posibles nuevos comportamientos.

PRODUCTO CARTESIANO

1) Si P0, P1, ..., P{n-1} son órdenes parciales, entonces P0 x P1 x ... x P{n-1} también lo es, donde el orden entre tuplas se define componente a componente.
2) Una cadena de tuplas es una secuencia <p10, p11, ..., p1{n-1}> ≤ <p20, p21, ..., p2{n-1}> ≤ ... ≤ <pm0, pm1, ..., pm{n-1}> ≤ ... tal que si tomamos componente a componente p10 ≤_0 p20 ≤_0 ... ≤_0 pm0 ≤_0 ... es una cadena de P10, p11 ≤_1 p21 ≤_1 ... ≤_1 pm1 ≤_1 ... es una cadena de P1, ..., p1{n-1} ≤_{n-1} p2{n-1} ≤_{n-1} ... ≤_{n-1} pm{n-1} ≤_{n-1} ... es una cadena de P{n-1},
3) Si P0, P1, ..., P{n-1} son predominios, entonces P0 x P1 x ... x P{n-1} también lo es, donde el supremo de una cadena de tuplas se define componente a componente.
4) Si P0, P1, ..., P{n-1} son dominios, entonces P0 x P1 x ... x P{n-1} también lo es, donde el mínimo es la tupla que consiste del mínimo de cada uno de los dominios.

Todas las funciones sencillas usuales (proyecciones, constructor de tuplas, etc) son trivialmente continuas.

UNIONES DISJUNTAS

Dados conjuntos P0, P1, ..., P{n-1} se define la unión disjunta
  P0 + P1 + ... + P{n-1} = { <i,p> | p ∈ Pi}.
Se definen las inyecciones Li ∈ Pi -> P0 + P1 + ... + P{n-1} por
  Li p = <i,p>

1) Si P0, P1, ..., P{n-1} son órdenes parciales, entonces P0 + P1 + ... + P{n-1} también lo es, donde el orden "no mezcla" los órdenes de los diferentes conjuntos dados:
  <i,p> ≤ <j,q> sii i = j ∧ p ≤_i q
2) Una cadena de P0 + P1 + ... + P{n-1} es una que proviene enteramente de uno de los Pi: <i,p1> ≤ <i,p2> ≤ ... ≤ <i,pn> ≤ ... donde los pj provienen todos de Pi.
3) Si P0, P1, ..., P{n-1} son órdenes parciales, entonces P0 + P1 + ... + P{n-1} también lo es, donde el supremo de una cadena es simplemente el supremo de la componente de donde la cadena proviene:
  sup(<i,pj>) = <i,sup_i(pj)>
4) Si P0, P1, ..., P{n-1} son dominios, entonces P0 + P1 + ... + P{n-1} en general no lo es (sólo lo es en el caso trivial n = 1). ¿Cuál sería el mínimo, si los mínimos de los diferentes Pi no se comparan entre sí?

Todas las funciones sencillas usuales (inyecciones, análisis por casos, etc) son trivialmente continuas.