GADTs meet their Match

Tom Schrijvers

KU LEUVEN

with Georgios Karachalias, Dimitrios Vytiniotis
and Simon Peyton Jones

Checking Pattern Matches

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

```
*Main> zip [] [1]
*** Exception:
      Non-exhaustive patterns in function zip
```

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

```
*Main> zip [] [1]
*** Exception:
      Non-exhaustive patterns in function zip
```

Runtime crash

😢

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

```
*Main> :l zip.hs
[1 of 1] Compiling Main
Warning:
    Pattern match(es) are non-exhaustive
    In an equation for ‘zip’:
        Patterns not matched:
            [] (_ : _)
            (_ : _) []
```

Property 1:
Exhaustiveness

```
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

```
*Main> :l zip.hs
[1 of 1] Compiling Main
Warning:
    Pattern match(es) are non-exhaustive
    In an equation for ‘zip’:
        Patterns not matched:
            [] (_ : _)
            (_ : _) []
```

Compiler warning

😀

GADT Problem 1

```
data Vect :: Nat -> * -> * where
  VN :: Vect Zero a
  VC :: a -> Vect n a -> Vect (Succ n) a
```

```
vzip :: Vect n a -> Vect n b -> Vect n (a,b)
vzip VN         VN         = VN
vzip (VC x xs) (VC y ys) = VC (x,y) (vzip xs ys)
```

GADT Problem I

```
data Vect :: Nat -> * -> * where
  VN :: Vect Zero a
  VC :: a -> Vect n a -> Vect (Succ n) a
```

```
vzip :: Vect n a -> Vect n b -> Vect n (a,b)
vzip VN         VN         = VN
vzip (VC x xs) (VC y ys) = VC (x,y) (vzip xs ys)
```

```
Pattern match(es) are non-exhaustive
 In an equation for ‘vzip’:
     Patterns not matched:
         VN (VC _ _)
         (VC _ _)VN
```

GADT Problem I

```
data Vect :: Nat -> * -> * where
  VN :: Vect Zero a
  VC :: a -> Vect n a -> Vect (Succ n) a
```

```
vzip :: Vect n a -> Vect n b -> Vect n (a,b)
vzip VN         VN         = VN
vzip (VC x xs) (VC y ys) = VC (x,y) (vzip xs ys)
```

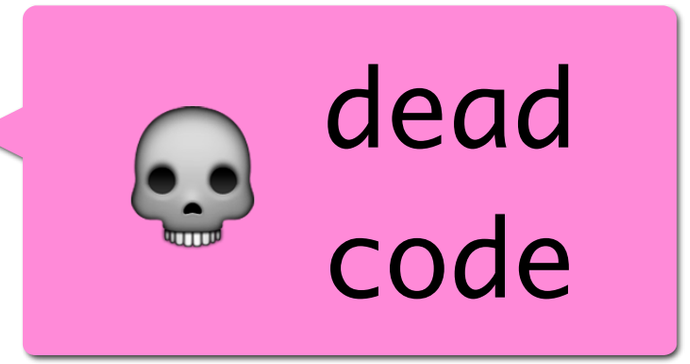
```
Pattern match(es) are non-exhaustive
  In an equation for ‘vzip’:
      Patterns not matched:
          VN (VC _ _)
          (VC _ _)VN
```

Property 2:
Redundancy

```
len []       =  0
len (x:xs)   =  1 + len xs
len l        =  42
```

Property 2:
Redundancy

```
len []       =  0
len (x:xs)   =  1 + len xs
len l        =  42
```



Property 2:
Redundancy

```
len []       =  0
len (x:xs)   =  1 + len xs
len l        =  42
```


```
Warning:
    Pattern match(es) are overlapped
    In an equation for ‘len’: len l = ...
```

Property 2:
Redundancy

```
len []       =  0
len (x:xs)   =  1 + len xs
len l        =  42
```

dead code

```
Warning:
    Pattern match(es) are overlapped
    In an equation for ‘len’: len l = ...
```

Compiler warning

GADT Problem 2

```
vzip VN VN                    =  VN
vzip (VC x xs) (VC y ys)      =  VC (x,y) (vzip xs ys)
vzip _          _             =  error “unreachable”
```

GADT Problem 2

```
vzip VN VN                       =  VN
vzip (VC x xs) (VC y ys)         =  VC (x,y) (vzip xs ys)
vzip _          _                =  error “unreachable”
```

suppresses bogus non-exaustiveness warnings

GADT Problem 2

```
vzip VN VN                      =  VN
vzip (VC x xs) (VC y ys)        =  VC (x,y) (vzip xs ys)
vzip _          _               =  error “unreachable”
```

suppresses bogus non-exaustiveness warnings

dead code

GADT Problem 2

```
vzip VN VN                      =  VN
vzip (VC x xs) (VC y ys)        =  VC (x,y) (vzip xs ys)
vzip _          _               =  error “unreachable”
```

suppresses bogus non-exaustiveness warnings

dead code

No compiler warning

Quick Summary

Exhaustiveness

Redundancy

} broken for GADTs

Quick Summary

Exhaustiveness
Redundancy
} broken for GADTs

But there is more …

Challenge 1

```
data F a where
  F1 :: F Int
  F2 :: F Bool

data G a where
  G1 :: G Int
  G2 :: G Char
```

```
h :: F a -> G a -> Int
h F1 G1 = 1
h _  _  = 2
```

Challenge 1

```
data F a where
  F1 :: F Int
  F2 :: F Bool

data G a where
  G1 :: G Int
  G2 :: G Char
```

```
h :: F a -> G a -> Int
h F1 G1 = 1
h _  _  = 2
```

redundant?

Challenge 1

```
data F a where
  F1 :: F Int
  F2 :: F Bool
```

```
data G a where
  G1 :: G Int
  G2 :: G Char
```

```
h :: F a -> G a -> Int
h F1 G1 = 1
h _  _  = 2
```

redundant?

```
> h F2 undefined
2
```

Challenge 1

```
data F a where
  F1 :: F Int
  F2 :: F Bool
```

```
data G a where
  G1 :: G Int
  G2 :: G Char
```

```
h :: F a -> G a -> Int
h F1 G1 = 1
h _  _  = 2
```

redundant?

```
> h F2 undefined
2
```

Challenge 1

```
data F a where
  F1 :: F Int
  F2 :: F Bool

data G a where
  G1 :: G Int
  G2 :: G Char
```

```
h :: F a -> G a -> Int
h F1 G1 = 1
h _  _  = 2
```

redundant?

```
> h F2 undefined
2
```

```
> h F2 undefined
*** Exception
```

Challenge 2

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

Challenge 2

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

redundant?

Challenge 2

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

redundant?

```
> g undefined True
*** Exception
```

Challenge 2

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

redundant?

```
> g undefined True
*** Exception
```

```
> g undefined True
3
```

Challenge 2

`useful' pattern

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

```
> g undefined True
*** Exception
```

```
> g undefined True
3
```

Challenge 2

```
g :: Bool -> Bool -> Int
g _     False = 1
g True  False = 2
g _     _     = 3
```

`useful' pattern

unreachable RHS

```
> g undefined True
*** Exception
```

```
> g undefined True
3
```

The Guard Challenge

```
abs2 :: Int -> Int
abs2 x | x < 0  = -x
       | x >= 0 = x
```

exhaustive?

The Guard Challenge

```
abs2 :: Int -> Int
abs2 x | x < 0  = -x
       | x >= 0 = x
```

exhaustive?

```
append xs ys
  | []     <- xs = ys
  | (p:ps) <- xs = p : append ps ys
```

exhaustive?

What we need:

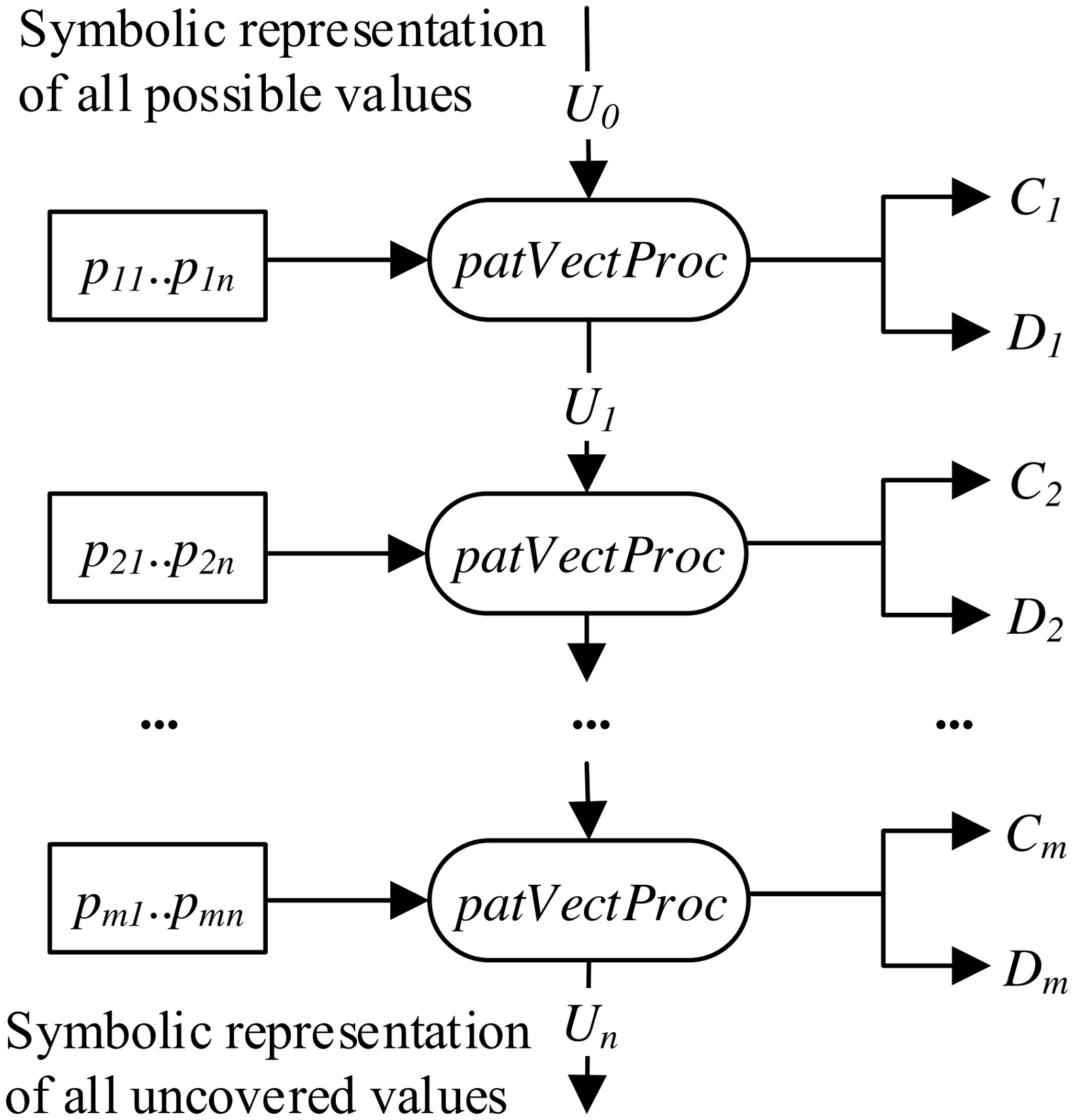
A uniform approach to deal with:

GADTs

Laziness

Guards

Our Approach


Symbolic representation
of all possible values
U_0
$p_{11}..p_{1n}$
patVectProc
C_1
D_1
U_1
$p_{21}..p_{2n}$
patVectProc
C_2
D_2
...
...
...
$p_{m1}..p_{mn}$
patVectProc
C_m
D_m
U_n
Symbolic representation
of all uncovered values

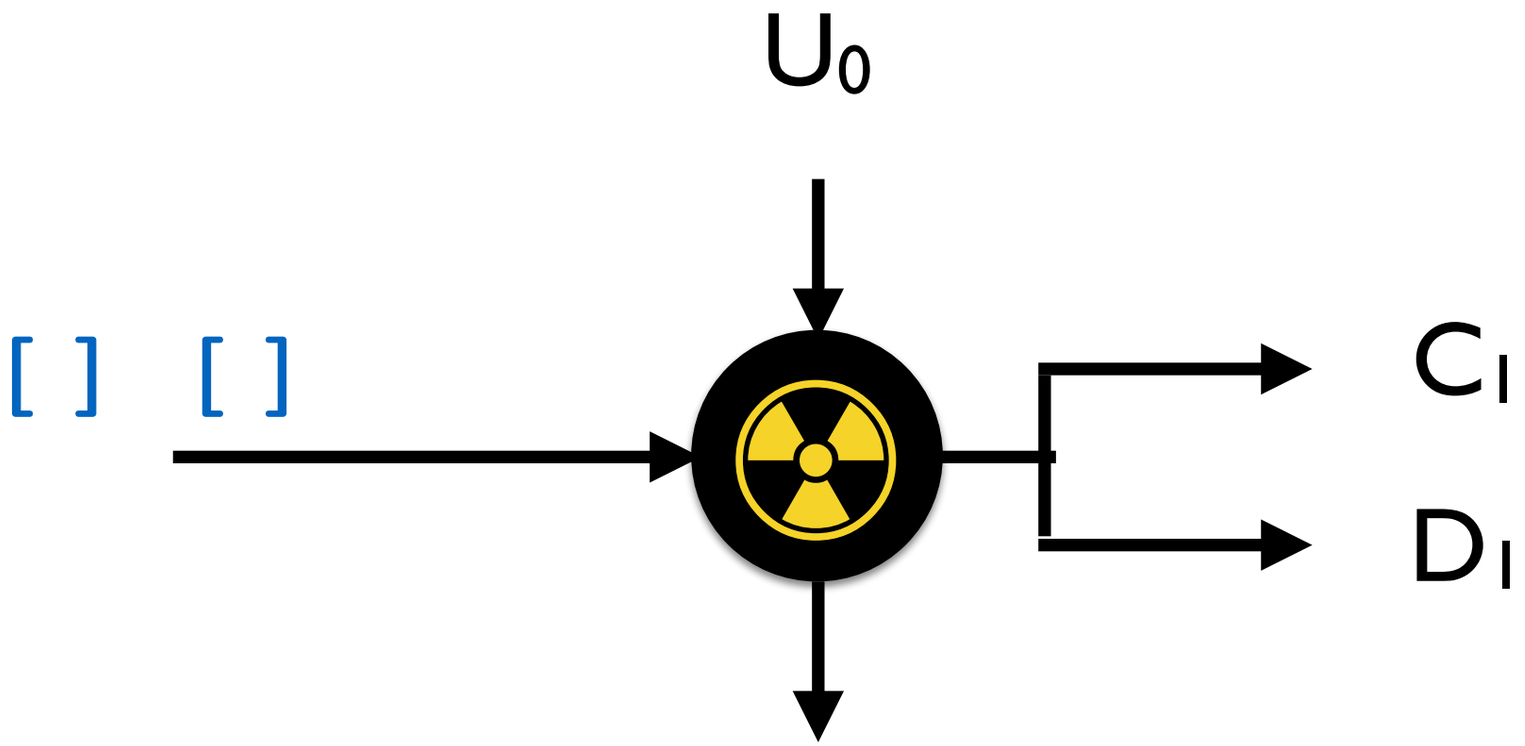
Basic ADT Support

```
zip [] [] = …
zip (x:xs) (y:ys) = …
```

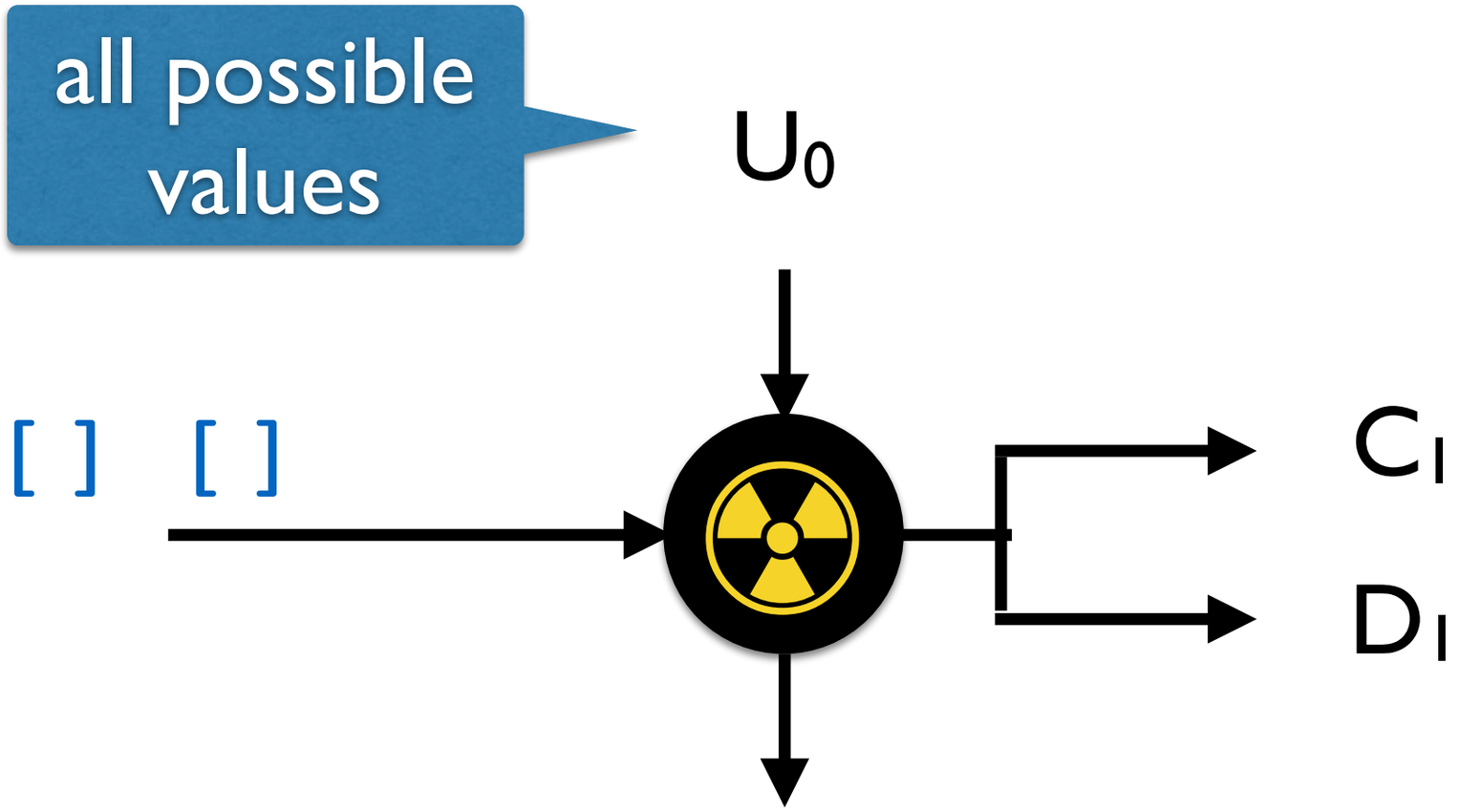
```
zip [] [] = …
zip (x:xs) (y:ys) = …
```

```
[ ]   [ ]
```

```
(x:xs) (y:ys)
```

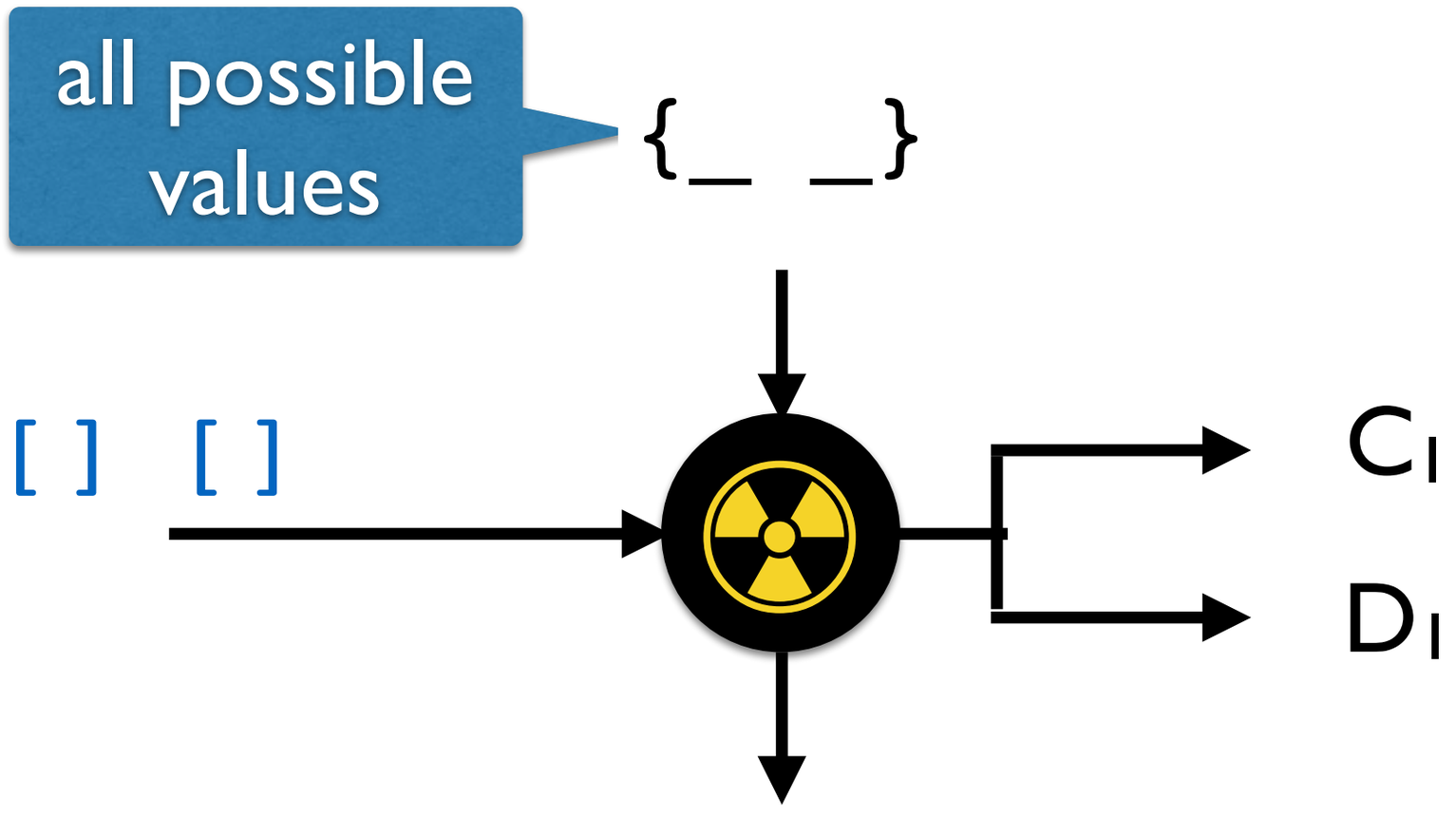



```
(x:xs) (y:ys)
```

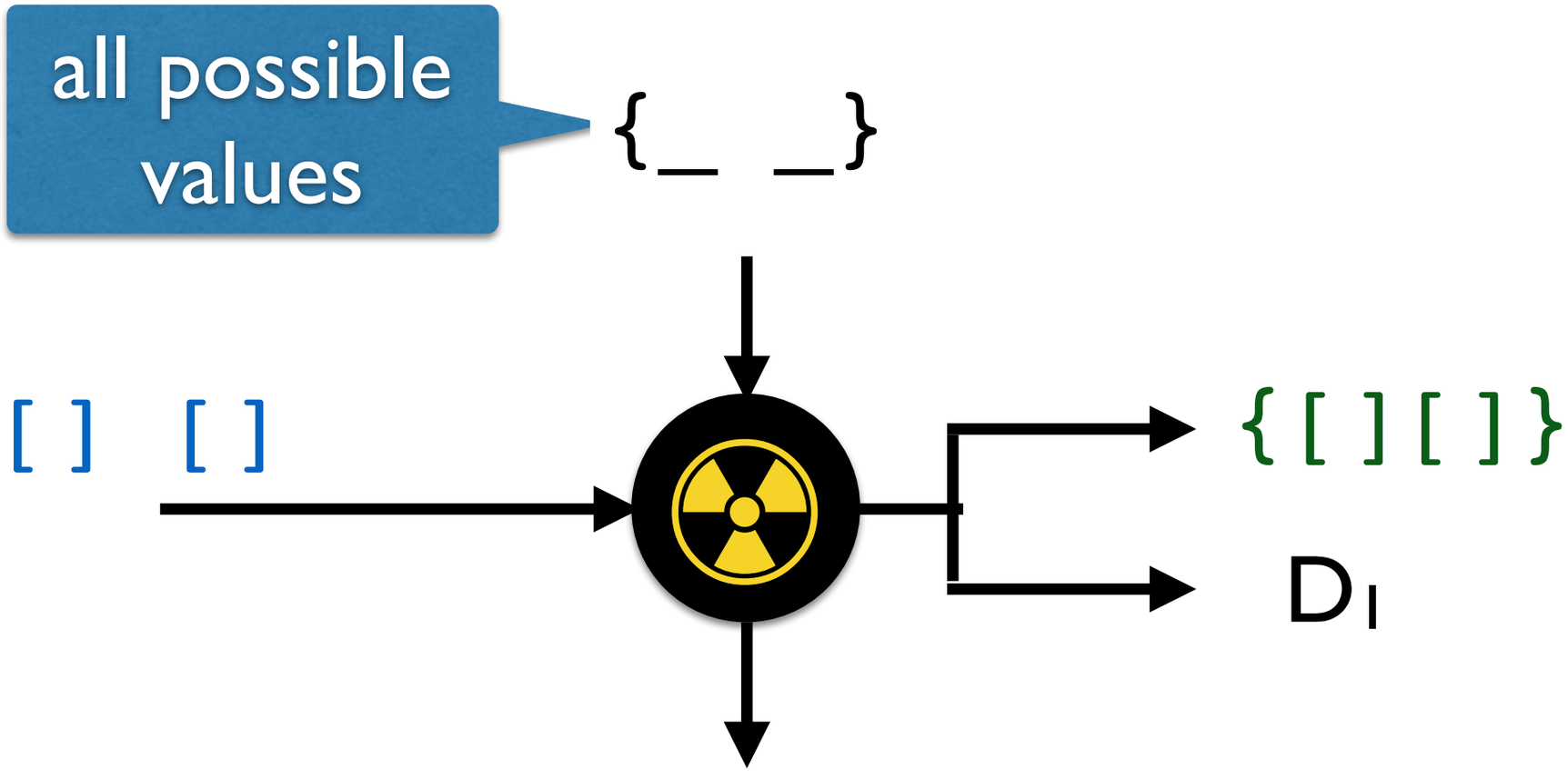



```
(x:xs) (y:ys)
```

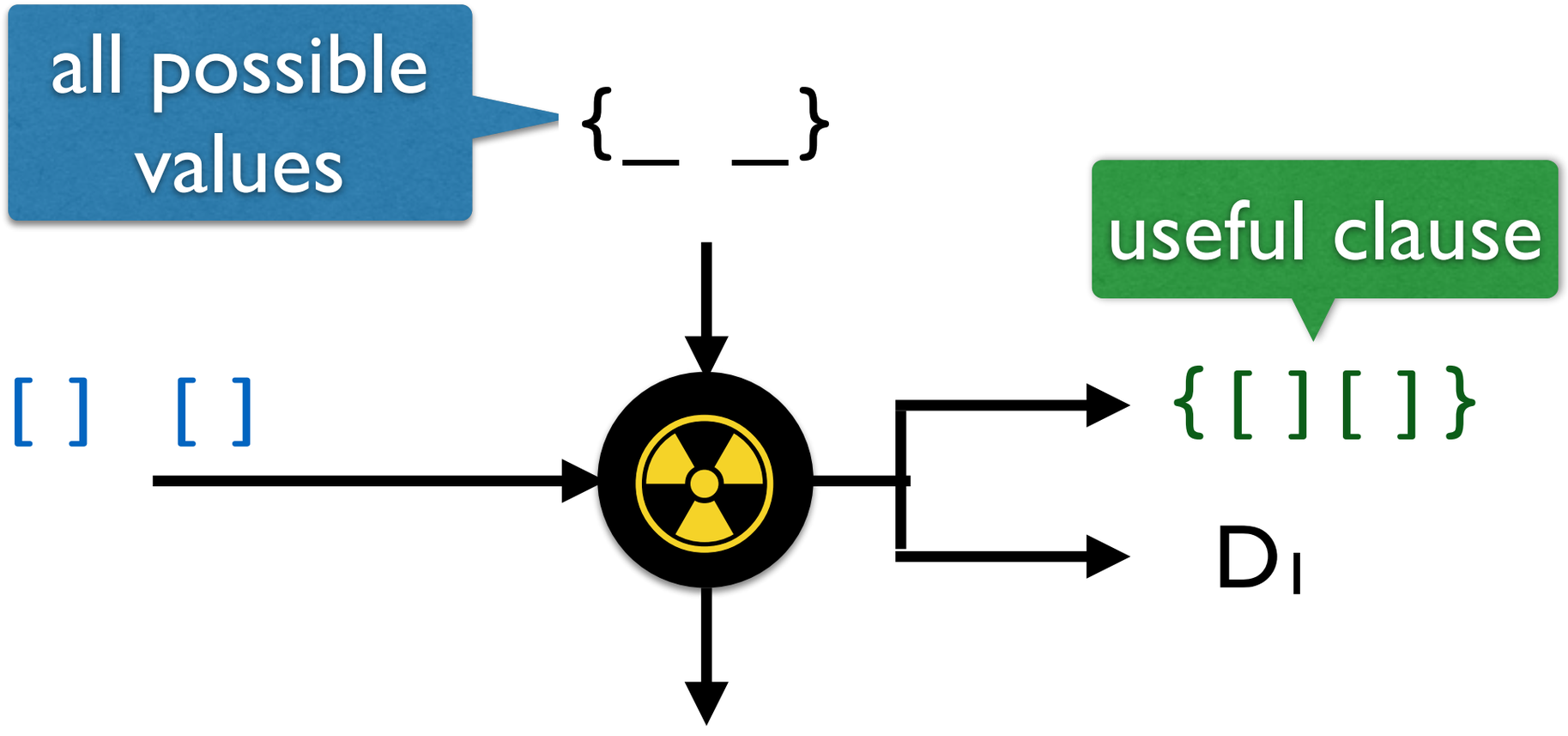
all possible values



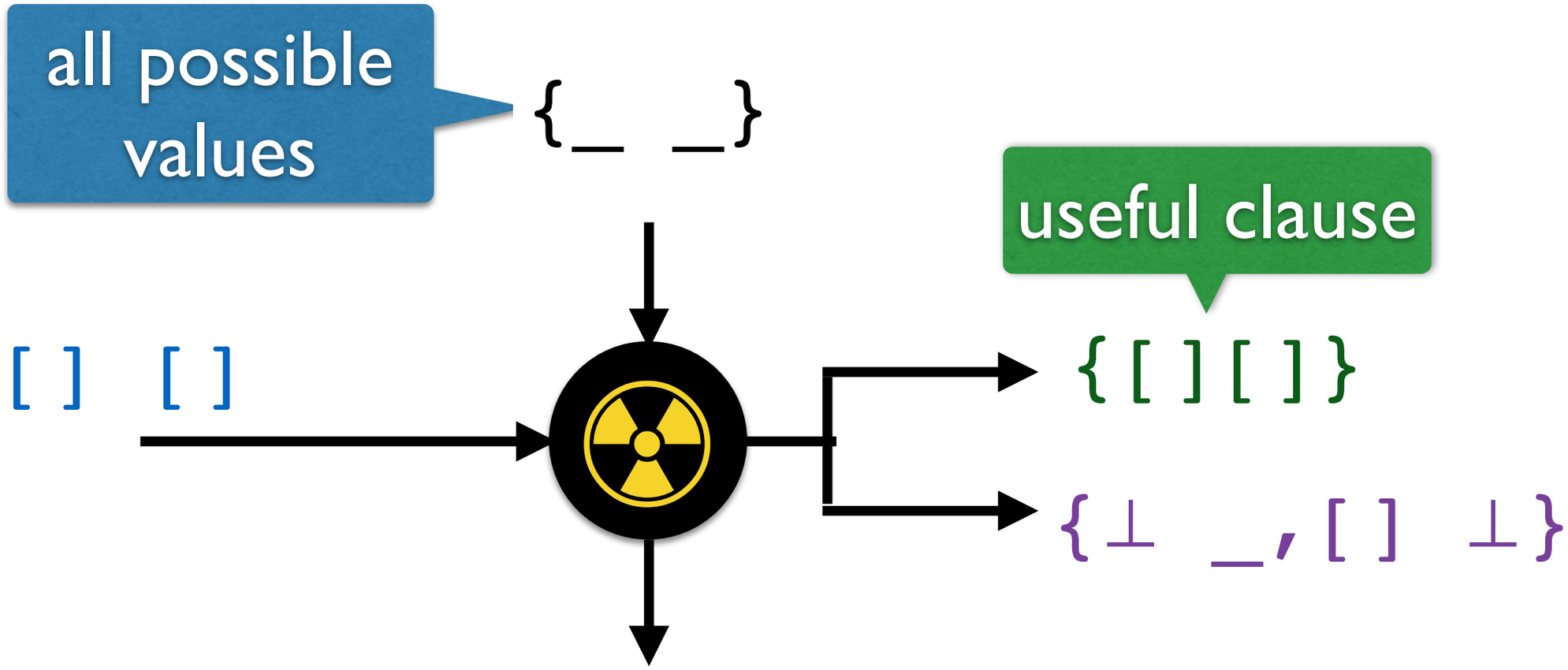

```
(x:xs) (y:ys)
```




```
(x:xs) (y:ys)
```

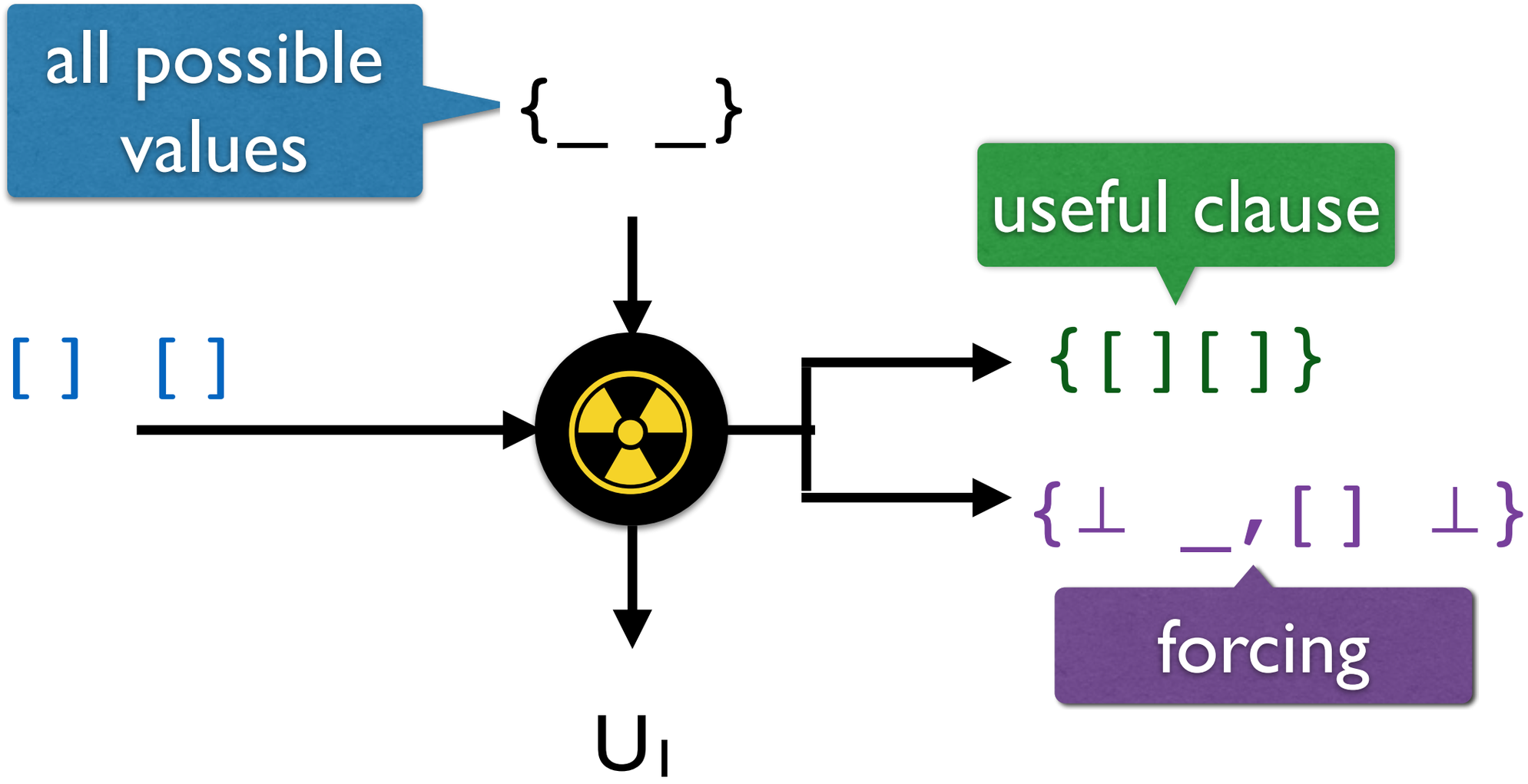



```
(x:xs) (y:ys)
```



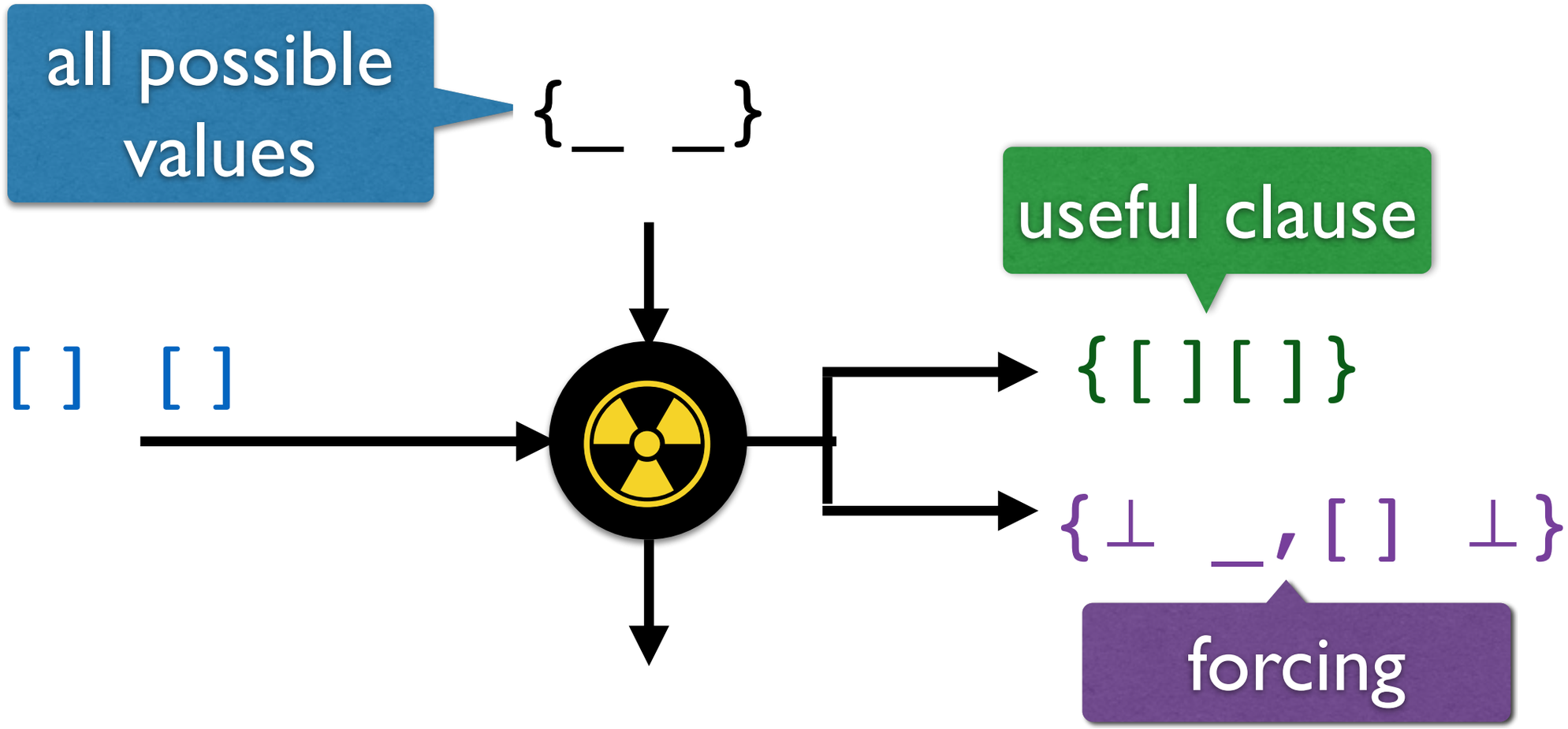
U_1

```
(x:xs) (y:ys)
```



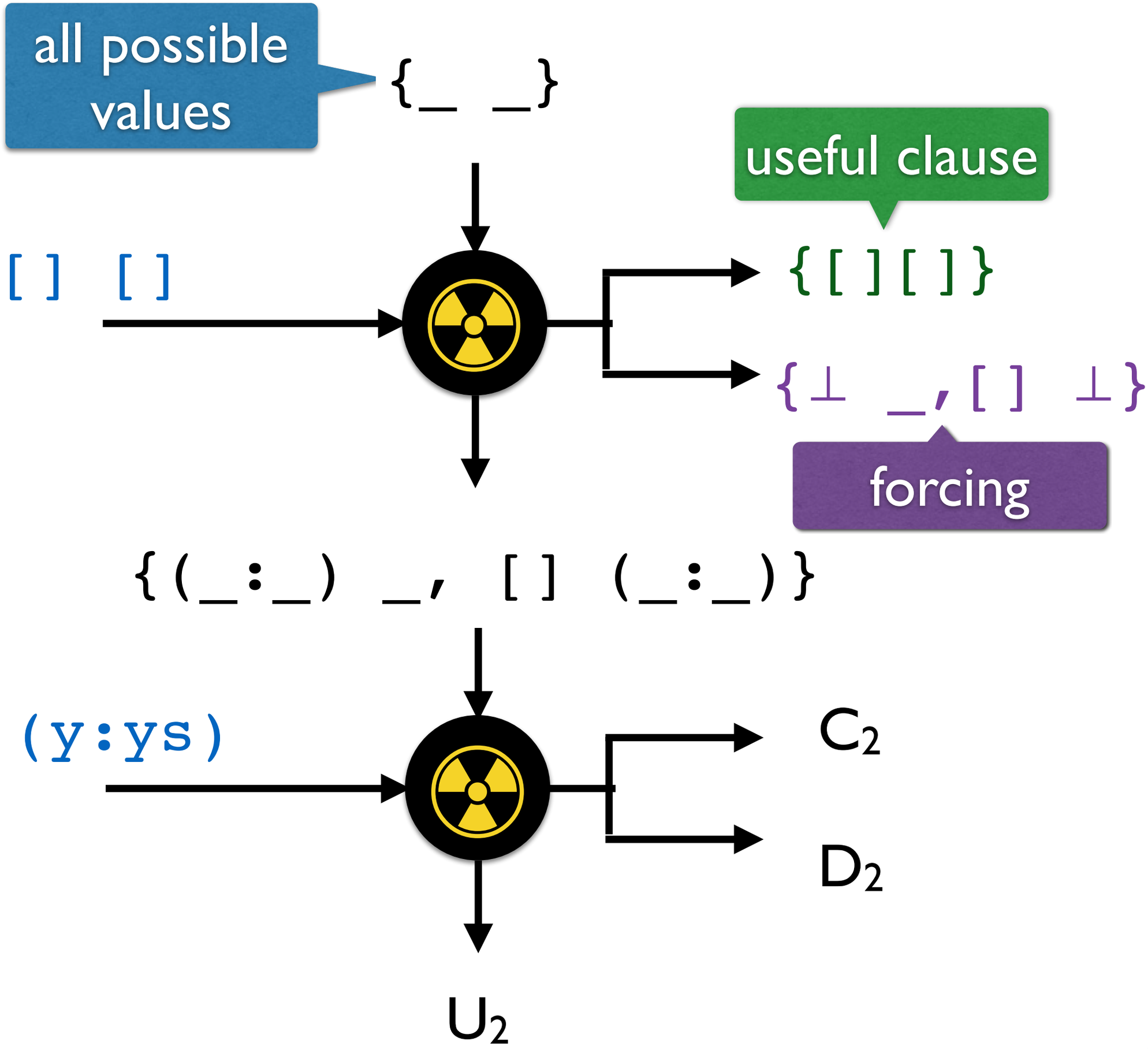
U_I

```
(x:xs) (y:ys)
```




```
{(_:_) _, [] (_:_)}
```

```
(x:xs) (y:ys)
```

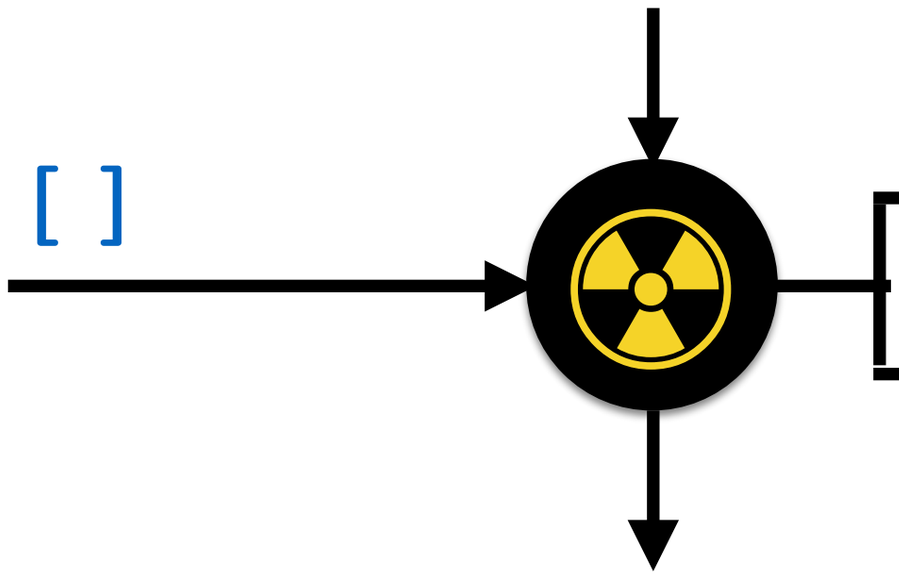

all possible values
{_ _}
useful clause
[ ] [ ]
{[ ][ ]}
{⊥ _,[ ] ⊥}
forcing
{(_:_) _, [] (_:_)}
(x:xs) (y:ys)
C2
D2
U2

all possible values

{_ _}

useful clause

[ ] [ ]



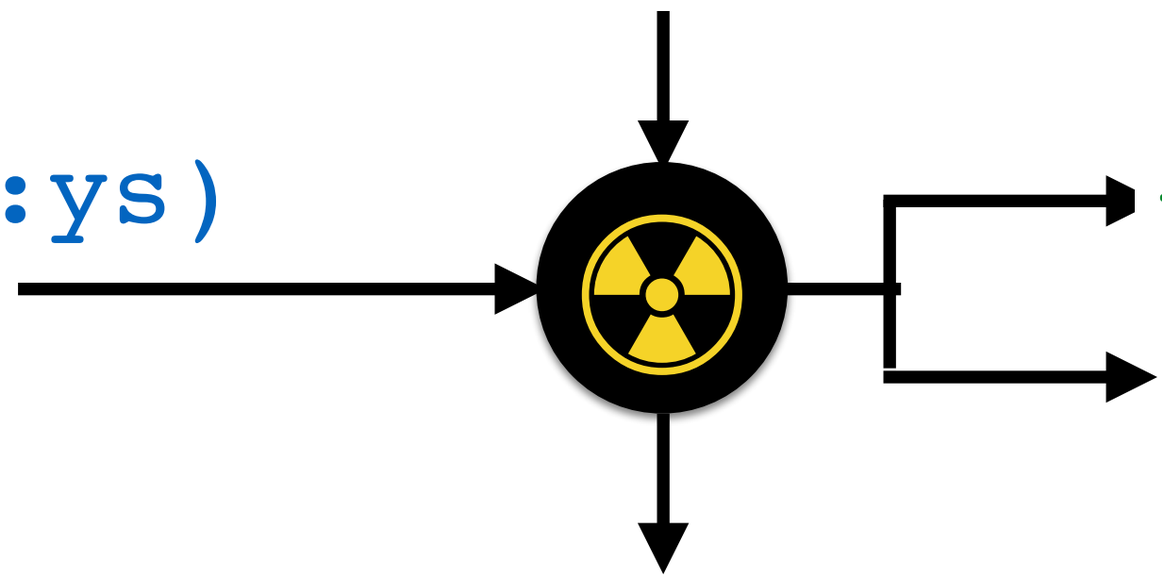
{[ ][ ]}

{⊥ _,[ ] ⊥}

forcing

{(_:_) _, [] (_:_)}

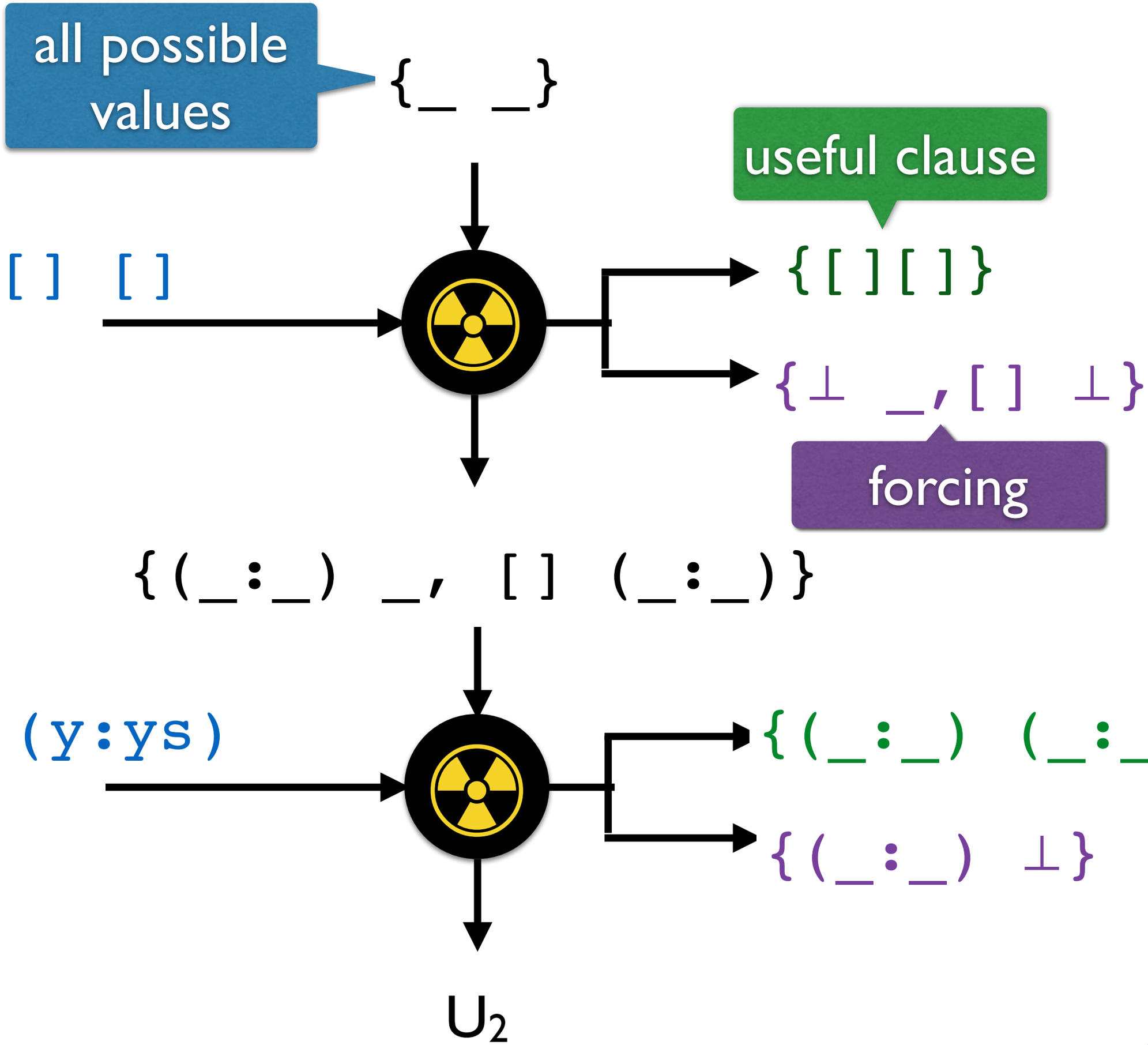
(x:xs) (y:ys)



{(_:_) (_:_)}

D_2

U_2

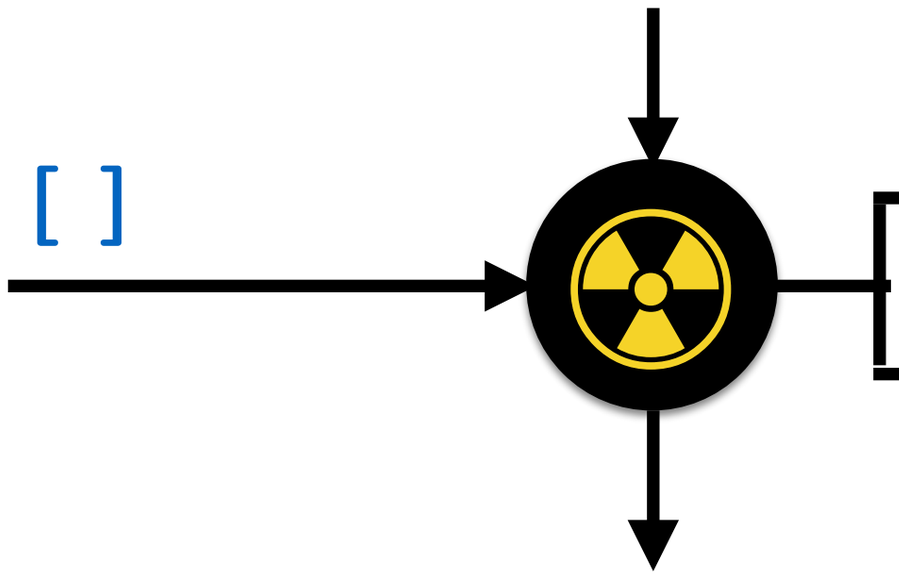

all possible values
{_ _}
useful clause
[ ] [ ]
{[ ][ ]}
{⊥ _,[ ] ⊥}
forcing
{(_:_) _, [] (_:_)}
(x:xs) (y:ys)
{(_:_) (_:_)}
{(_:_) ⊥}
U2

all possible values

{_ _}

useful clause

[ ] [ ]



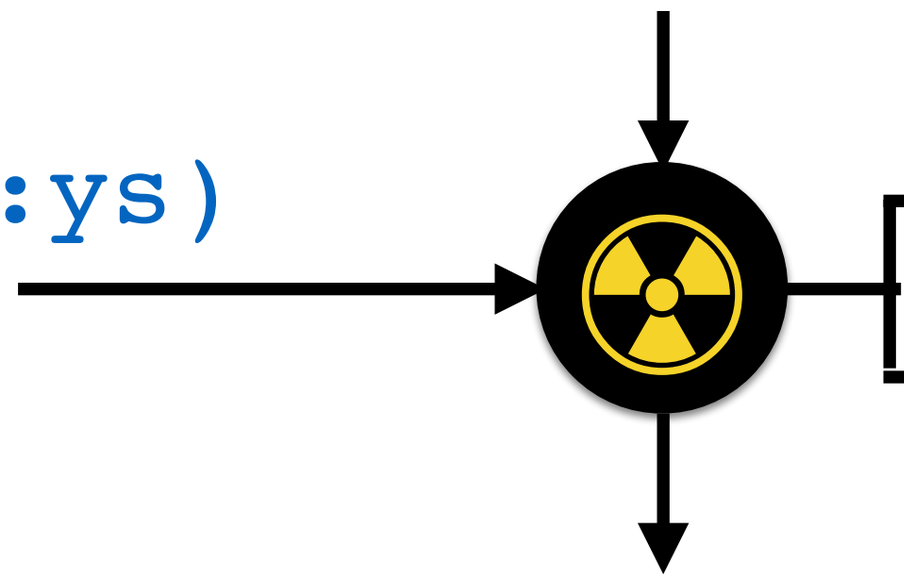
{[ ][ ]}

{⊥ _,[ ] ⊥}

forcing

{(_:_) _, [] (_:_)}

(x:xs) (y:ys)



{(_:_) (_:_)}

{(_:_) ⊥}

{(_:_) [], [] (_:_)}

all possible values

{_ _}

useful clause

[ ] [ ]

{[ ][ ]}

{⊥ _,[ ] ⊥}

forcing

{(_:_) _, [] (_:_)}

(x:xs) (y:ys)

{(_:_) (_:_)}

{(_:_) ⊥}

not exhaustive

{(_:_) [], [] (_:_)}

Basic Syntax

Pattern

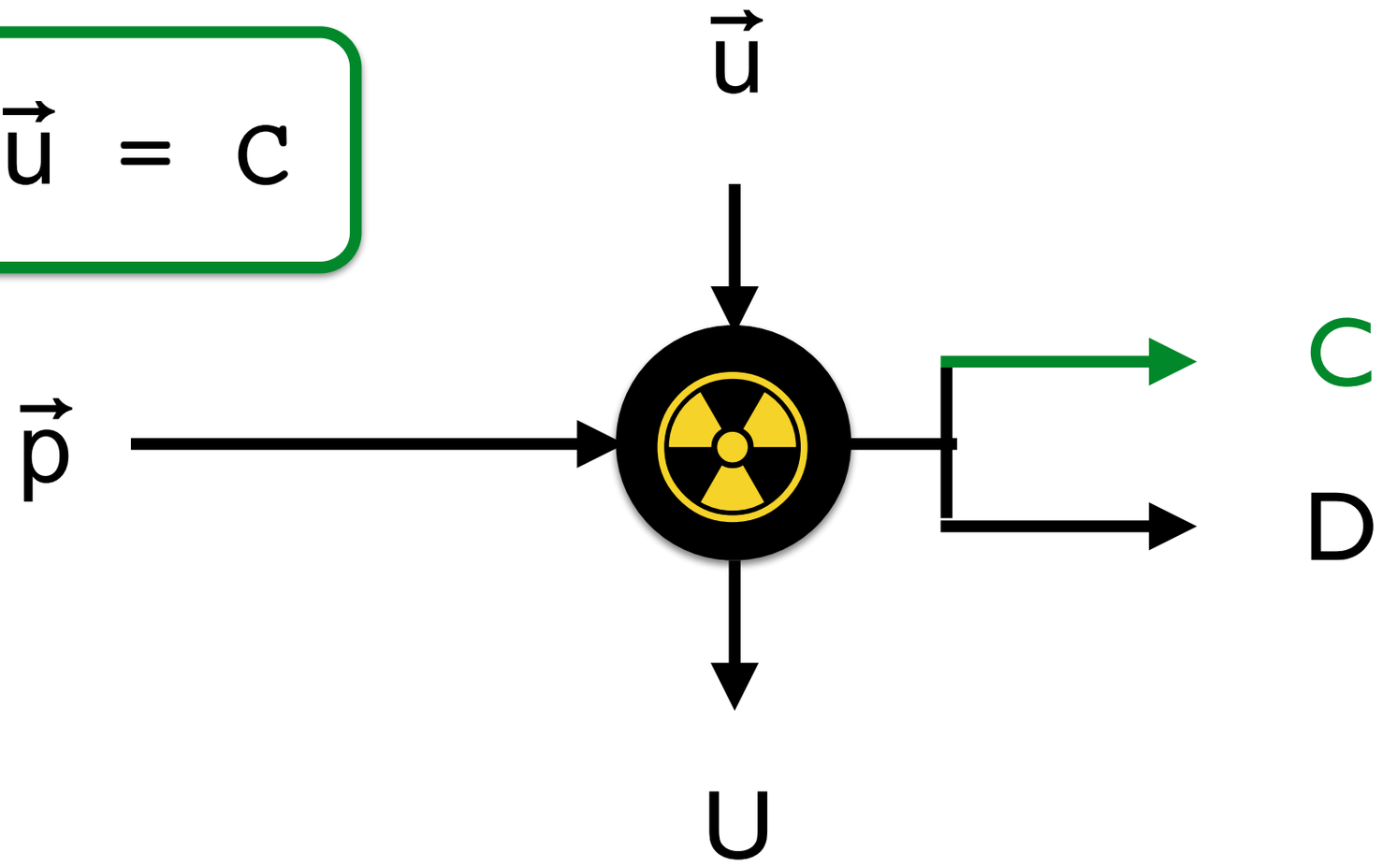
$$p ::= \_ \mid \kappa\ \vec{p}$$

Value Abstraction

$$u ::= \overline{\_} \mid \kappa\ \vec{u}$$

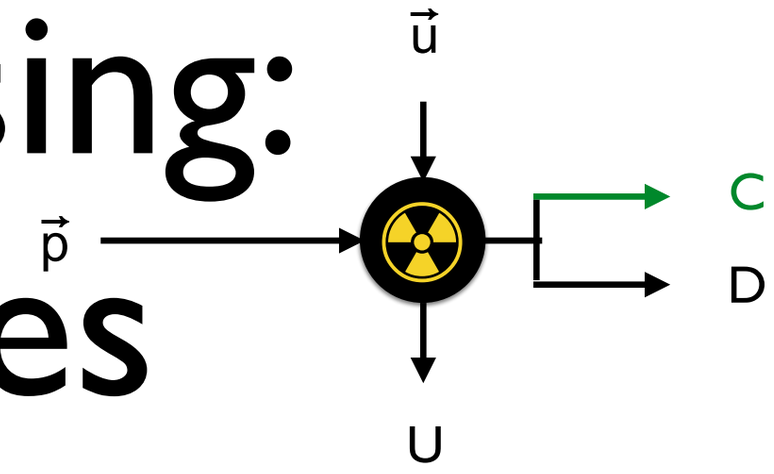
Clause Processing: Covered Values

$C\ \vec{p}\ \vec{u} = c$



Clause Processing: Covered Values

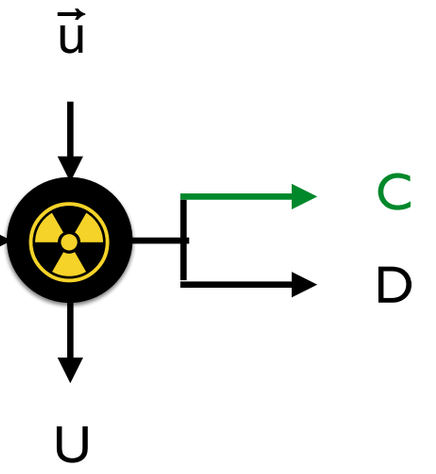
$C\ \vec{p}\ \vec{u} = c$



$C\ \varepsilon \qquad \varepsilon \quad = \{\varepsilon\}$

Clause Processing: Covered Values

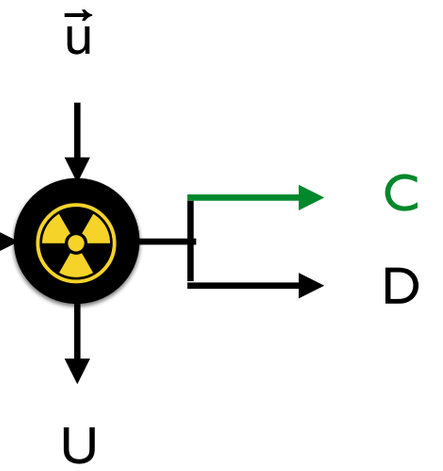
$C\ \vec{p}\ \vec{u}\ =\ c$




```
C ε       ε     = {ε}
C (_ p) (u u) = {u w | w <- C p u }
```

Clause Processing: Covered Values

$C\ \vec{p}\ \vec{u}\ =\ c$



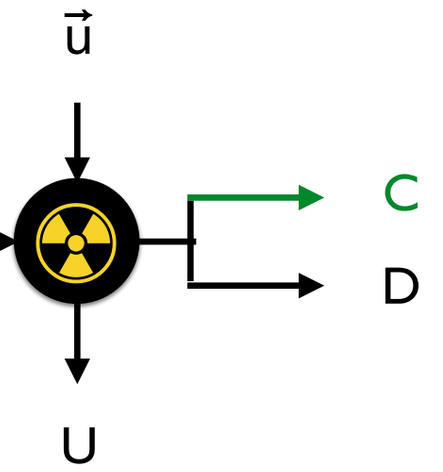

```
C ε       ε     = {ε}

C (_ p) (u u) = {u w | w <- C p u }

C ((Ki q) p) ((Kj w) u)
  | Ki == Kj = {(Ki w') u'
                |w' u' <- C (q p) (w u)}
  | otherwise = {}
```

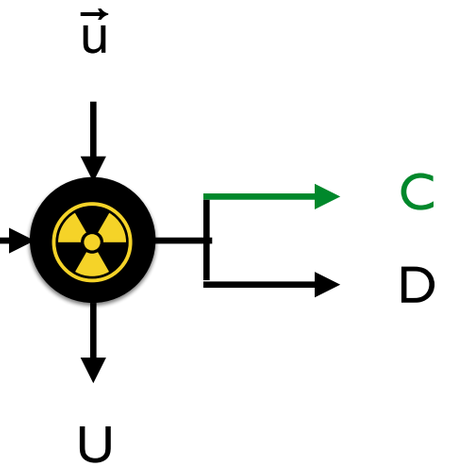
Clause Processing: Covered Values

$C\ \vec{p}\ \vec{u} = c$




```
C ε       ε     = {ε}
C (_ p) (u u) = {u w | w <- C p u }
C ((Ki q) p) ((Kj w) u)
  | Ki == Kj = {(Ki w') u'
                |w' u' <- C (q p) (w u)}
  | otherwise = {}
C ((Ki q) p) (_ u)
  = C ((Ki q) p) ((Ki _…_) u)
```

Clause Processing: Covered Values



$C\ \vec{p}\ \vec{u}\ =\ c$

Uncovered

$\mathcal{U}\ \vec{p}\ \vec{u}\ =\ u$

Diverging

$\mathcal{D}\ \vec{p}\ \vec{u}\ =\ D$

```
= {ε}

= {u w⃗ | w⃗ <- C p⃗ u⃗ }

(Kⱼ w⃗) u⃗)

{(Kᵢ w⃗′) u⃗′
|w⃗′ u⃗′ <- C (q⃗ p⃗) (w⃗ u⃗)}

= {}
```

$C\ ((K_i\ \vec{q})\ \vec{p})\ (\_\ \vec{u})$

$= C\ ((K_i\ \vec{q})\ \vec{p})\ ((K_i\ \_\ldots\_)\ \vec{u})$

GADTs

GADTs and Type Constraints

```
data Vec :: Nat -> * -> * where
  VN ::                    Vec Z a
  VC :: a -> Vec m a -> Vec (S m) a
```

GADTs and Type Constraints

```
data Vec :: Nat -> * -> * where
  VN ::                    Vec Z a
  VC :: a -> Vec m a -> Vec (S m) a
```

equivalent

```
data Vec :: Nat -> * -> * where
  VN :: n ~ Z    =>                  Vec n a
  VC :: n ~ S m => a -> Vec m a -> Vec n a
```

GADTs and Type Constraints

```
data Vec :: Nat -> * -> * where
  VN ::                    Vec Z a
  VC :: a -> Vec m a -> Vec (S m) a
```

equivalent

```
data Vec :: Nat -> * -> * where
  VN :: n ~ Z     =>                    Vec n a
  VC :: n ~ S m   => a -> Vec m a -> Vec n a
```

type constraints

Syntax with Type Constraints

Value Abstraction

$$v ::= \Gamma \vdash \vec{u} \triangleright \Delta$$

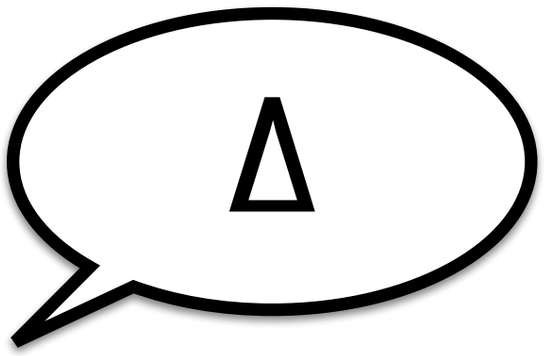
$$u ::= \_ \mid K\ \vec{u}$$

refined by type constraints

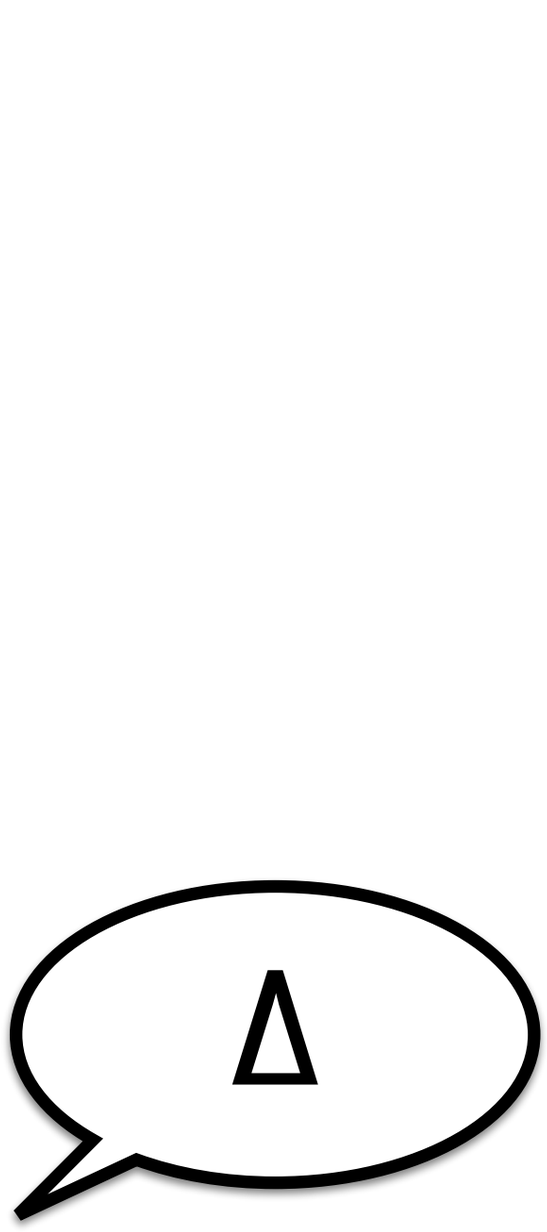
$$\Gamma ::= \varepsilon \mid \Gamma, a$$

$$\Delta ::= \varepsilon \mid \Delta \cup \Delta \mid \tau \sim \tau$$

The Oracle

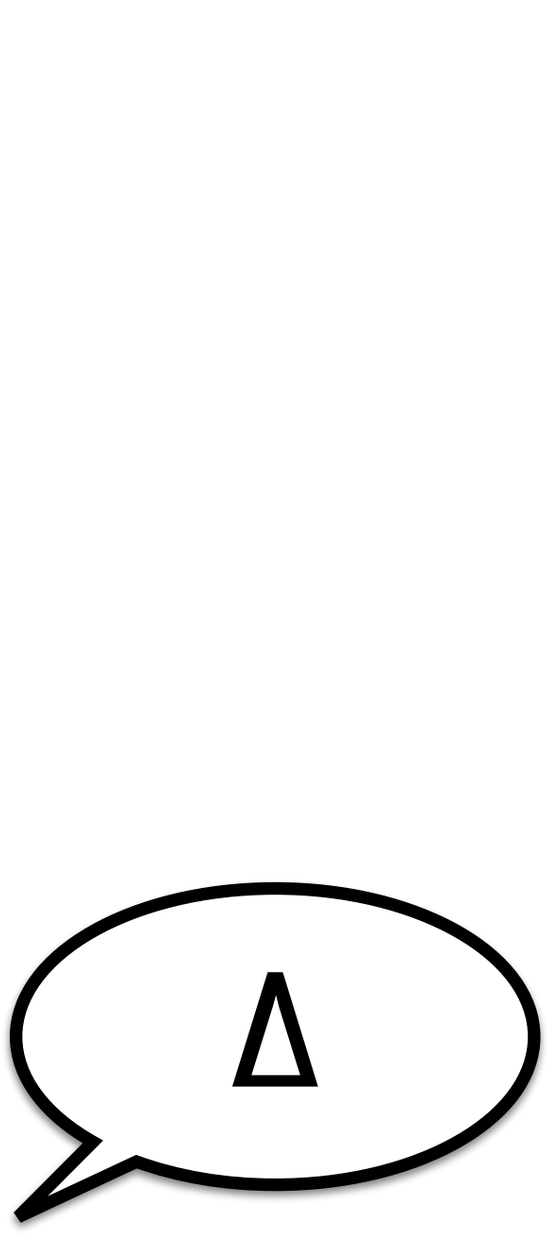


The Oracle



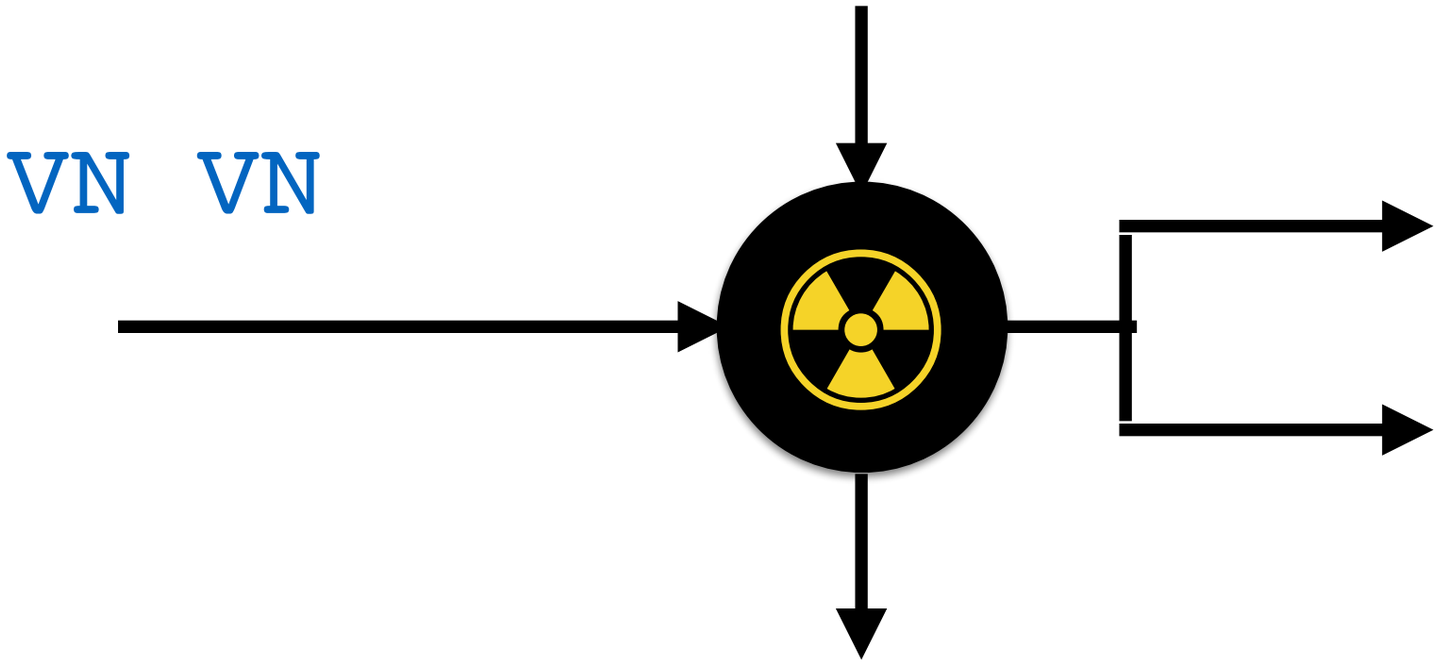
Not Satisfiable

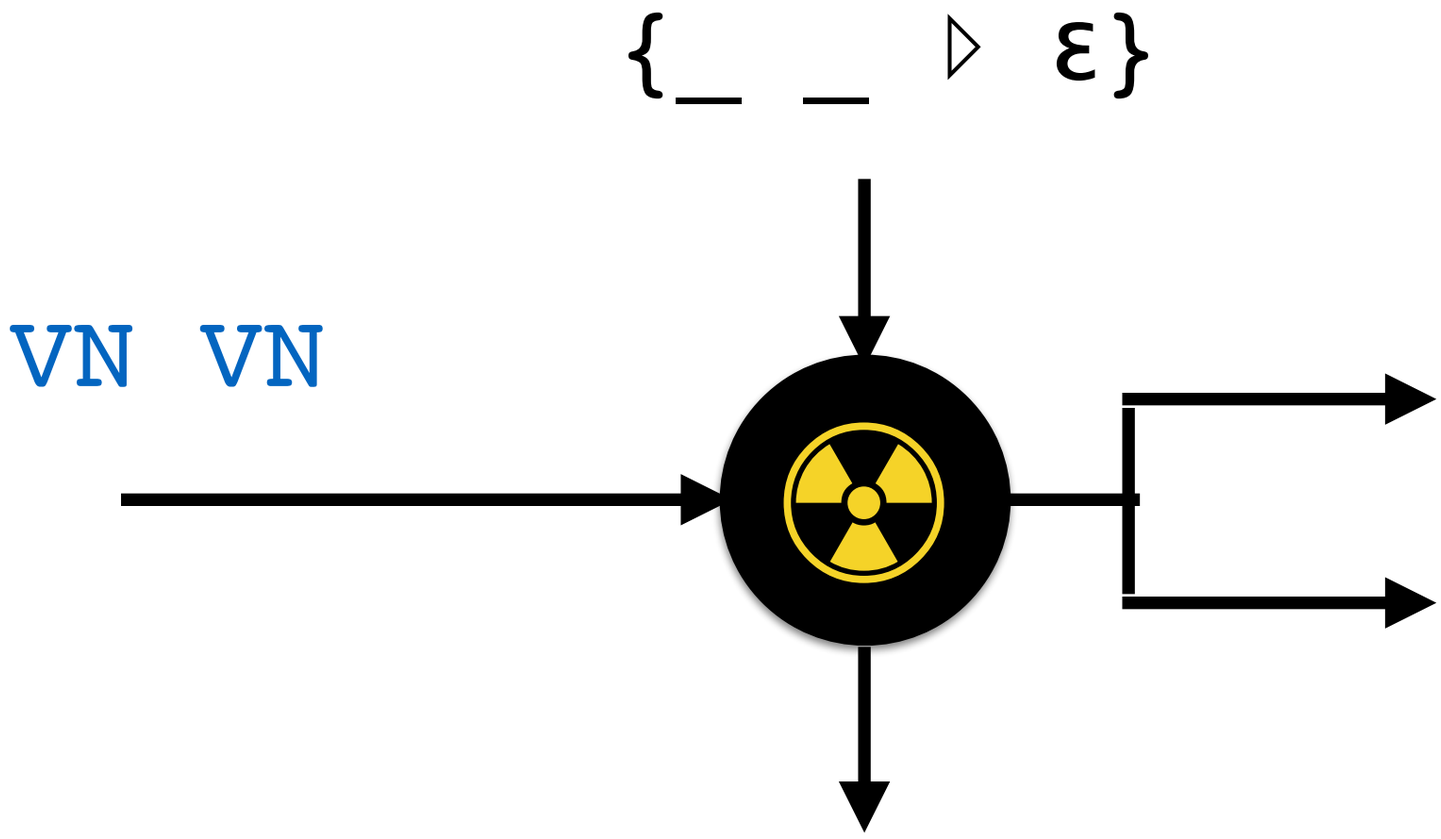
The Oracle



Not Satisfiable

Maybe Satisfiable

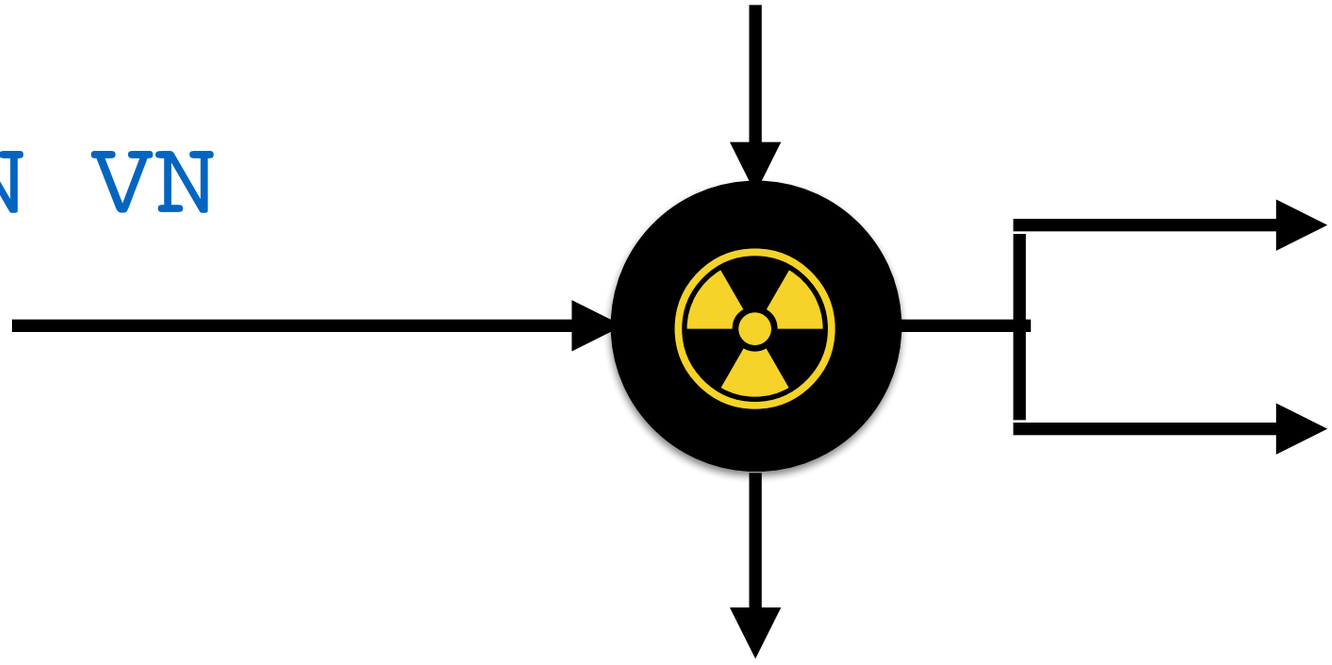

VN VN


{_ _ ▷ ε}
VN VN

unconstrained

{_ _ ▷ ε}

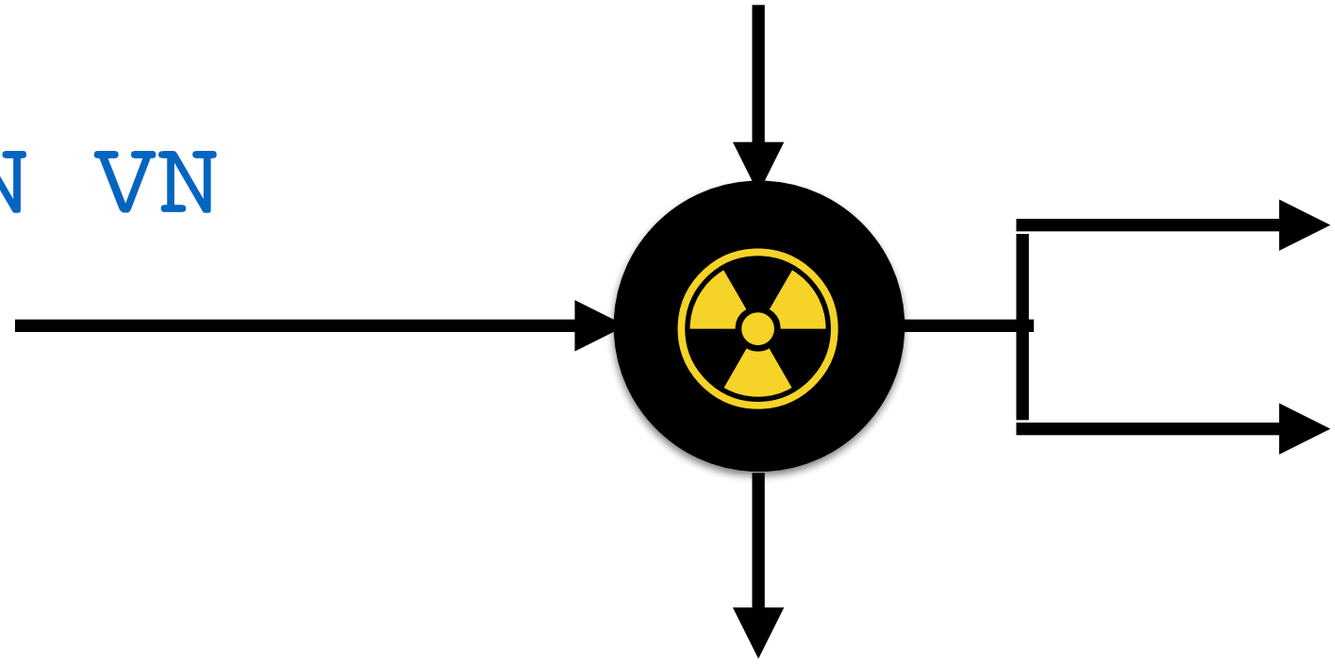
VN VN



unconstrained

{_ _ ▷ ε}

VN VN

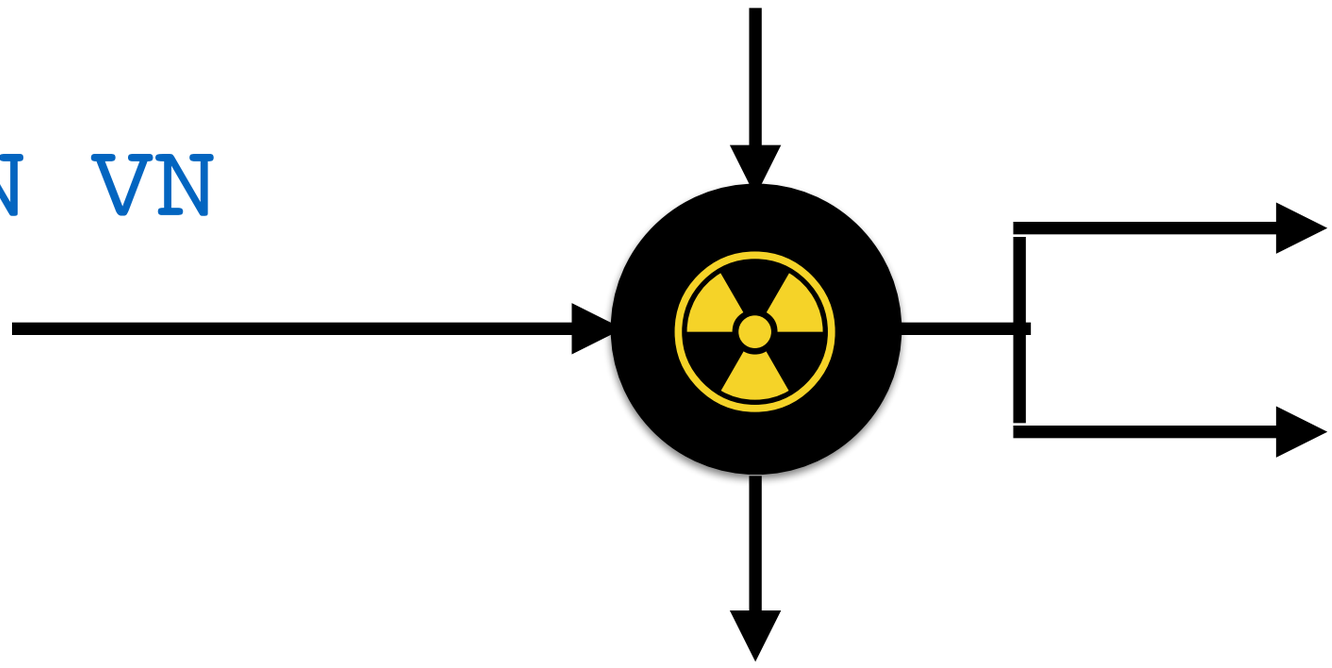


{ VN VN
▷ (n~Z)U(n~Z)}

unconstrained

{_ _ ▷ ε}

VN VN

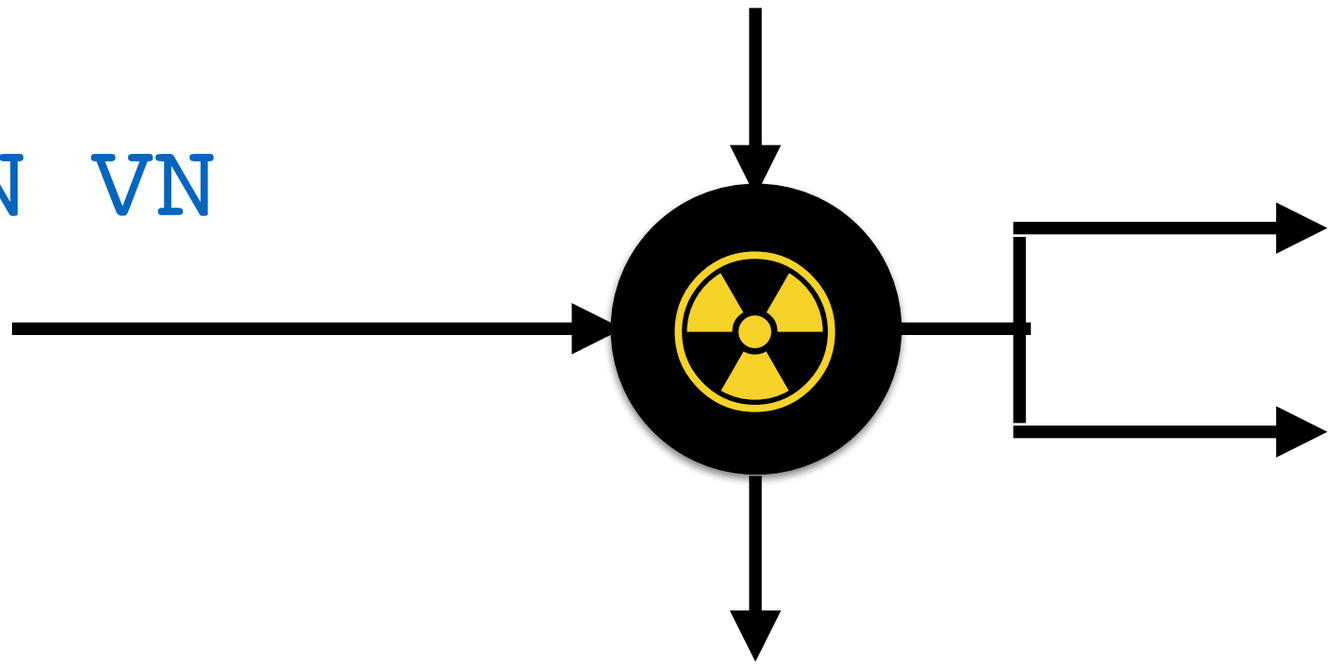


{ VN VN
▷ (n~Z)U(n~Z)}

{⊥ _ ▷ ε
,VN ⊥ ▷ n~Z}

unconstrained

{_ _ ▷ ε}



VN VN

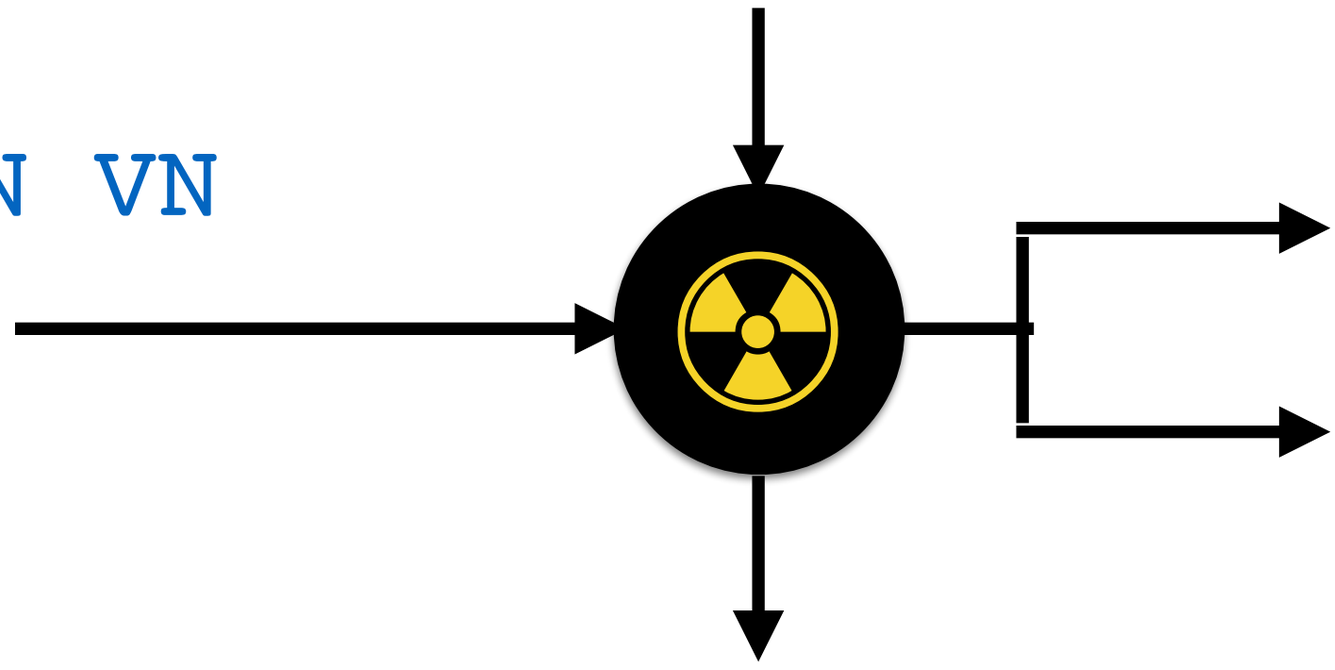
{ VN VN
▷ (n~Z)U(n~Z)}

{⊥ _ ▷ ε
,VN ⊥ ▷ n~Z}

{(VC _ _) _ ▷ n ~ S m
, VN (VC _ _) ▷ (n ~ Z) U (n ~ S m)}

unconstrained

{_ _ ▷ ε}



VN VN

{ VN VN
▷ (n~Z)U(n~Z)}

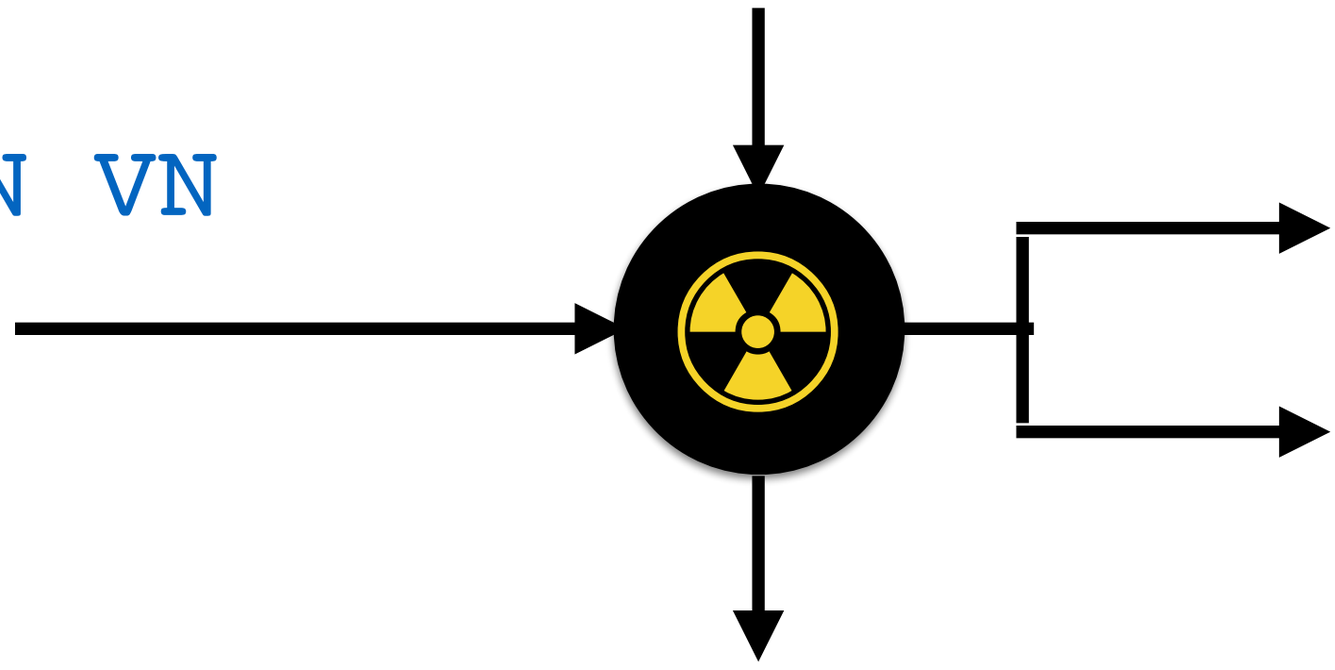
{⊥ _ ▷ ε
,VN ⊥ ▷ n~Z}

{(VC _ _) _ ▷ n ~ S m
, VN (VC _ _) ▷ (n ~ Z) U (n ~ S m)}

inconsistent

unconstrained

{_ _ ▷ ε}



VN VN

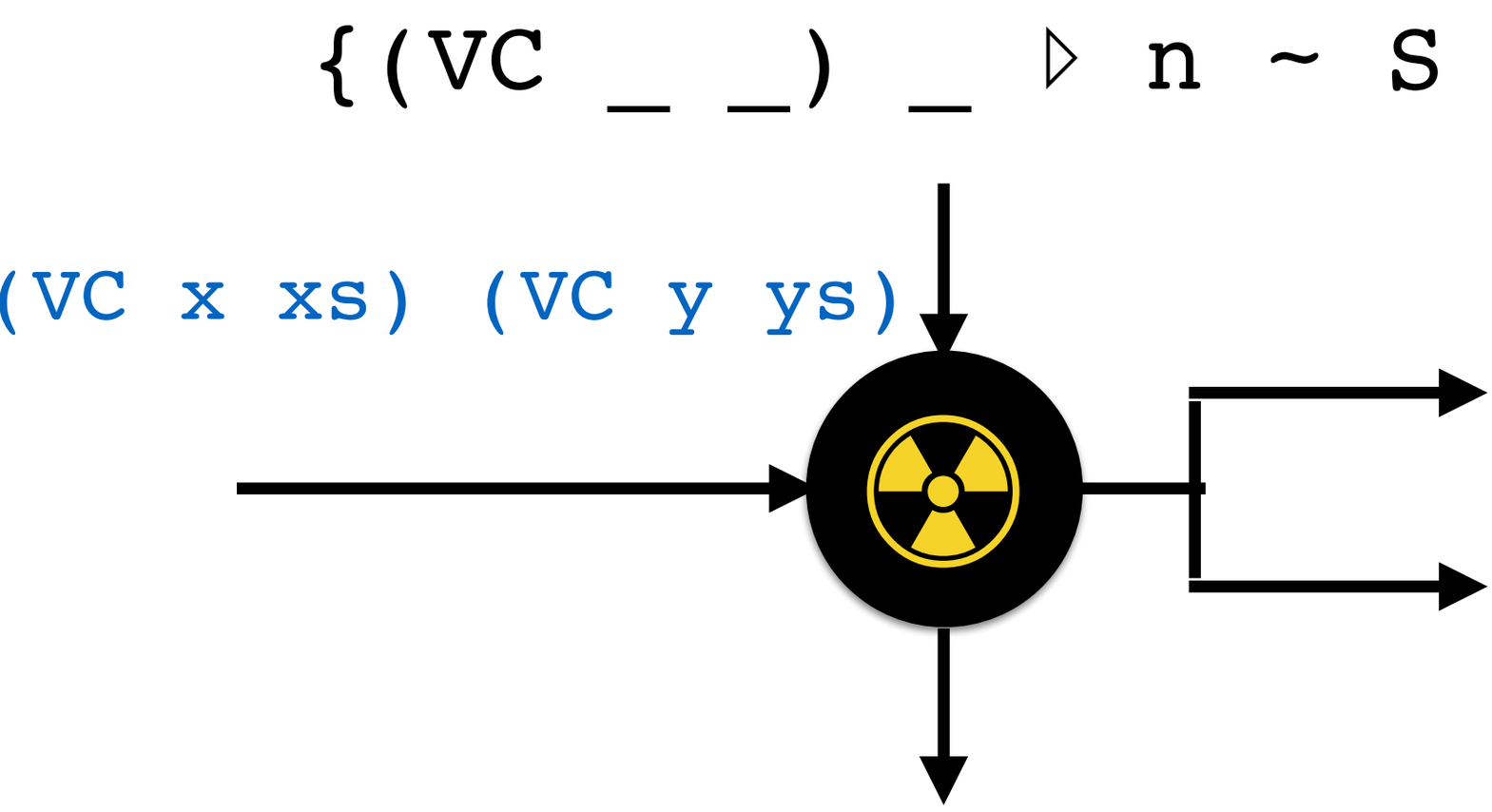
{ VN VN
▷ (n~Z)U(n~Z)}

{⊥ _ ▷ ε
,VN ⊥ ▷ n~Z}

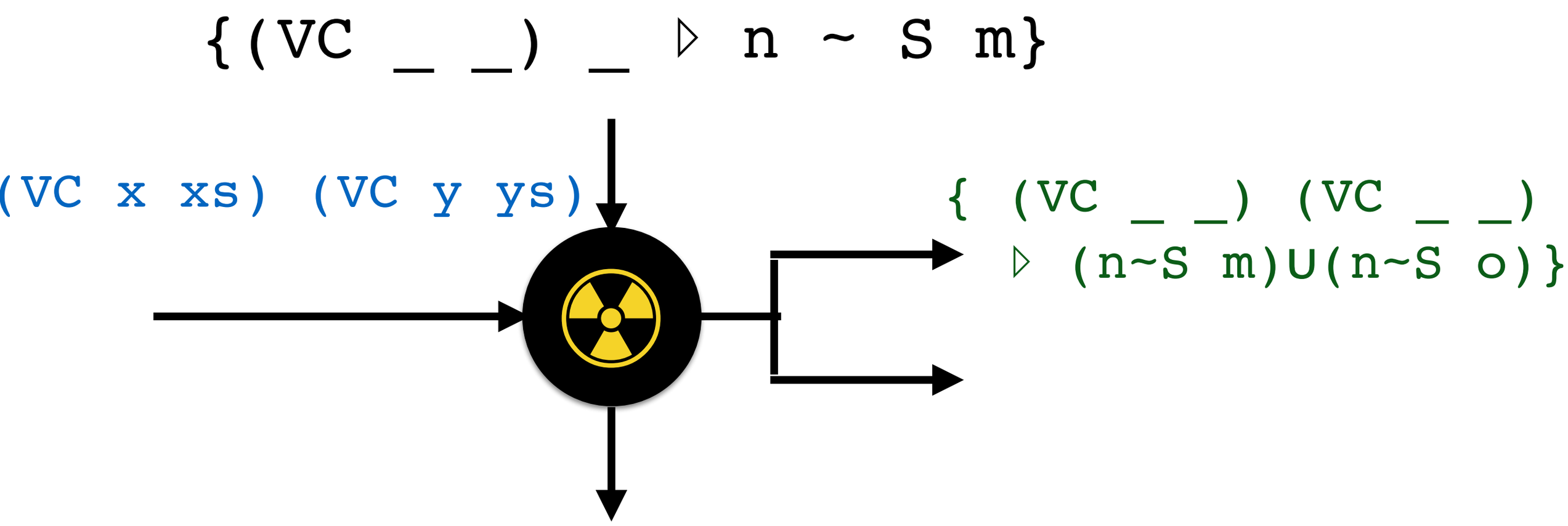
{(VC _ _) _ ▷ n ~ S m
, ~~VN (VC _ _) ▷ (n ~ Z) U (n ~ S m)~~}

inconsistent

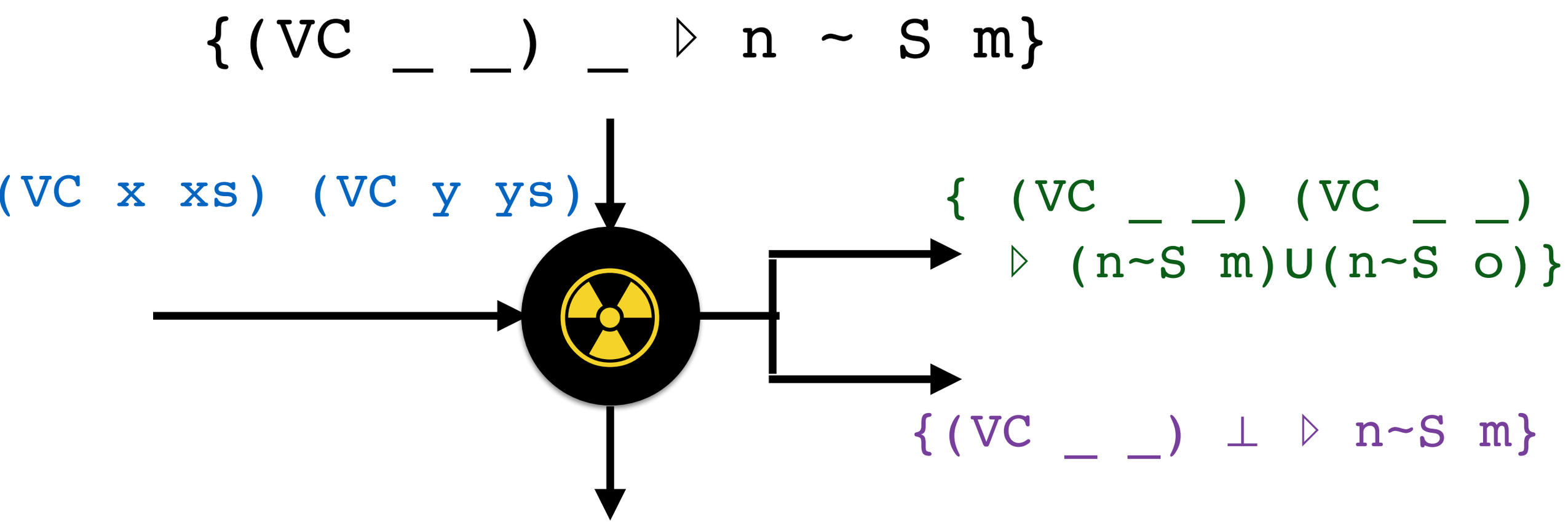
continued



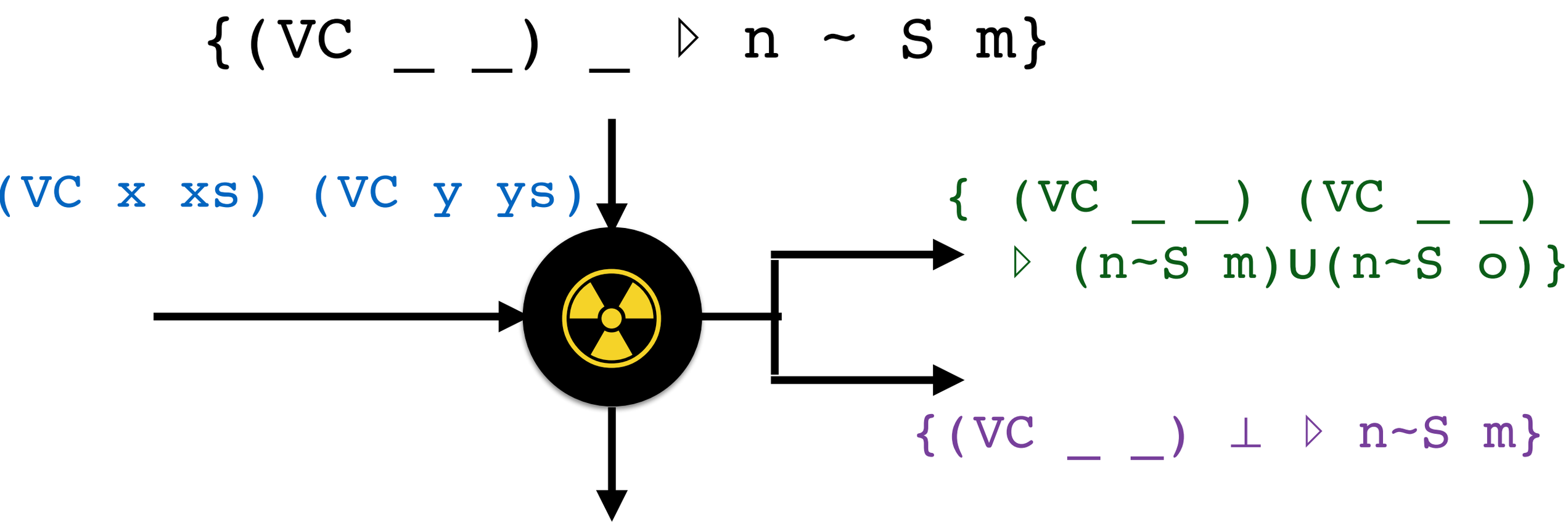
continued



continued

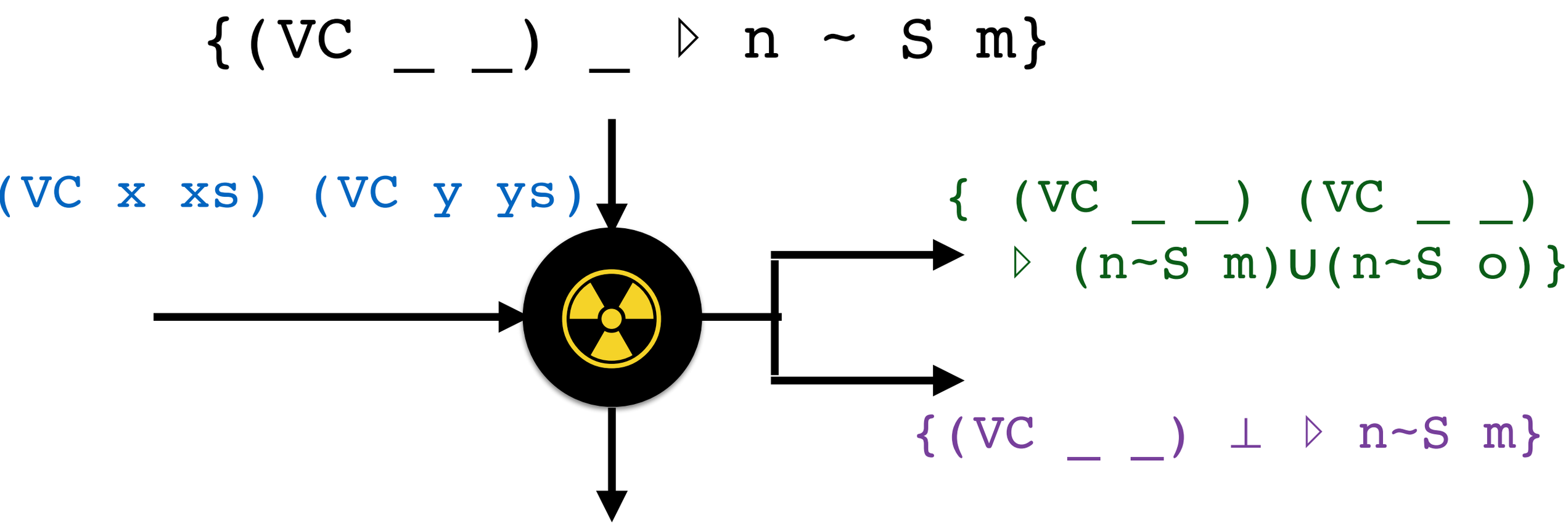


continued



{(VC _ _) VN ▷ (n ~ S m) U (n ~ Z)}

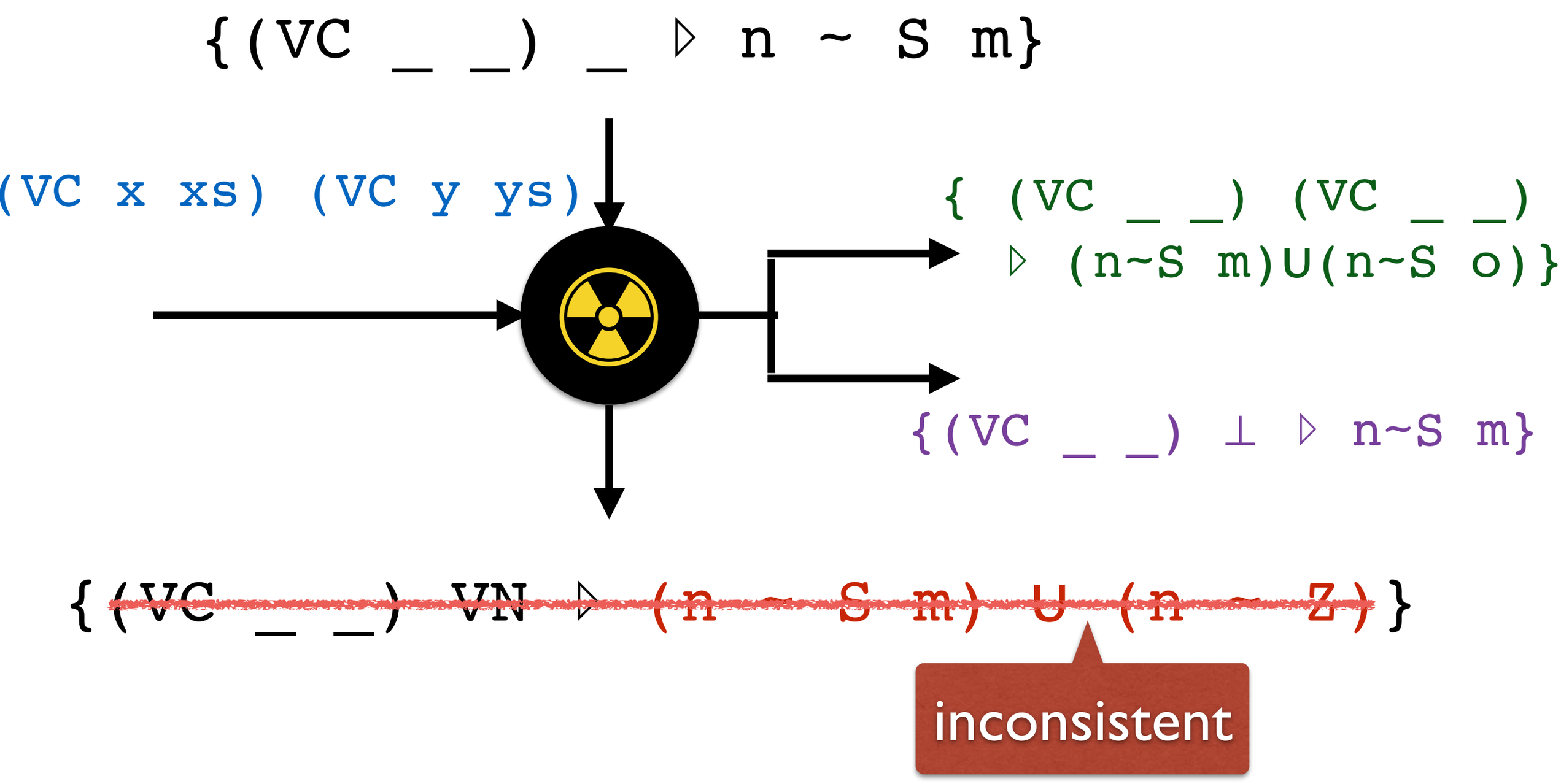
continued



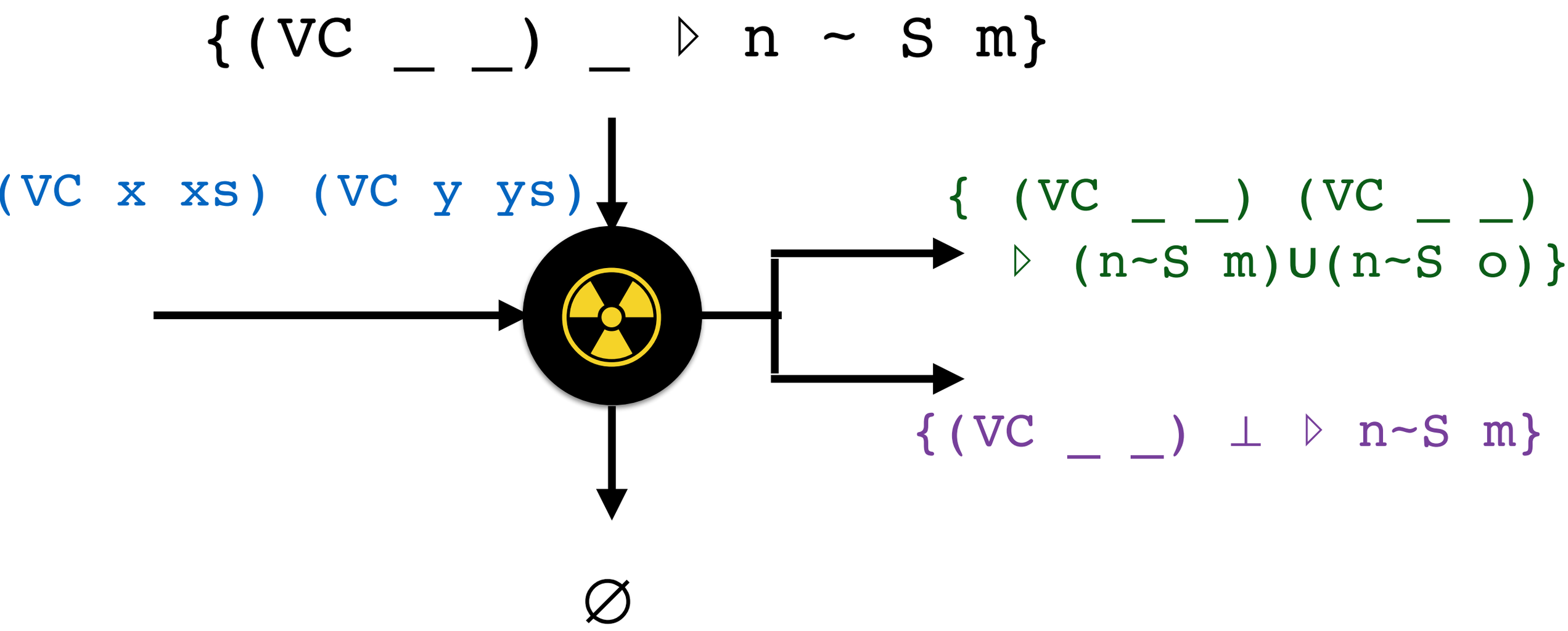
{(VC _ _) VN ▷ (n ~ S m) U (n ~ Z)}

inconsistent

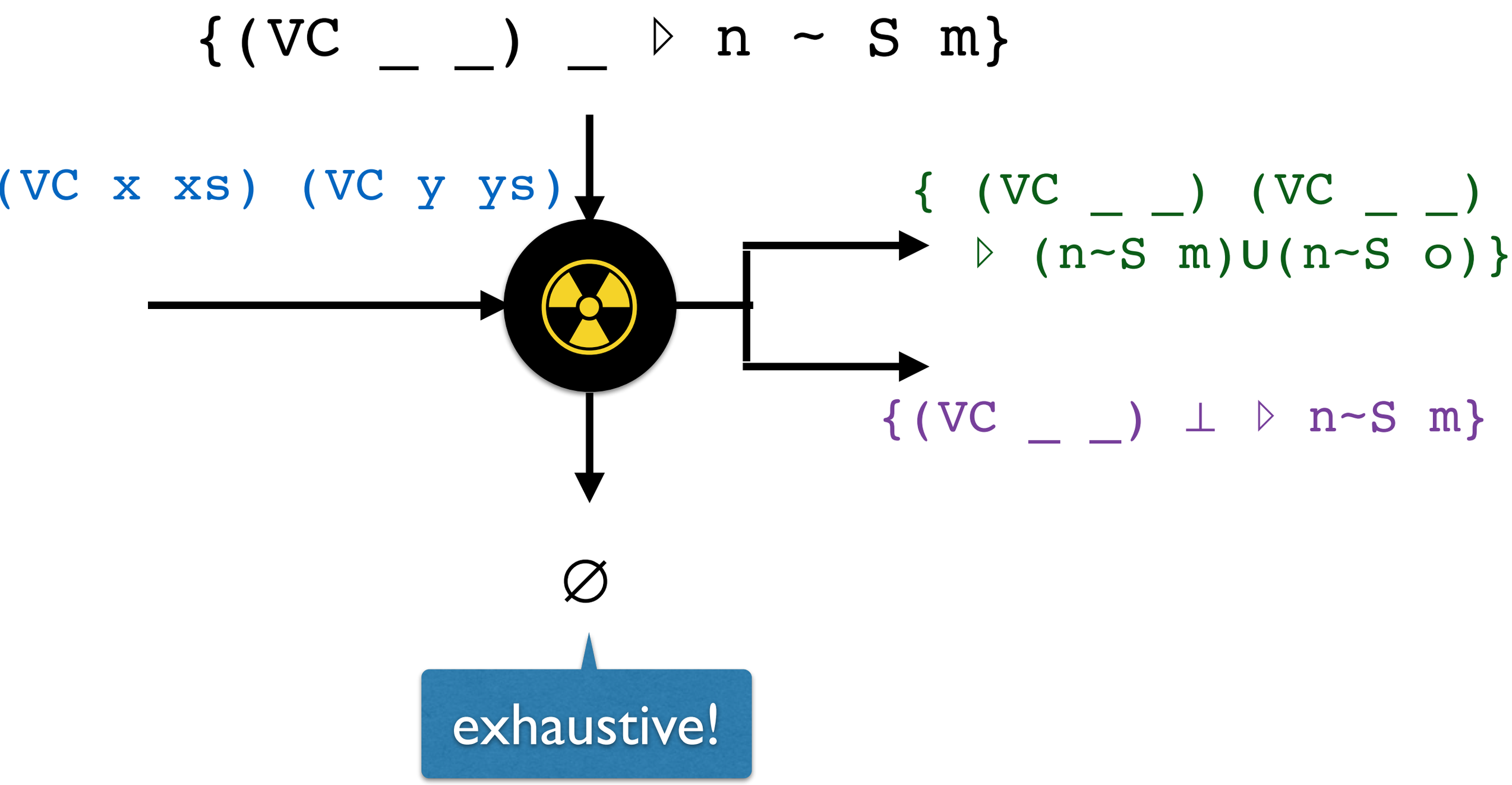
continued


{(VC _ _) _ ▷ n ~ S m}
(VC x xs) (VC y ys)
{ (VC _ _) (VC _ _)
▷ (n~S m)U(n~S o)}
{(VC _ _) ⊥ ▷ n~S m}
{(VC _ _) VN ▷ (n ~ S m) U (n ~ Z)}
inconsistent

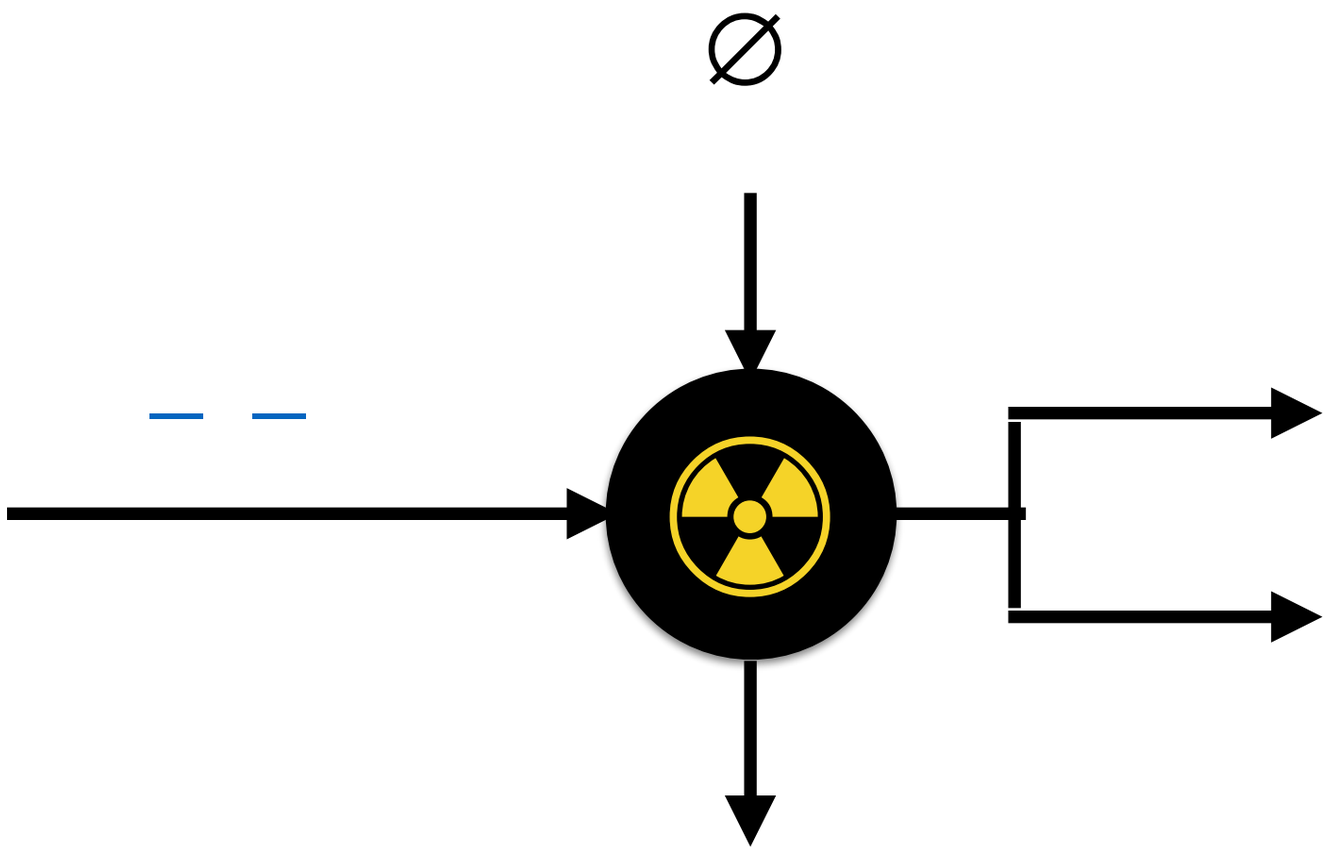
continued



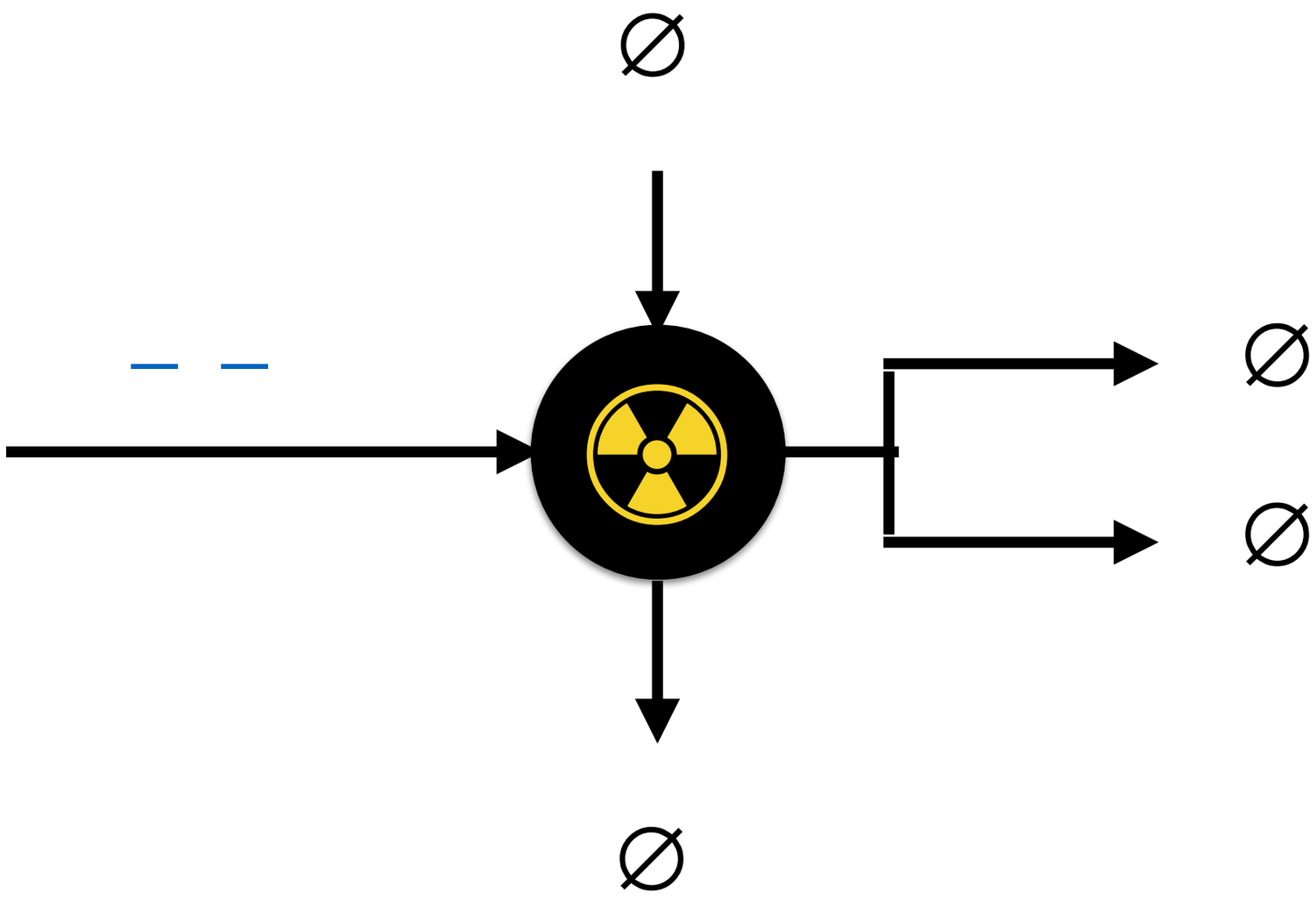
continued



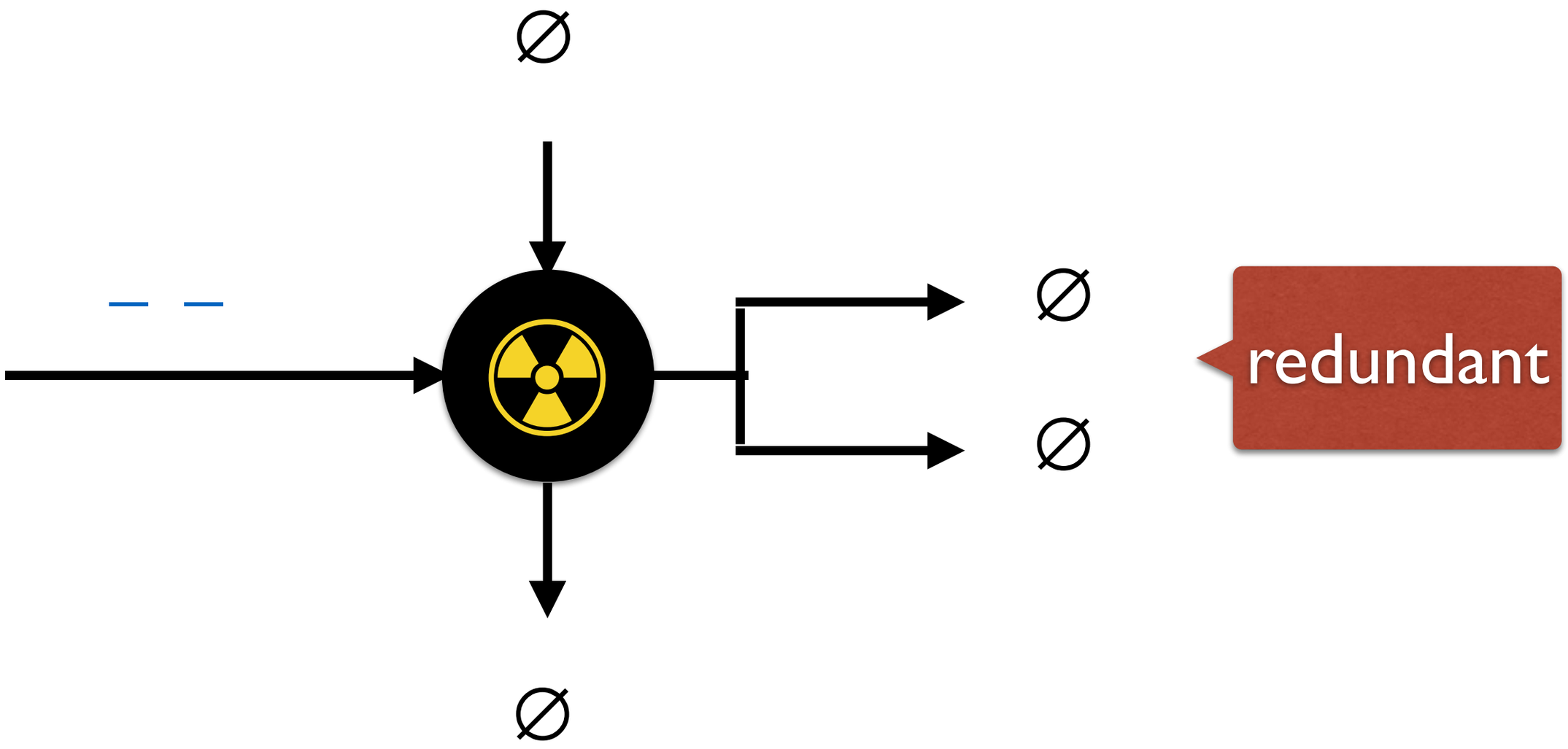
suppose one more catch-all clause



suppose one more catch-all clause



suppose one more catch-all clause



Guards

Guards

```
abs x   | x < 0    =   …
        | x >= 0   =   …
```

Guard Core Syntax

Pattern

```
p ::= x
    |   K p⃗
    |   p <- e
```

variable names

pattern guards

```
abs x (True <- x < 0) =  …
abs x (True <- x >= 0)  =  …
```

More Desugaring

literal patterns

```
f :: Num a => a -> [b] -> …
f 0 [] =  …
```

More Desugaring

literal patterns

```
f :: Num a => a -> [b] -> …
f 0 [] =  …
```

```
f x (True <- x == fromInteger 0) [] = …
```

More Desugaring

literal patterns

```
f :: Num a => a -> [b] -> …
f 0 [] =  …
```

```
f x (True <- x == fromInteger 0) [] = …
```

view patterns

```
g (toRad -> (theta,r)) = …
```

More Desugaring

literal patterns

```
f :: Num a => a -> [b] -> …
f 0 [] =  …
```

```
f x (True <- x == fromInteger 0) [] = …
```

view patterns

```
g (toRad -> (theta,r)) = …
```

```
g x ((theta,r) <- toRad x) = …
```

Guarded Value Abstraction

Value Abstraction

$$v ::= \Gamma \vdash \vec{u} \triangleright \Delta$$

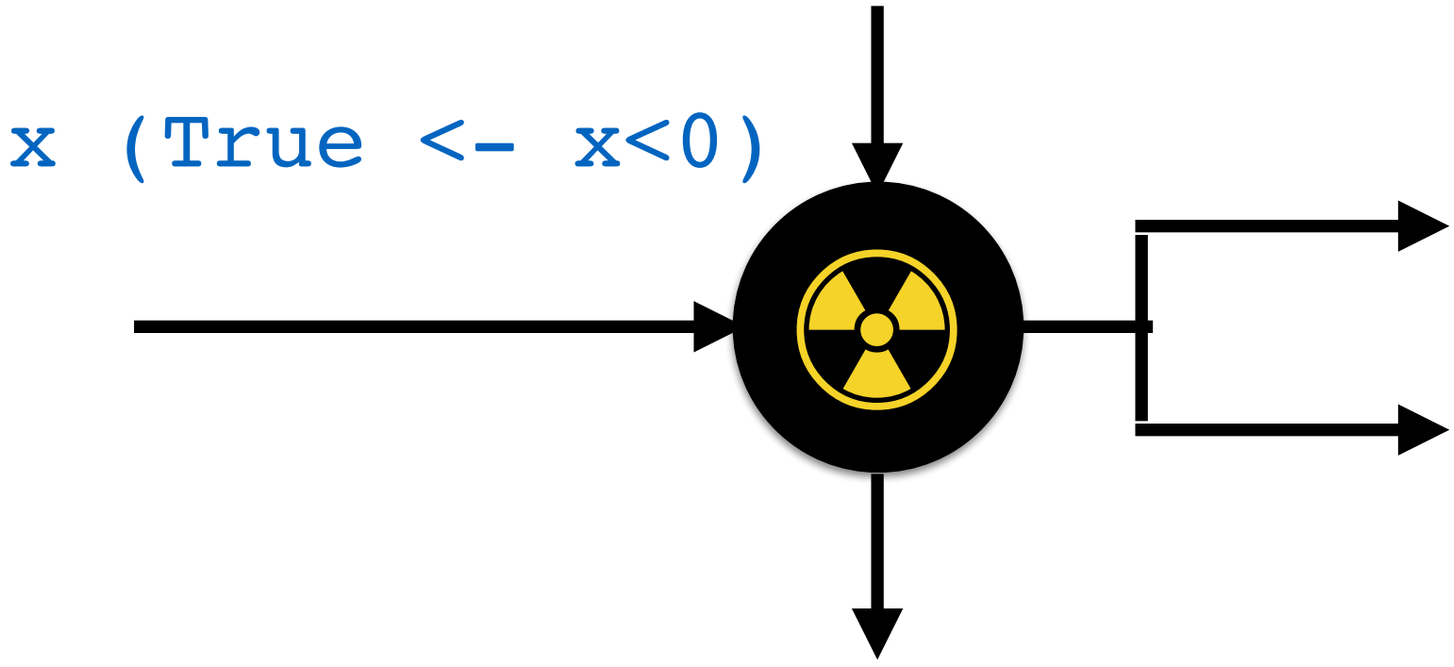
$$u ::= x \mid K\ \vec{u}$$

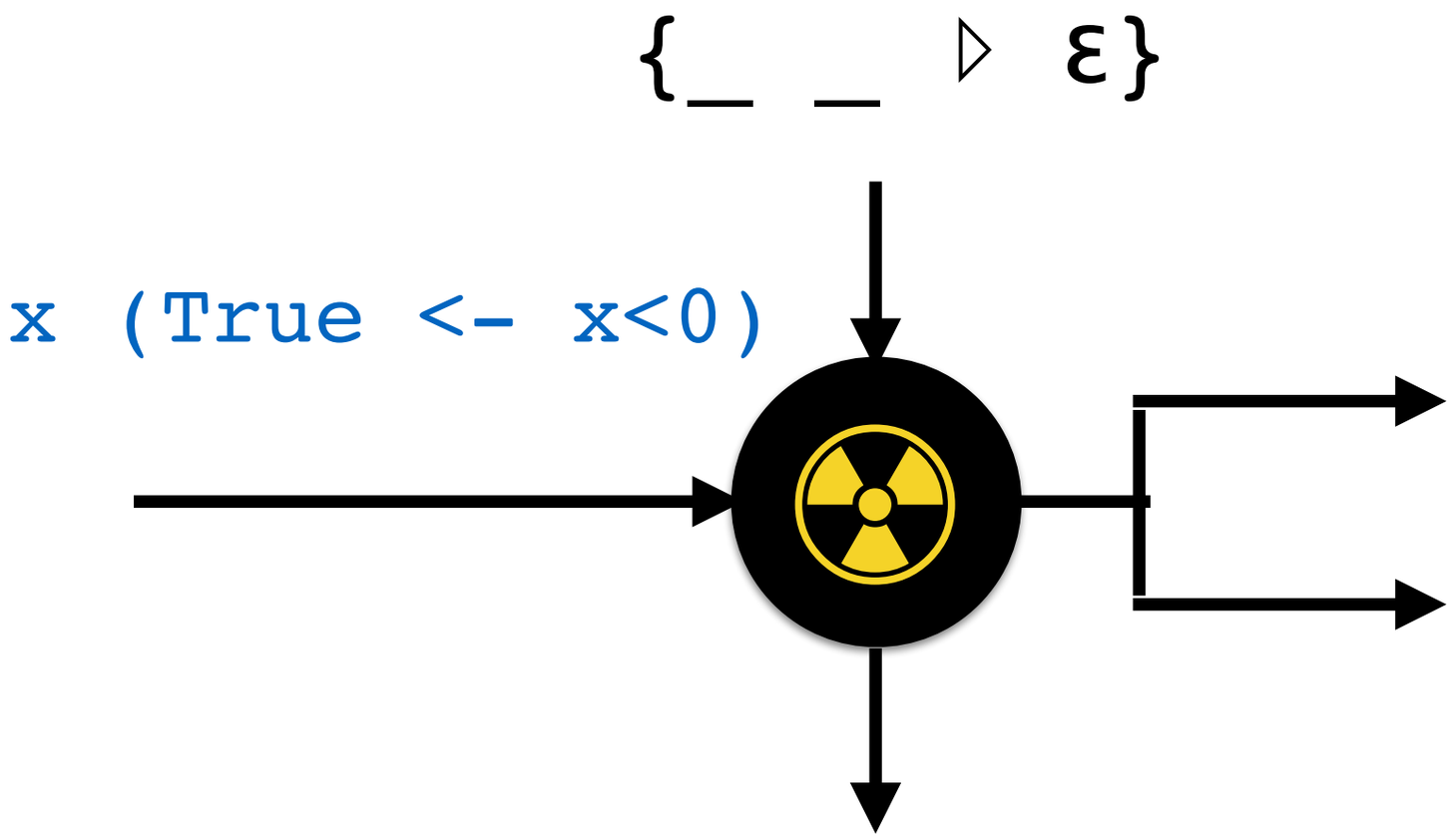
$$\Gamma ::= \varepsilon \mid \Gamma, a \mid \Gamma, x{:}\tau$$

$$\Delta ::= \varepsilon \mid \Delta \cup \Delta \mid \tau \sim \tau \mid x \approx e \mid x \approx \bot$$

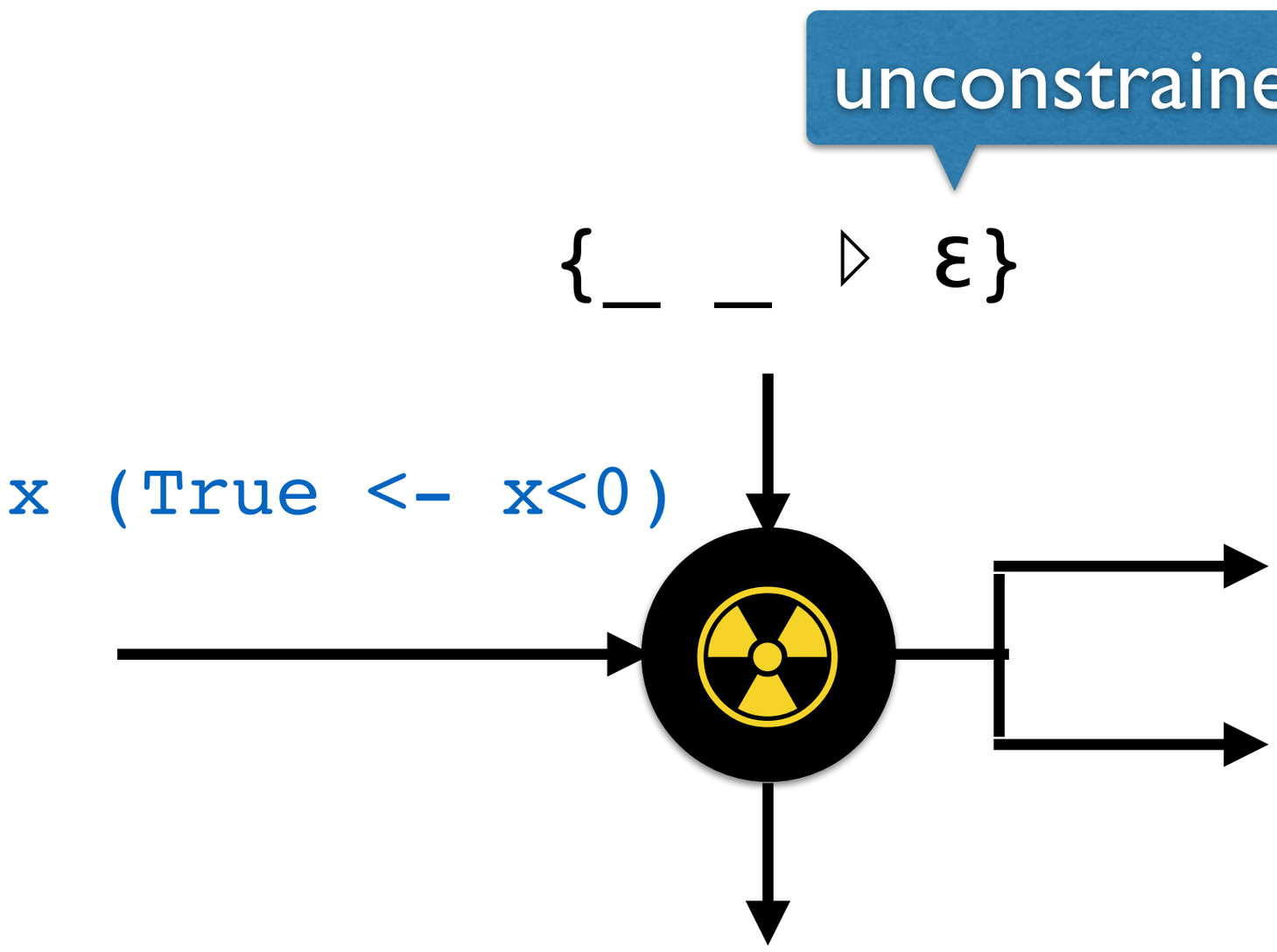
term constraint

strictness constraint

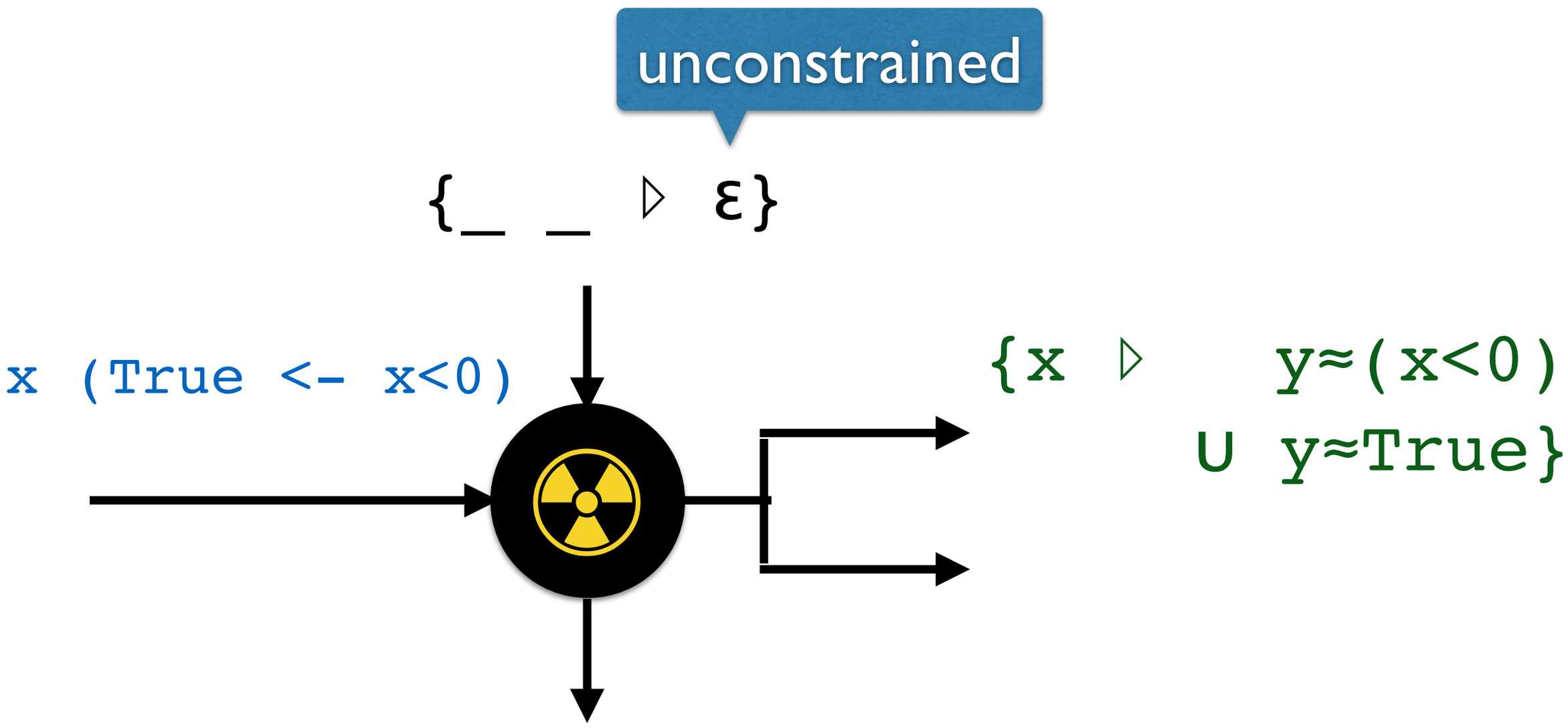

x (True <- x<0)


{_ _ ▷ ε}
x (True <- x<0)

unconstrained



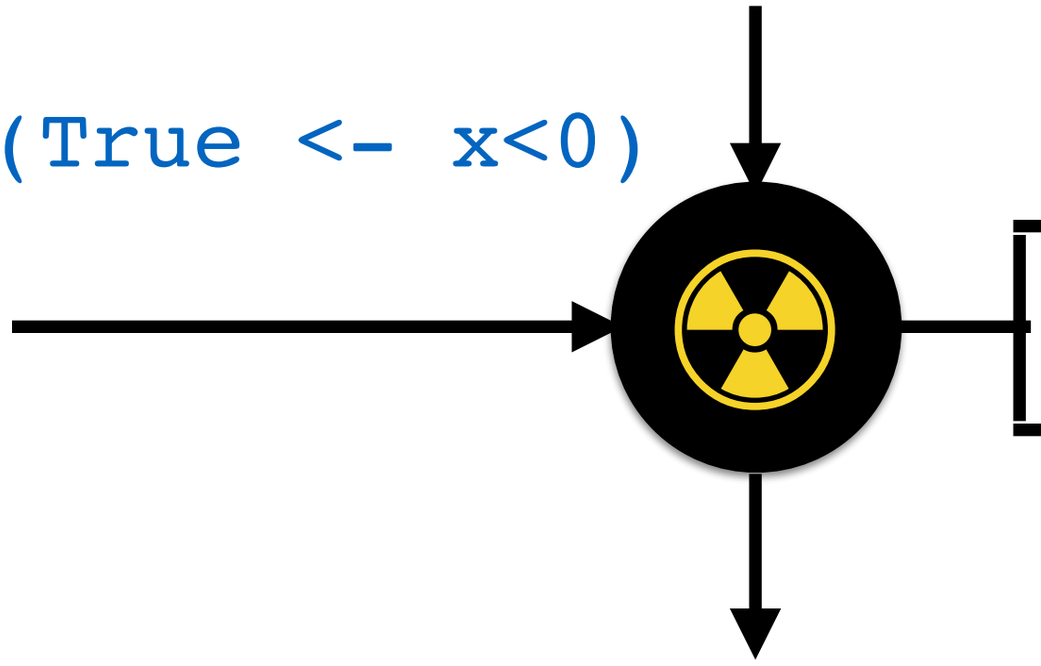
unconstrained



unconstrained

{_ _ ▷ ε}

x (True <- x<0)

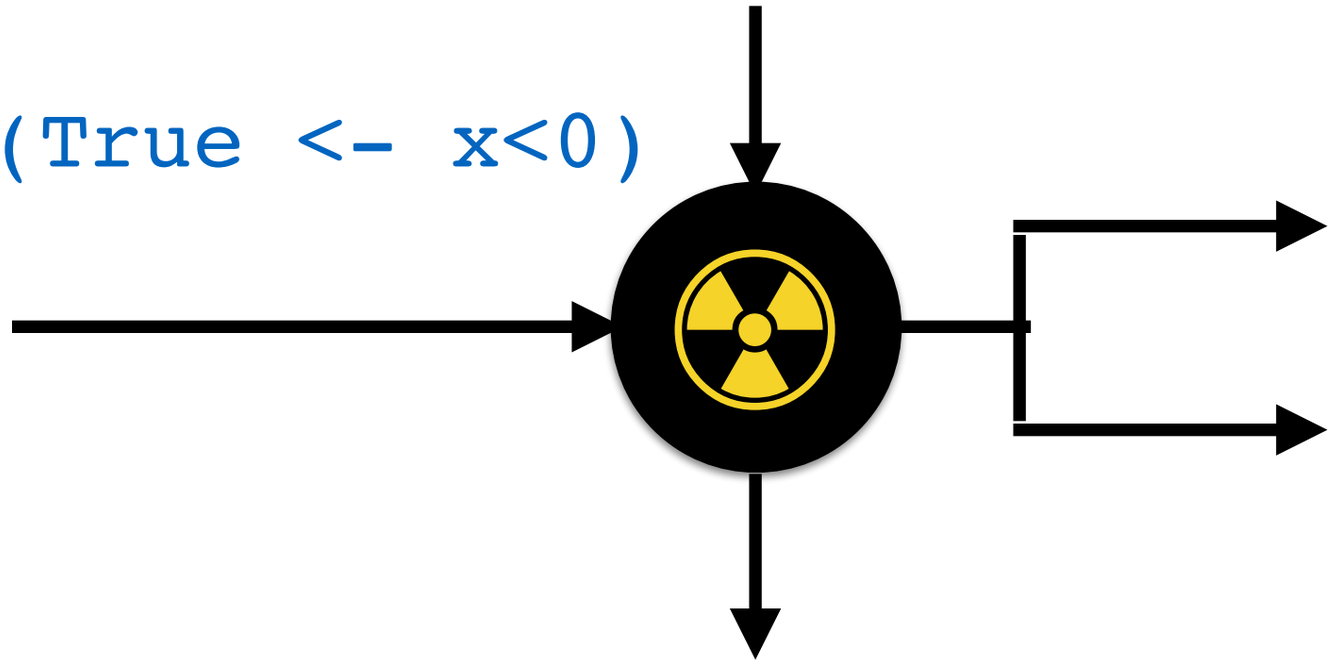


{x ▷ y≈(x<0)
U y≈True}

{x ▷ y≈(x<0)
U y≈⊥}

unconstrained

{_ _ ▷ ε}



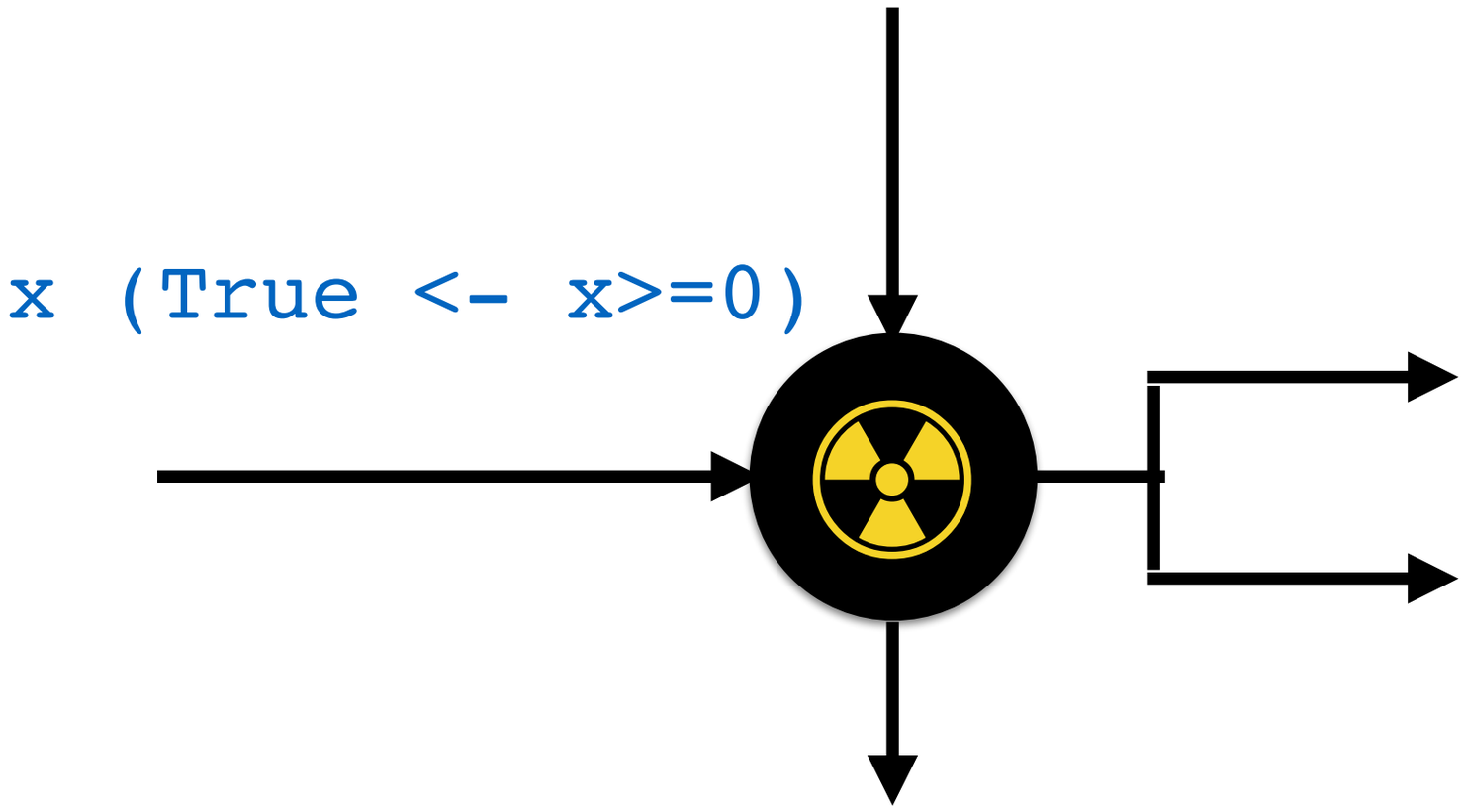
x (True <- x<0)

{x ▷ y≈(x<0) U y≈True}

{x ▷ y≈(x<0) U y≈⊥}

{x ▷ y≈(x<0) U y≈False}

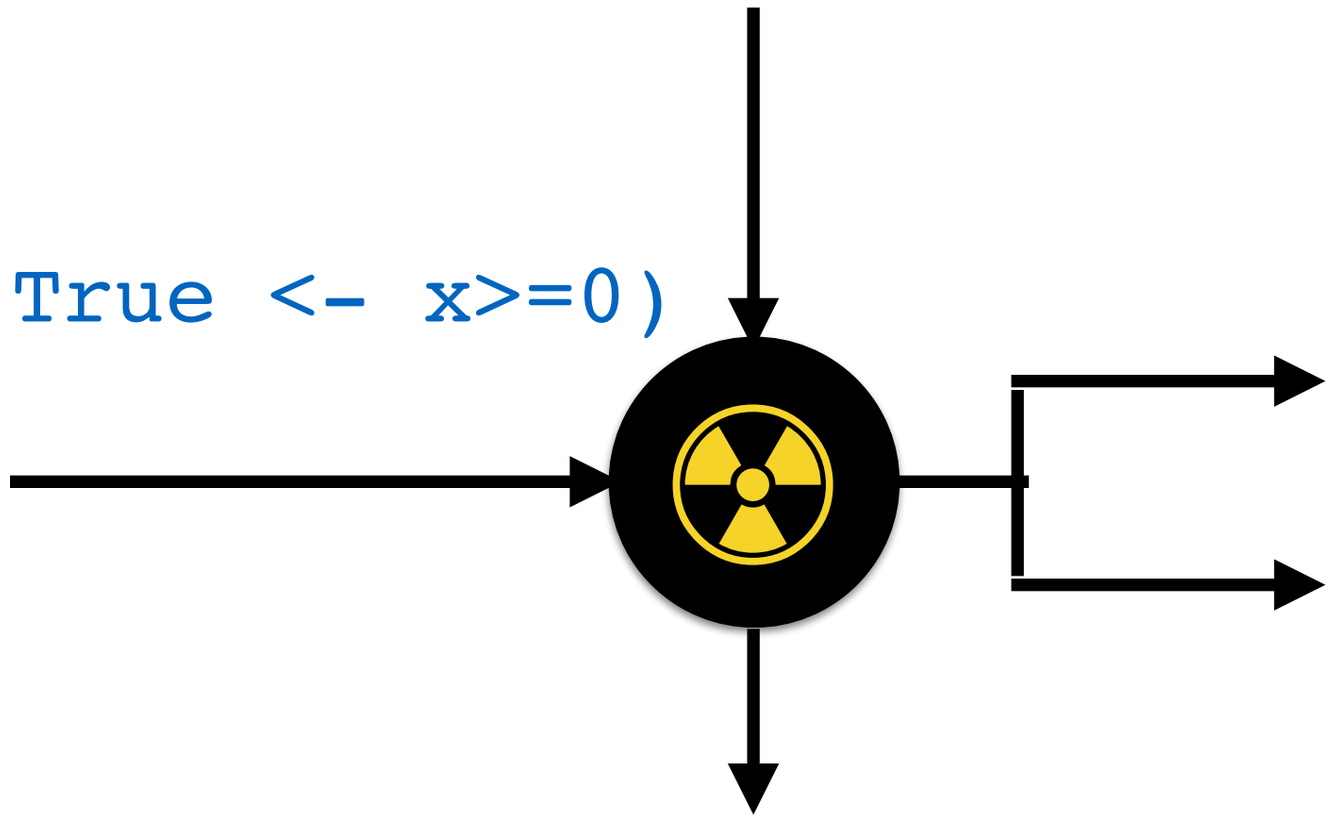
continued



continued

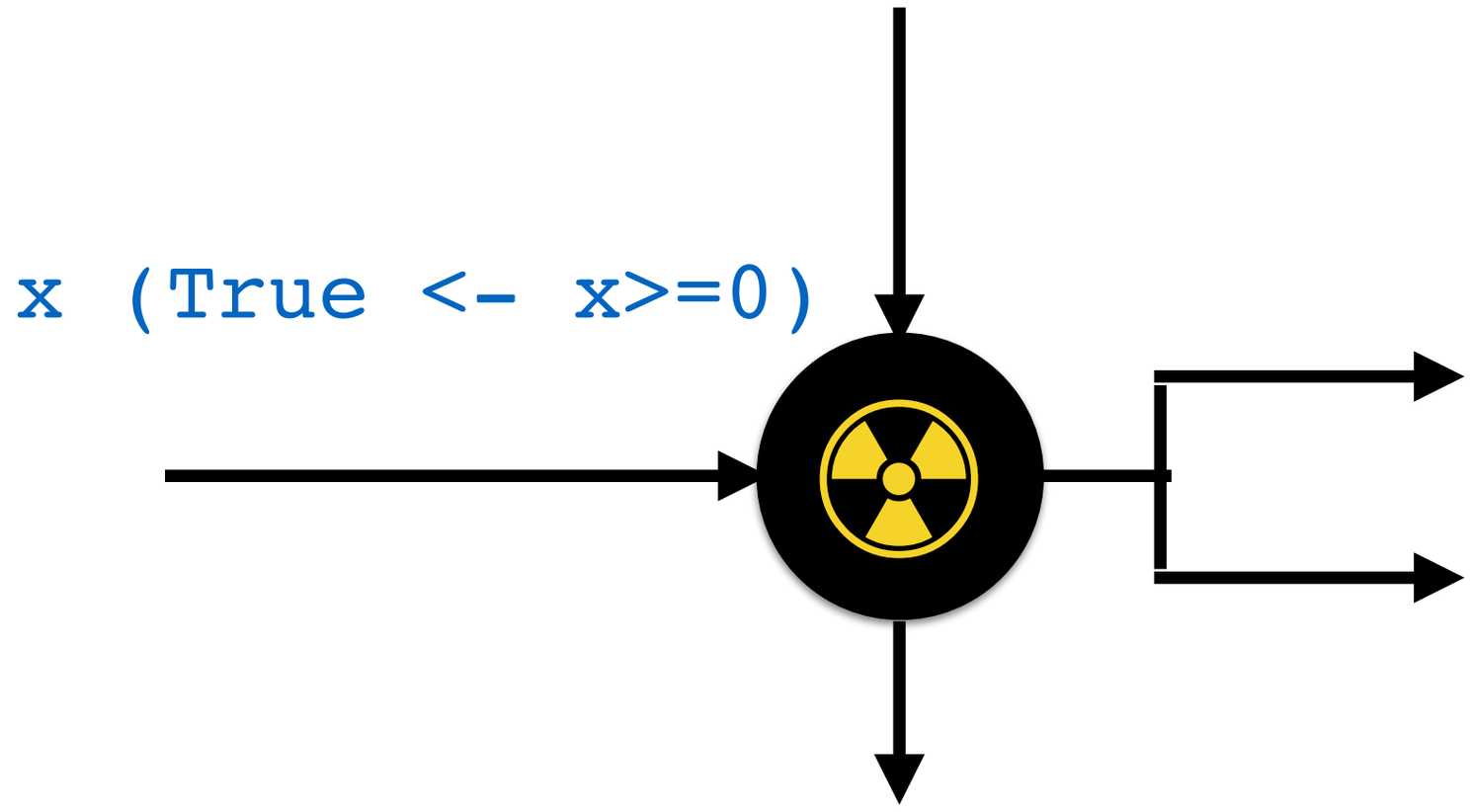
{_ _ ▷ x ▷ y≈(x<0) ∪ y≈False}

x (True <- x>=0)



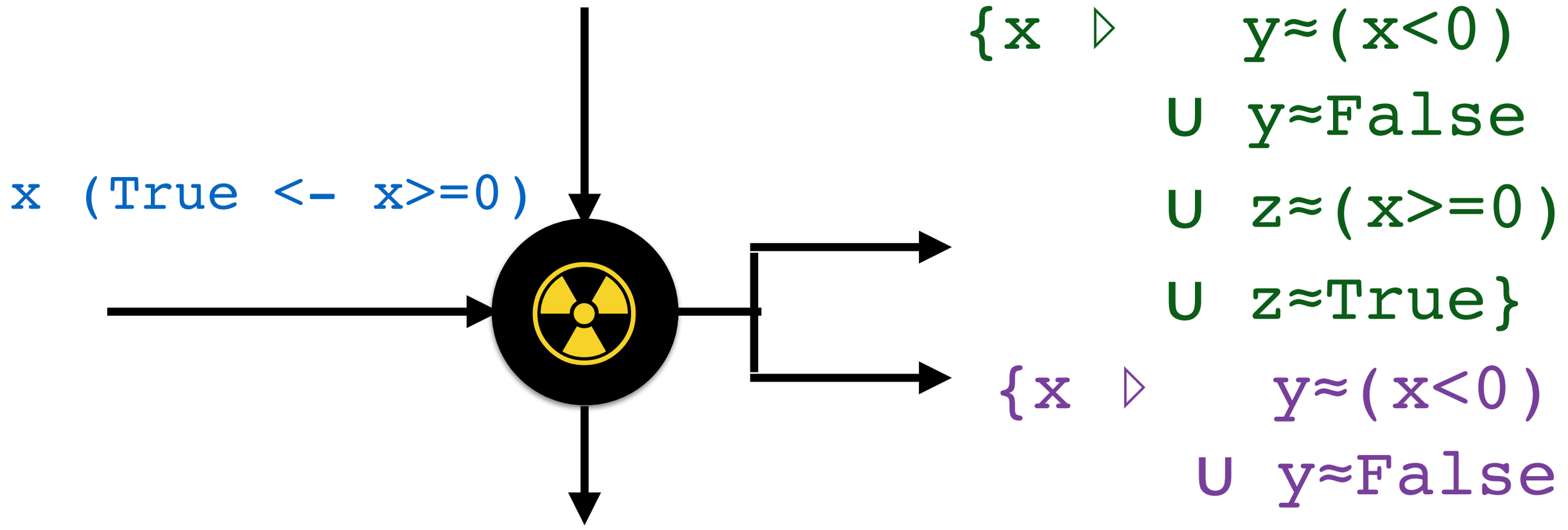
continued

{_ _ ▷ x ▷ y≈(x<0) ∪ y≈False}

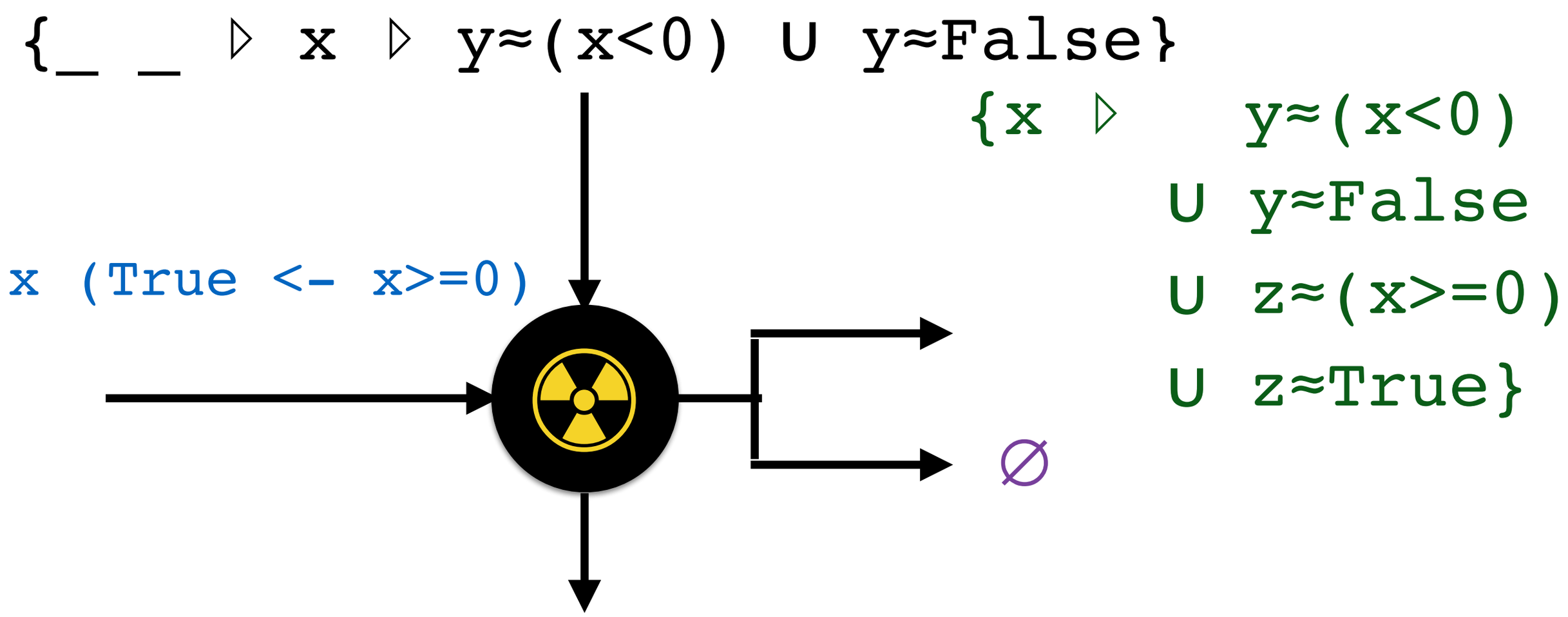


{x ▷ y≈(x<0)
∪ y≈False
∪ z≈(x>=0)
∪ z≈True}

continued



continued

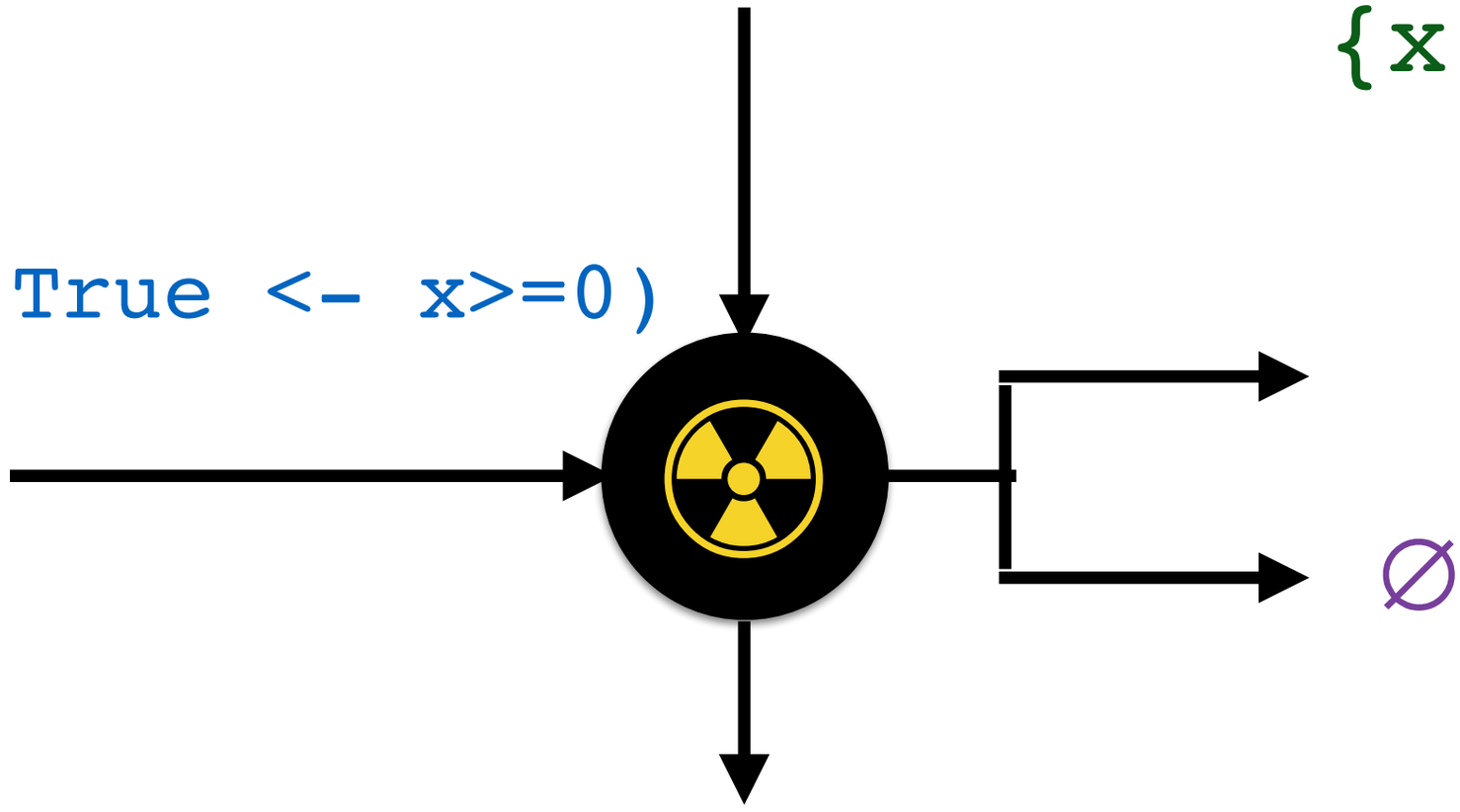


continued

{_ _ ▷ x ▷ y≈(x<0) ∪ y≈False}

{x ▷ y≈(x<0)
∪ y≈False
∪ z≈(x>=0)
∪ z≈True}

x (True <- x>=0)



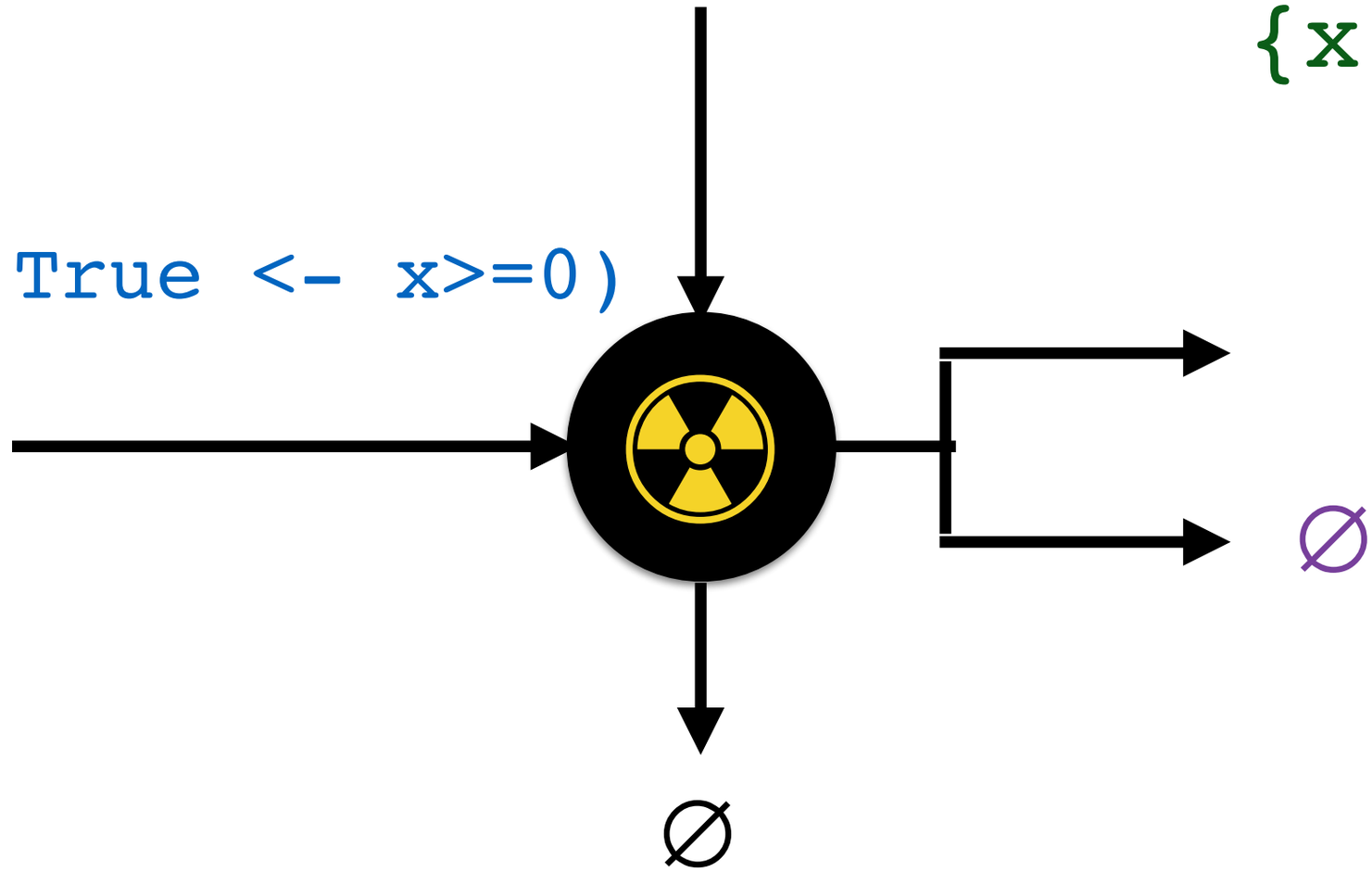
{x ▷ y≈(x<0)
∪ y≈False
∪ z≈(x>=0)
∪ z≈False}

continued

{_ _ ▷ x ▷ y≈(x<0) ∪ y≈False}

{x ▷ y≈(x<0)
∪ y≈False
∪ z≈(x>=0)
∪ z≈True}

x (True <- x>=0)



Evaluation

Prototype Implementation

- GHC branch
- 504 LoC (vs. old 588 LoC)
- Type oracle: GHC type checker
- Term oracle:
  for now only ($y \approx \mathsf{False} \;\cup\; y \approx \mathsf{False}$)

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	1,008	0	0	0	0	0	3
hoopl	2,147	33	0	0	0	0	3
json-sop	393	0	0	0	0	0	2
lens-sop	280	2	0	0	0	0	2
pretty-sop	27	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11		0	9	0	8	14
ad	1		0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	1,008	0	0	0	0	0	3
hoopl	2,147	33	0	0	0	0	3
json-sop	393	0	0	0	0	0	2
lens-sop	280	2	0	0	0	0	2
pretty-sop	27	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

Original GHC

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11		0			8	14
ad	1		0			0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	1,008	0	0	0	0	0	3
hoopl	2,147	33	0	0	0	0	3
json-sop	393	0	0	0	0	0	2
lens-sop	280	2	0	0	0	0	2
pretty-sop	27	0	0	0	0	0	1

Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

Original GHC

Adhoc Patch

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11		0			8	
ad	1		0			0	
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	1,008	0	0	0	0	0	3
hoopl	2,147	33	0	0	0	0	3
json-sop	393	0	0	0	0	0	2
lens-sop	280	2	0	0	0	0	2
pretty-sop	27	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

Original GHC

Adhoc Patch

Our Approach

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop			0	0	0	0	3
hoopl			0	0	0	0	3
json-sop			0	0	0	0	2
lens-sop			0	0	0	0	2
pretty-sop			0	0	0	0	1
Additional tests			**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

libraries with checking problems

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	[illegible]	0	0	0	0	0	3
hoopl	[illegible]	33	0	0	0	0	3
json-sop	[illegible]	0	0	0	0	0	2
lens-sop	[illegible]	2	0	0	0	0	2
pretty-sop	[illegible]	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses erroneously reported Missing

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	[illegible]	0	0	0	0	0	3
hoopl	[illegible]	33	0	0	0	0	3
json-sop	[illegible]	0	0	0	0	0	2
lens-sop	[illegible]	2	0	0	0	0	2
pretty-sop	[illegible]	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses erroneously reported Missing

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop	[illegible]	0	0	0	0	0	3
hoopl	[illegible]	33	0	0	0	0	3
json-sop	[illegible]	0	0	0	0	0	2
lens-sop	[illegible]	2	0	0	0	0	2
pretty-sop	[illegible]	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses erroneously reported Missing

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	1
ad	1,903	2	0	0	0	0	
boolsimplifier	256	10	0	0	0	0	
d-bus	2,753	45	0	42	0	16	
generics-sop	1,[illegible]3	0	0	0	0	0	
hoopl	[illegible]7	33	0	0	0	0	
json-sop	[illegible]	0	0	0	0	0	2
lens-sop	[illegible]	2	0	0	0	0	2
pretty-sop	[illegible]7	0	0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses erroneously reported Missing

better term oracle needed

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop			0	0	0	0	3
hoopl			0	0	0	0	3
json-sop			0	0	0	0	2
lens-sop			0	0	0	0	2
pretty-sop			0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses reported **R**edundant

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop			0	0	0	0	3
hoopl			0	0	0	0	3
json-sop			0	0	0	0	2
lens-sop			0	0	0	0	2
pretty-sop			0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses reported **R**edundant

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop			0	0	0	0	3
hoopl			0	0	0	0	3
json-sop			0	0	0	0	2
lens-sop			0	0	0	0	2
pretty-sop			0	0	0	0	1
Additional tests	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
lists	66	1	0	0	0	0	3
heterogeneous lists	38	0	0	0	0	0	2
heaps	540	3	0	0	0	0	1

clauses reported Redundant

		GHC-1		GHC-2		New	
Hackage Packages	**LoC**	**M**	**R**	**M**	**R**	**M**	**R**
accelerate	11,393	11	0	9	0	8	14
ad	1,903	2	0	0	0	0	6
boolsimplifier	256	10	0	0	0	0	0
d-bus	2,753	45	0	42	0	16	1
generics-sop			0	0	0	0	3
hoopl			0	0	0	0	3
json-sop			0				2
lens-sop			0				2
pretty-sop			0				1
Additional tests	**LoC**	**M**	**R**				**R**
lists	66	1	0				3
heterogeneous lists	38	0	0				2
heaps	540	3	0	0	0	0	1

clauses reported Redundant

only suppressing catch-all clauses

```
f _ _ = ...
```

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

maximum size of Covered/Unconvered sets per match

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

all in package ad

```
data T = A | B | C
f A A = True
f B B = True
f C C = True
```

Space Explosion?

Maximum size of C/U	Pattern Matches	(%)
$1-9$	8702	97.90
$10-99$	181	2.04
$100-2813$	5	0.06

54 x 54 constructors

all in package ad

```
data T = A | B | C
f A A = True
f B B = True
f C C = True
```

Wrapping Up

Summary

A uniform approach to deal with:

- GADTs
- Laziness
- Guards and extensions

  pattern guards, view patterns, literal patterns, …

Coming soon in GHC release

More in the Paper

- Full Algorithm
- Optimisation Opportunities
- Meta-Theory
- Related Work

GADTs meet their match

Pattern-matching warnings that account for GADTs, guards, and laziness.

Georgios Karachalias
Universiteit Gent
georgios.karachalias@ugent.be

Tom Schrijvers
Katholieke Universiteit Leuven
tom.schrijvers@cs.kuleuven.be

Dimitrios Vytiniotis
Simon Peyton Jones
Microsoft Research Cambridge
{dimitris,simonpj}@microsoft.com

Abstract

For ML and Haskell, accurate warnings when a function definition has redundant or missing patterns are mission critical. But today's compilers generate bogus warnings when the programmer uses guards (even simple ones), GADTs, pattern guards, or view patterns. We give the first algorithm that handles all these cases in a single, uniform framework, together with an implementation in GHC, and evidence of its utility in practice.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords Haskell, pattern matching, Generalized Algebraic Data Types, OUTSIDEIN(X)

1. Introduction

Is this function (in Haskell) fully defined?

```
zip :: [a] -> [b] -> [(a,b)]
zip []     []     = []
zip (a:as) (b:bs) = (a,b) : zip as bs
```

No, it is not: the call `(zip [] [True])` will fail, because neither equation matches the call. Good compilers will report missing patterns, to warn the programmer that the function is only partially defined. They will also warn about completely-overlapped, and hence redundant, equations. Although technically optional for soundness, these warnings are incredibly useful in practice, especially when the program is refactored (i.e. throughout its active life), with constructors added and removed from the data type (Section 2).

But what about this function?

```
vzip :: Vect n a -> Vect n b -> Vect n (a,b)
vzip VN     VN     = VN
vzip (VC x xs) (VC y ys) = VC (x,y) (vzip xs ys)
```

where the type `Vect n a` represents lists of length `n` with element type `a`. `Vect` is a Generalised Algebraic Data Type (GADT):

```
data Vect :: Nat -> * -> * where
  VN :: Vect Zero a
  VC :: a -> Vect n a -> Vect (Succ n) a
```

Unlike `zip`, function `vzip` is *fully* defined: a call with arguments of unequal length, such as `(vzip VN (VC True VN))`, is simply ill-typed. Comparing `zip` and `vzip`, it should be clear that only a *type-aware* algorithm can correctly decide whether or not the pattern-matches of a function definition are exhaustive.

Despite the runaway popularity of GADTs, and other pattern-matching features such as view patterns and pattern guards, no production compiler known to us gives accurate pattern-match overlap and exhaustiveness warnings when these features are used. Certainly our own compiler, GHC, does not; and nor does OCaml. In this paper we solve the problem. Our contributions are these:

- We characterise the challenges of generating accurate warnings (Section 2). The problem goes beyond GADTs! There are subtle issues concerning nested patterns, view pattens, guards, and laziness; the latter at least has never even been noticed before.
- We give the first type-aware algorithm for determining missing or redundant patterns (Sections 3 and 4). The algorithm is parameterised over an oracle that can solve constraints: both type constraints and boolean constraints for guards. Extending the oracle allows us to accommodate type system extensions or improve the precision of the reported warnings *without* affecting the main algorithm at all.

  The central abstraction in this algorithm is the compact symbolic representation of a set of values by a triple ($\Gamma \vdash u \triangleright \Delta$) consisting of an environment Γ, a syntactic value abstraction u and a constraint Δ (Section 4.1). The key innovation is to include the constraints Δ to refine the set of values; for example (x:`Int` $\vdash$ `Just` $x \triangleright x>3$) is the set of all applications of `Just` to integers bigger than 3.
- We formalise the correctness of our algorithm (Section 5) with respect to the Haskell semantics of pattern matching.
- We have implemented our algorithm in GHC, a production quality compiler for Haskell (Section 6). The new implementation is of similar code size as its predecessor although it is much more capable. It reuses GHC's existing type constraint solver as an oracle.
- We demonstrate the effectiveness of the new checker on a set of actual Haskell programs submitted by GHC users, for whom inaccurate warnings were troublesome (Section 7).

There is quite a bit of related work, which we discuss in Section 8.

2. The challenges that we tackle

The question of determining exhaustiveness and redundancy of pattern matching has been well studied (Section 8), but almost exclusively in the context of purely structural matching. In this section we identify three new challenges:

- The challenge of GADTs (Section 2.2).
- The challenge of laziness (Section 2.3).
- The challenge of guards (Section 2.4).

To our knowledge, *none* of these difficulties are systematically addressed in the literature, let alone in a real compiler. In this paper we show how to tackle all three, in a single unified framework.

The End