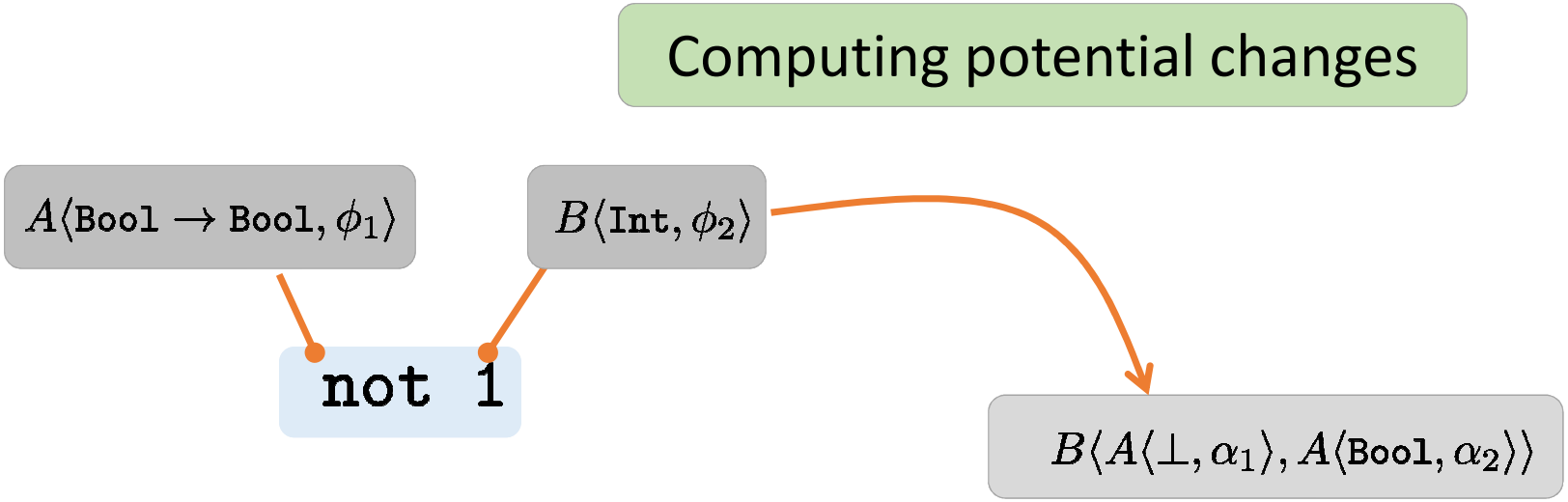
Guided Type Debugging

Sheng Chen and Martin Erwig
School of EECS
Oregon State University

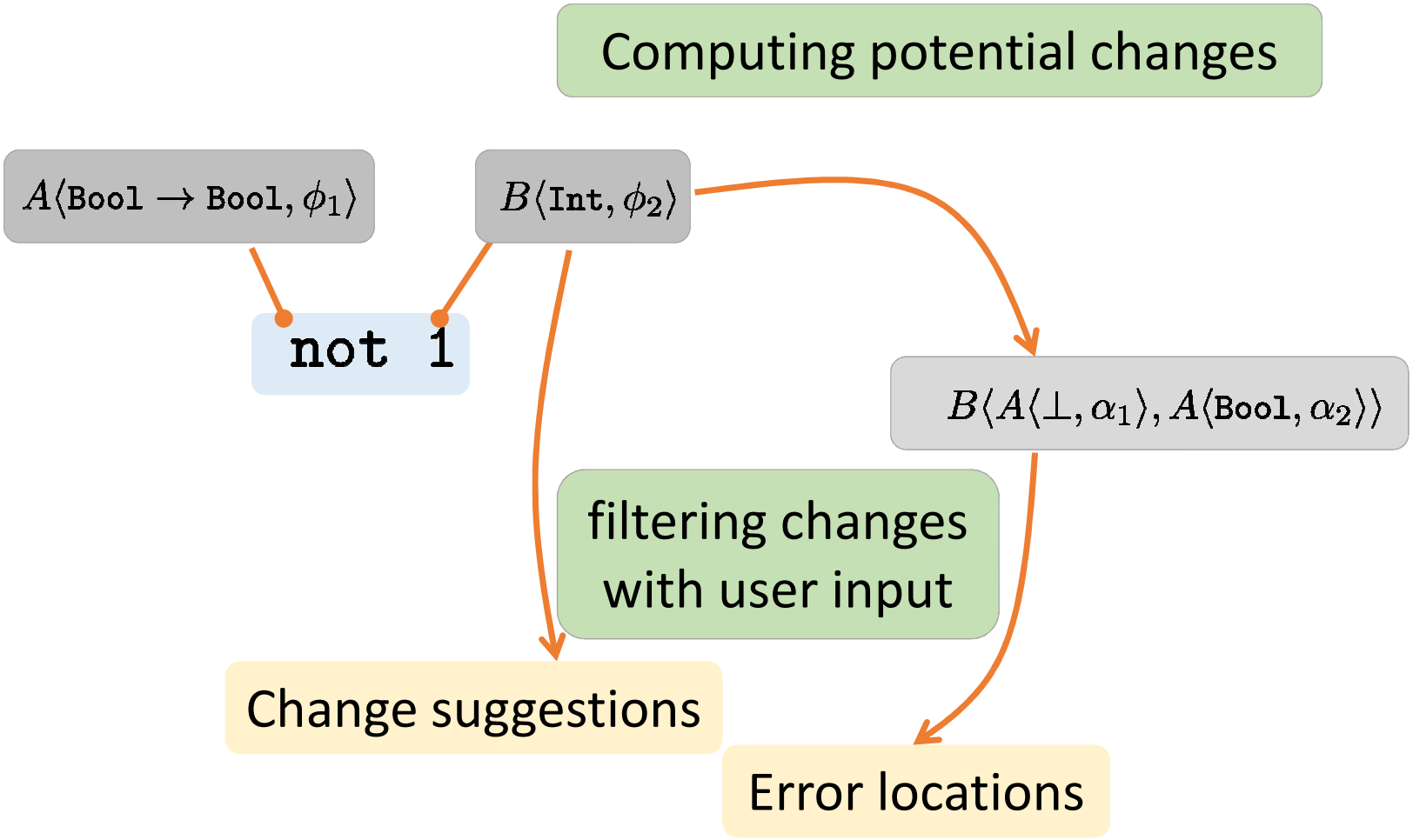
Outline

```
not 1
```

Outline


Computing potential changes
$A\langle \texttt{Bool} \to \texttt{Bool}, \phi_1 \rangle$
$B\langle \texttt{Int}, \phi_2 \rangle$
not 1
$B\langle A\langle \bot, \alpha_1 \rangle, A\langle \texttt{Bool}, \alpha_2 \rangle \rangle$

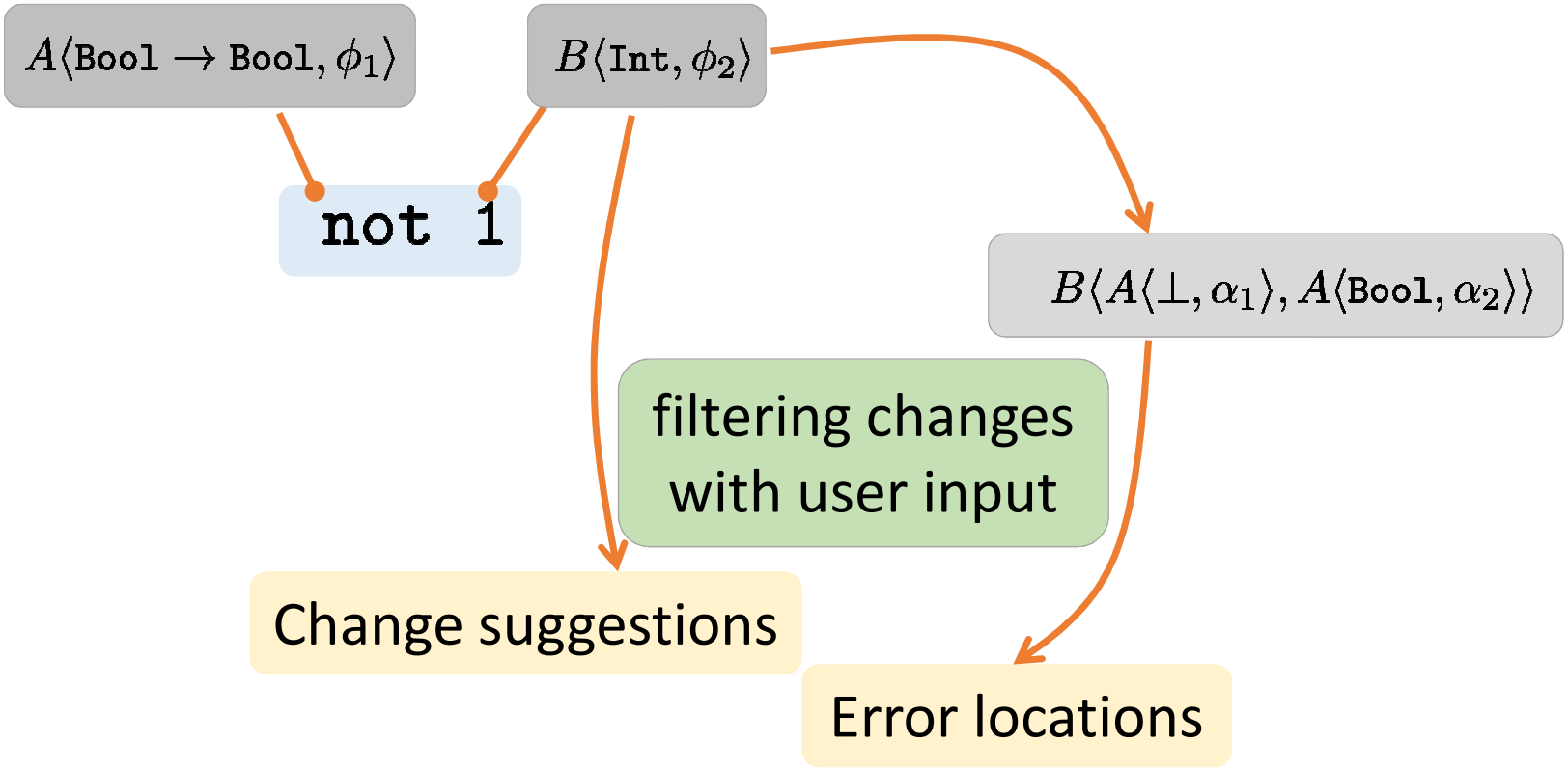
Outline


Computing potential changes
$A\langle \texttt{Bool} \to \texttt{Bool}, \phi_1 \rangle$
$B\langle \texttt{Int}, \phi_2 \rangle$
not 1
$B\langle A\langle \bot, \alpha_1 \rangle, A\langle \texttt{Bool}, \alpha_2 \rangle \rangle$
filtering changes with user input
Change suggestions
Error locations

Outline

previous work

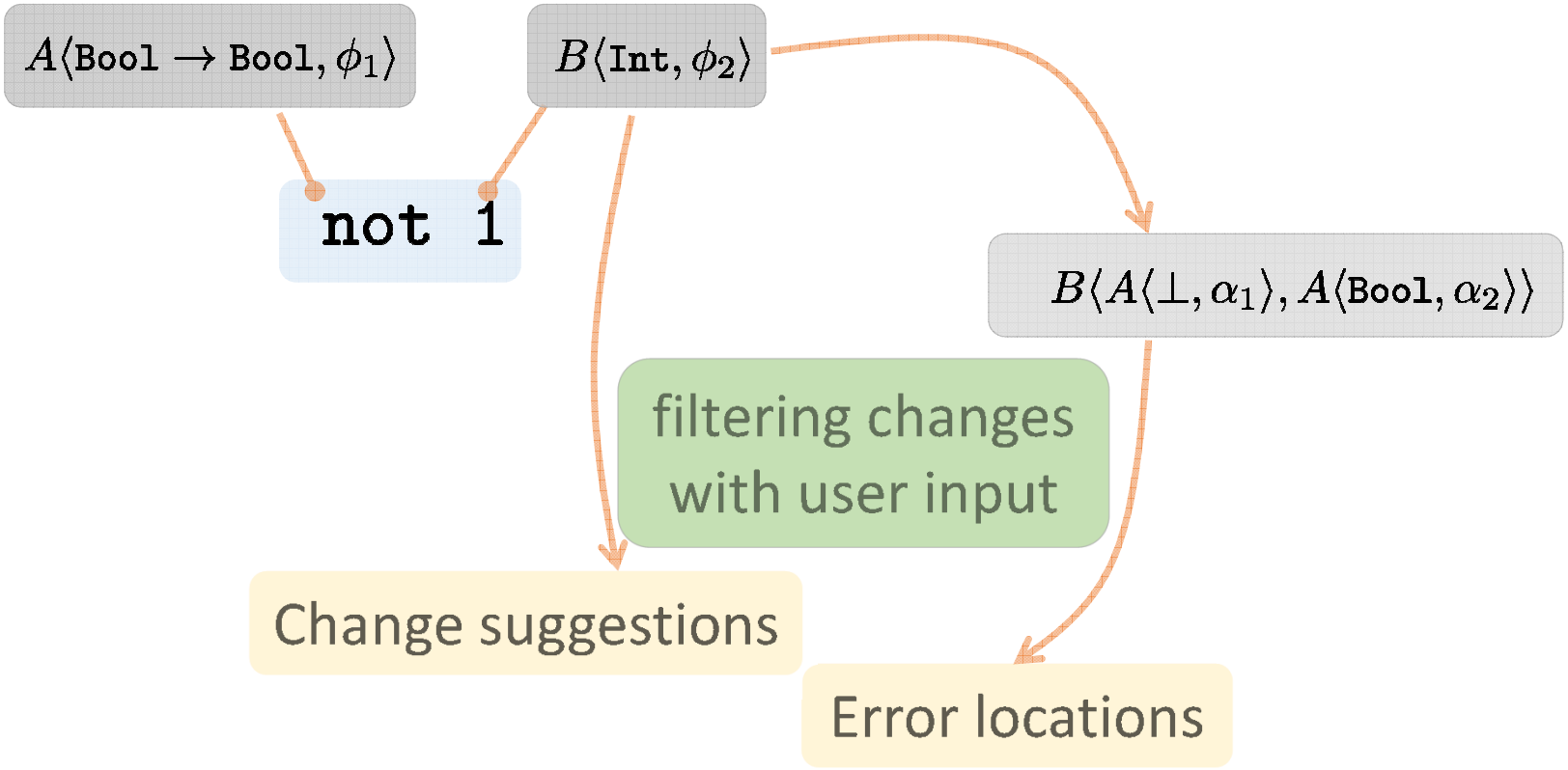
Computing potential changes



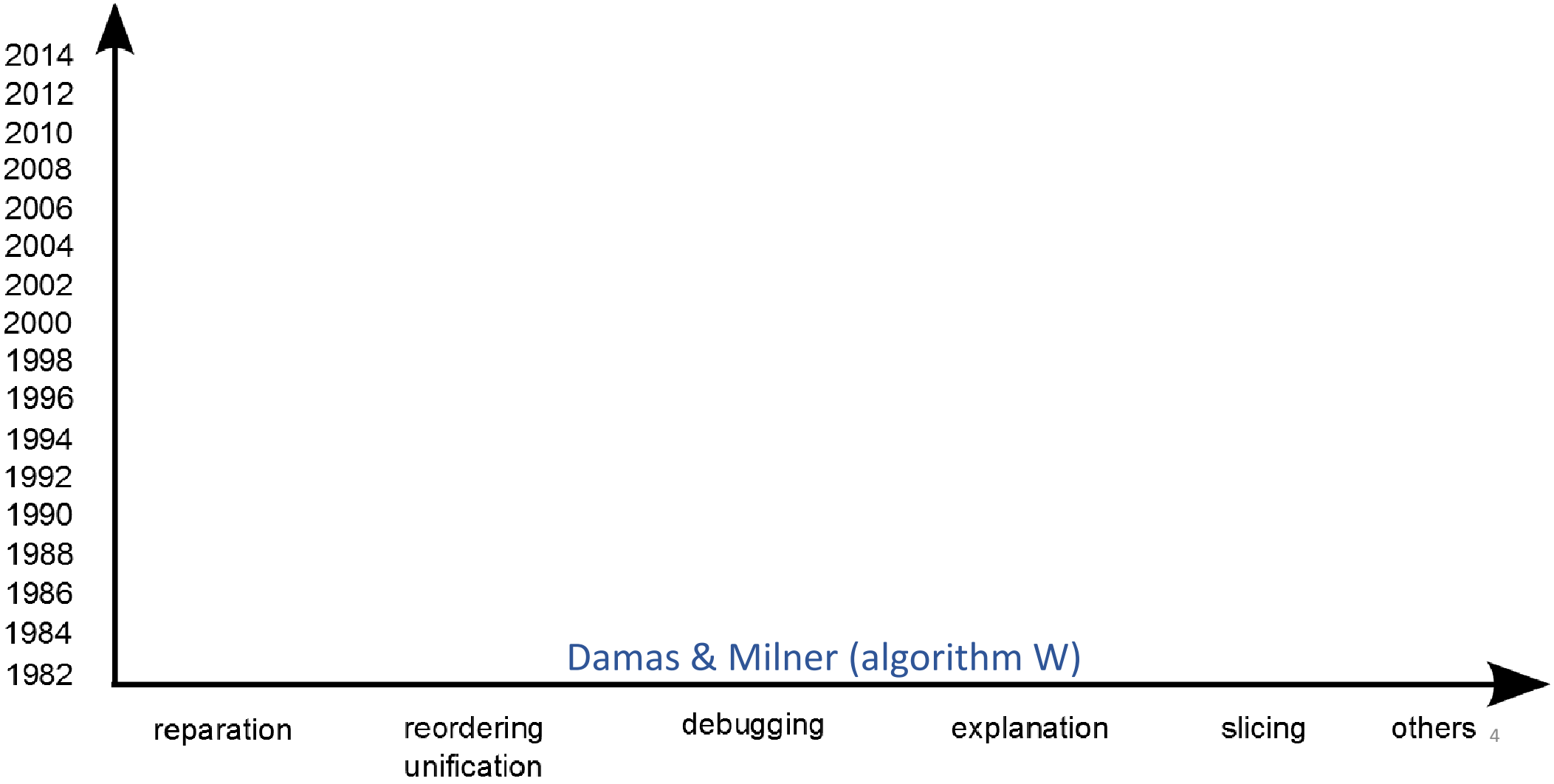
Outline

previous work

Computing potential changes

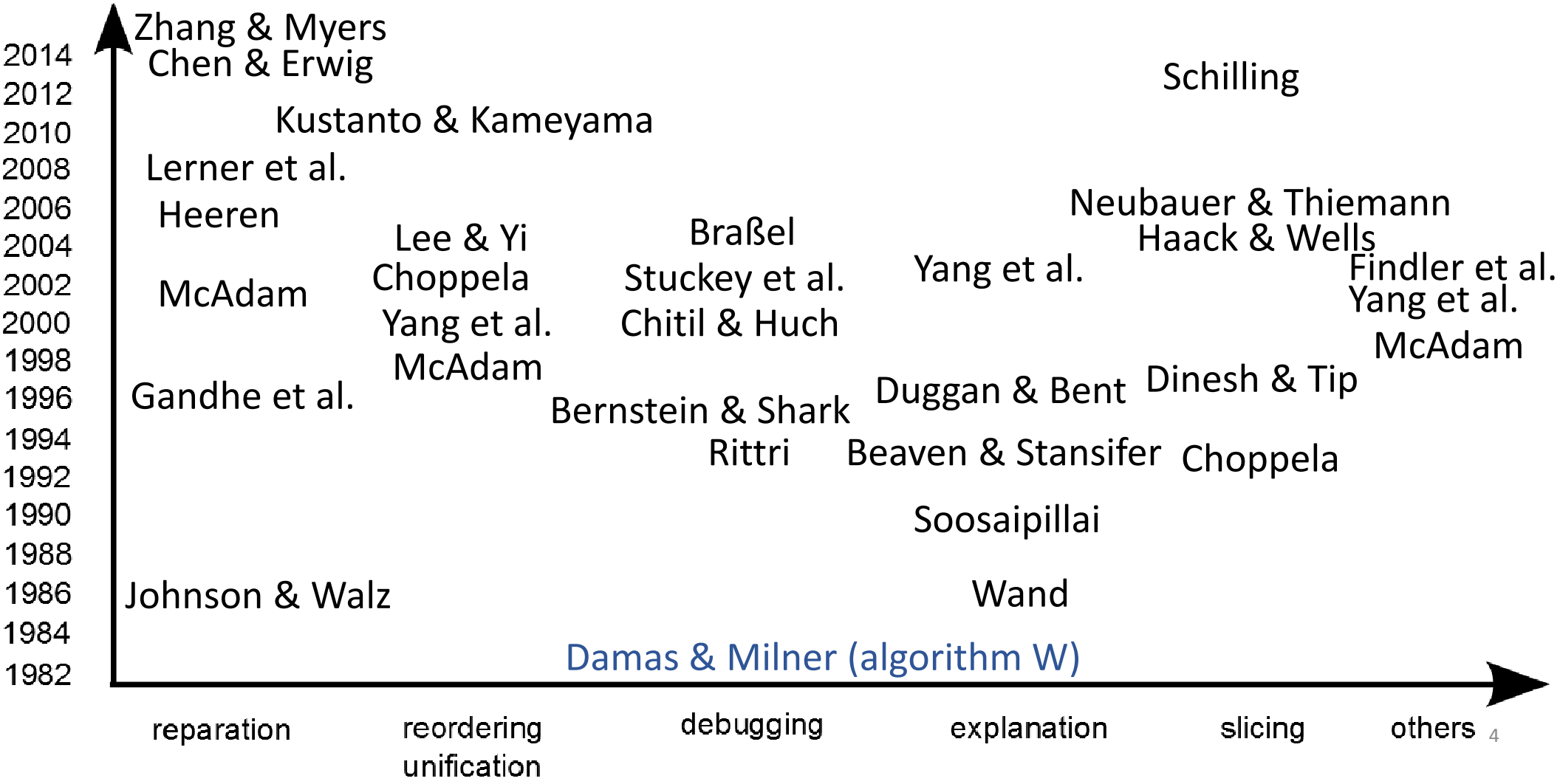


Diagnosing type errors: a retrospect



Diagnosing type errors: a retrospect

Diagnosing type errors: a retrospect



Let’s start simple

```
f = foldl (:) []
```

Let’s start simple

GHC

```
f = foldl (:) []
```

```
Occurs check: cannot construct the infinite type: a0 = [a0]
Expected type: [a0] -> [[a0]] -> [a0]
  Actual type: [a0] -> [[a0]] -> [[a0]]
In the first argument of ‘foldl’, namely ‘(:)’
In the expression: foldl (:) []
```

Let’s start simple

Helium

```
f = foldl (:) []
```

```
Type error in constructor
 expression        : :
   type            : a   -> [a] -> [a]
   expected type   : [b] -> c   -> [b]
 because           : unification would give infinite type
 probable fix      : use ++ instead
```

B. J. Heeren. Top Quality Type Error Messages. PhD thesis, 2005

Let’s start simple

Helium

```
f = foldl (:) []
```

```
Type error in constructor
 expression        : :
   type            : a   -> [a] -> [a]
   expected type   : [b] -> c   -> [b]
 because           : unification would give infinite type
 probable fix      : use ++ instead
```

Result type: `[[a]] -> [a]`
What if the expected type is:
`[a] -> [a]`

B. J. Heeren. Top Quality Type Error Messages. PhD thesis, 2005

Let's start simple

```
f = foldl (:) []
```

CF typing

```
f has type errors.

Some possible fixes
    1 change (:) from a -> [a] -> [a] to [a] -> b -> [a].
    2 change foldl
          from type (a -> b -> a) -> a -> [b] -> a
          to type (a -> [a] -> [a]) -> [b] -> c.
More fixes?
```

Sheng Chen and Martin Erwig. Counter-Factual Typing for Debugging Type Errors. POPL 2014

Let's start simple

```
f = foldl (:) []
```

CF typing

```
f has type errors.

Some possible fixes
    1 change (:) from a -> [a] -> [a] to [a] -> b -> [a].
    2 change foldl
          from type (a -> b -> a) -> a -> [b] -> a
          to type (a -> [a] -> [a]) -> [b] -> c.
More fixes?
```

Not very informative

Sheng Chen and Martin Erwig. Counter-Factual Typing for Debugging Type Errors. POPL 2014

The problem

There is a misalignment of type error debugger's goal and user's goal

The solution

Solicit user's goal about result type and suggest changes that match that goal

Guided type debugging in action

```
f = foldl (:) []
```

```
What is the expected type of f?
[a] -> [a]

Potential fixes:
1  Change (:) to (flip (:)).
2  change foldl
     from type (a -> b -> a) -> a -> [b] -> a
     to type (a -> [a] -> [a]) -> b -> [a] -> [a]
There are no other one-change fixes.
Show two-change fixes? (y/n)
```

Guided type debugging in action

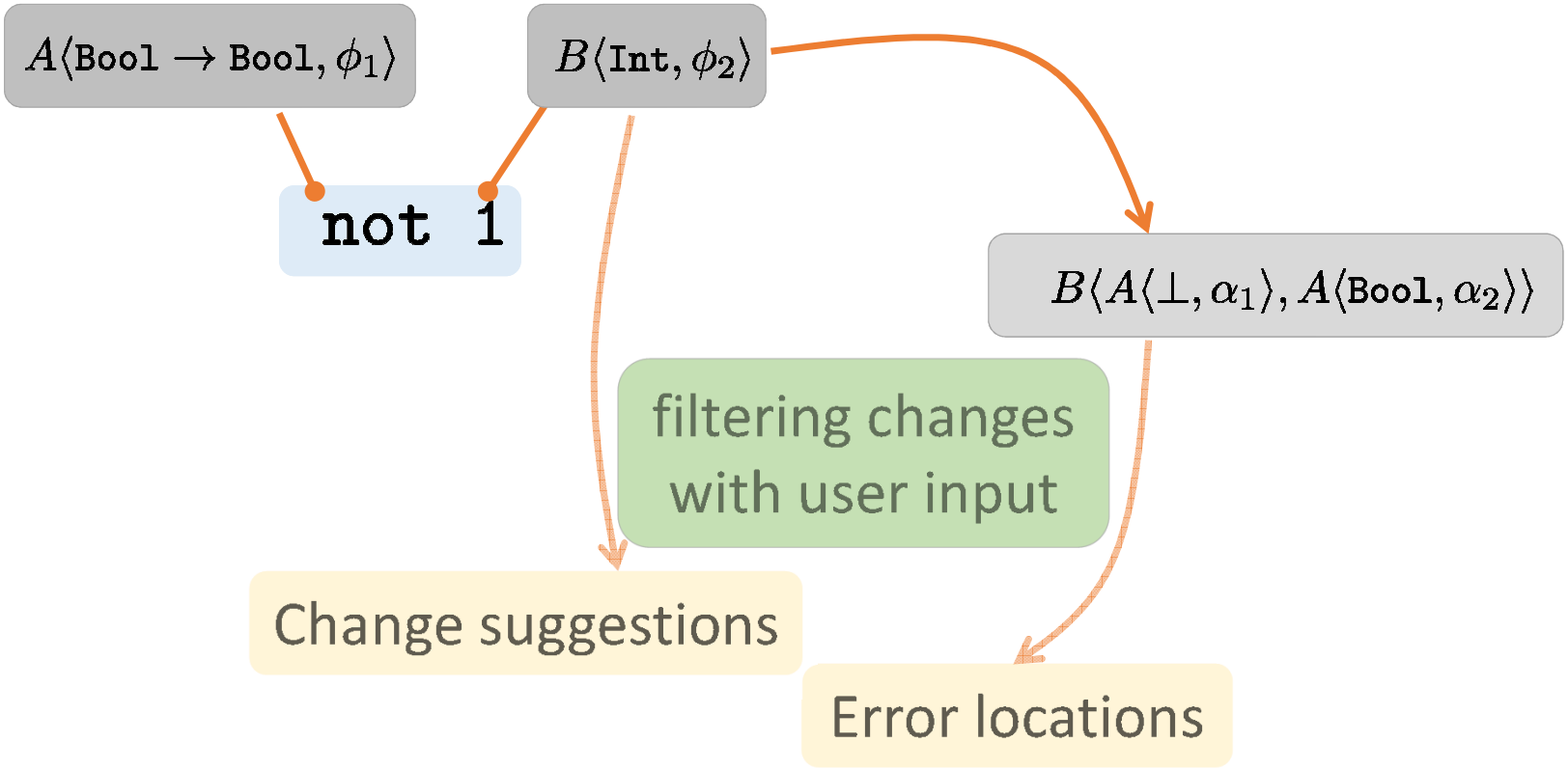
```
f = foldl (:) []
```

```
What is the expected type of f?
[[a]] -> [a]

Potential fixes:
1  Change (:)
     from type a -> [a] -> [a]
     to type [a] -> [a] -> [a]
2  change foldl
     from type (a -> b -> a) -> a -> [b] -> a
     to type (a -> [a] -> [a]) -> b -> [[a]] -> [a]
There are no other one-change fixes.
Show two-change fixes? (y/n)
```

Outline

previous work

Computing potential changes



Examine each location

What type should “(:)” have?

```
fold f z []     = (:) z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

Examine each location

What type should “(:)” have?

```
fold f z []     = uv z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

Examine each location

What type should “(:)” have?

$$(\tau, \theta) = infer(\Gamma \cup \{(\texttt{uv}, \alpha)\}, \texttt{palin})$$

```
fold f z []     = uv z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

Examine each location

What type should “(:)” have?

$(\tau, \theta) = infer(\Gamma \cup \{(\mathtt{uv}, \alpha)\}, \mathtt{palin})$

fail

```
fold f z []     = uv z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

(:) is not, or not the only, cause of the type error

Examine each location

What type should “(:)” have?

$(\tau, \theta) = infer(\Gamma \cup \{(\mathtt{uv}, \alpha)\}, \mathtt{palin})$

fail

succeed

(:) should have the type $\theta(\alpha)$

(:) is not, or not the only, cause of the type error

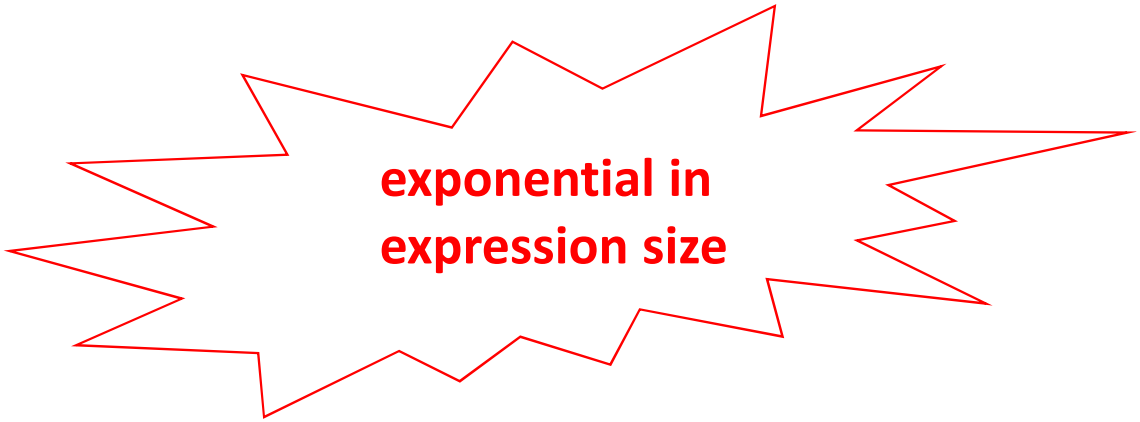
```
fold f z []     = uv z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

Combatting exponential blow-up

How to find changes for all locations ?

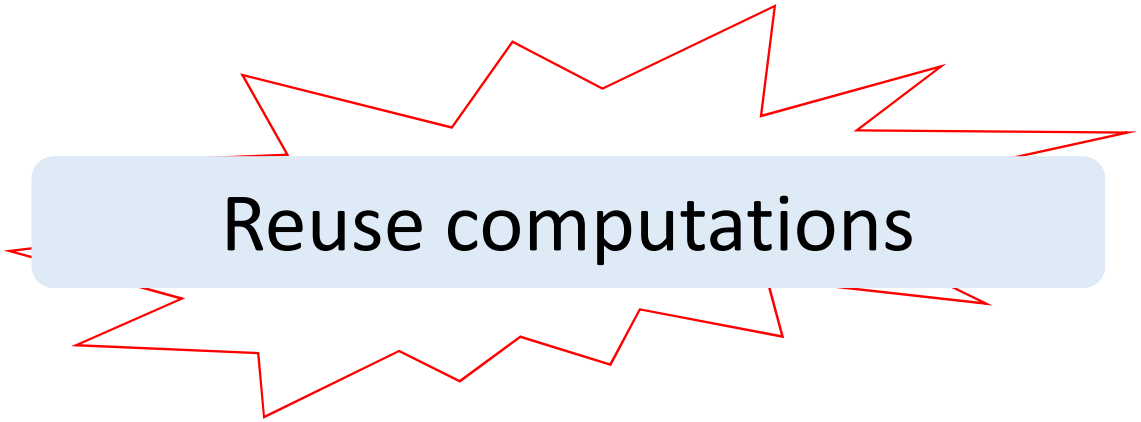
Combatting exponential blow-up

How to find changes for all locations ?



Combatting exponential blow-up

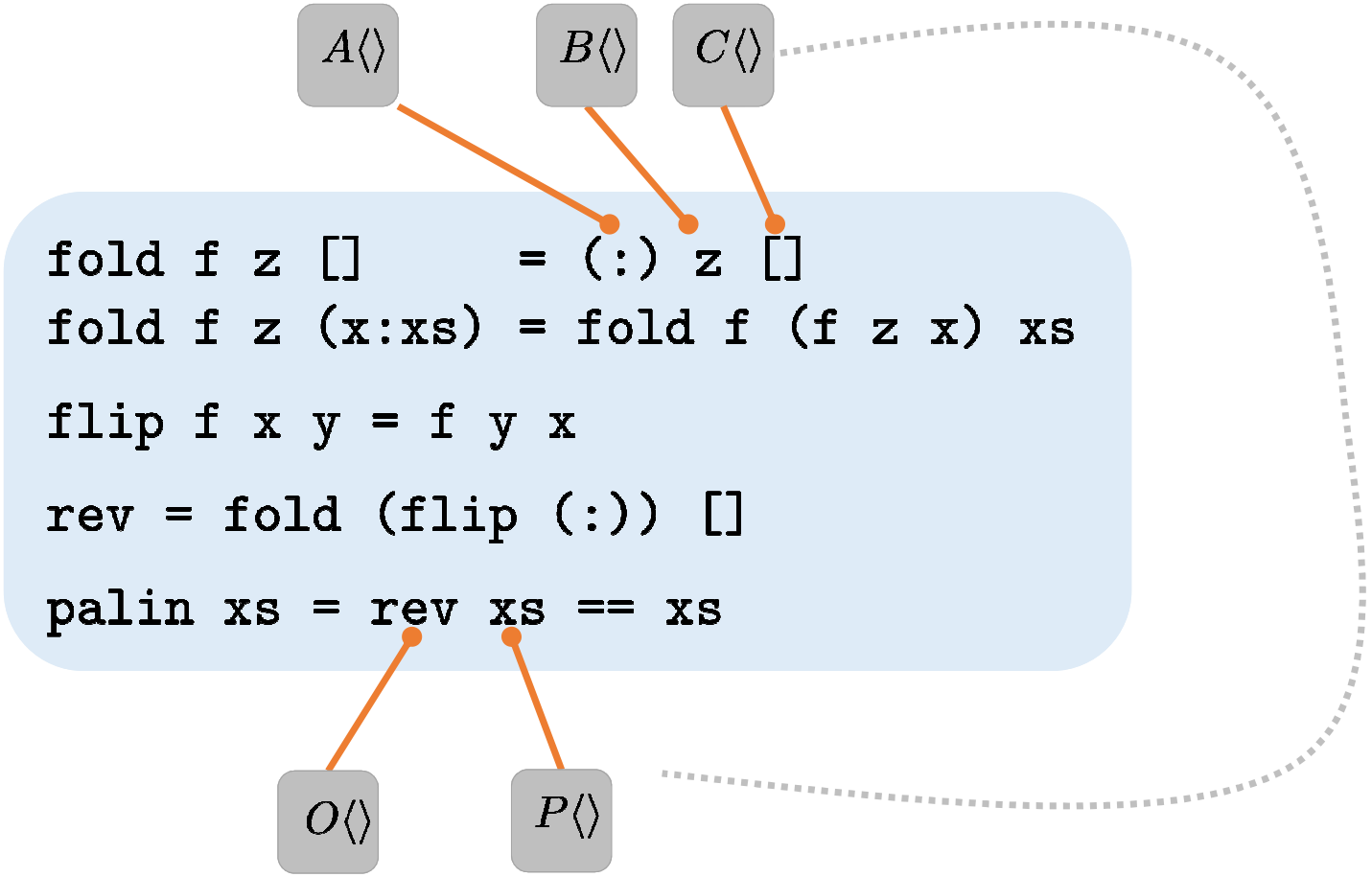
How to find changes for all locations ?



Reuse computations with choice types

```
fold f z []     = (:) z []
fold f z (x:xs) = fold f (f z x) xs

flip f x y = f y x

rev = fold (flip (:)) []

palin xs = rev xs == xs
```

Reuse computations with choice types



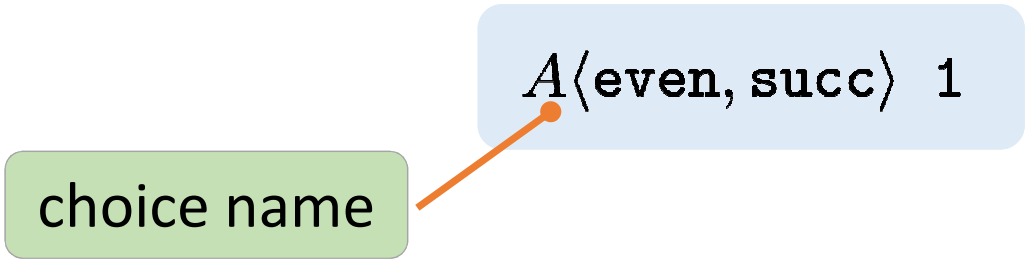
Choice expressions and choice types

Choices encode uncertainties

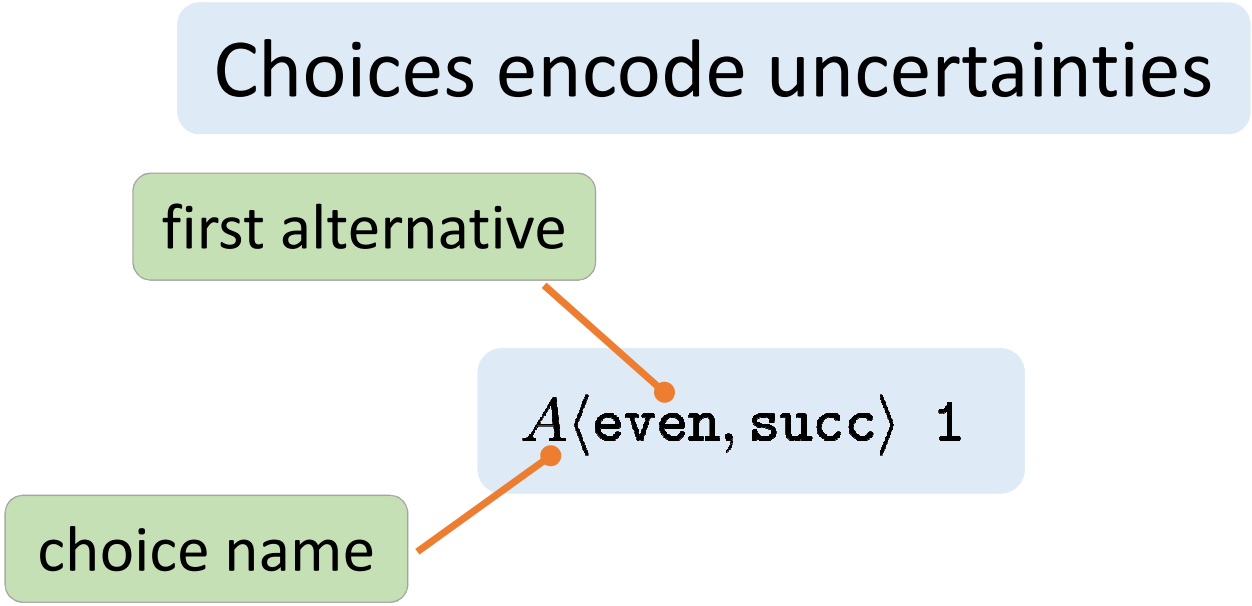
$A\langle \mathtt{even}, \mathtt{succ} \rangle\ \mathtt{1}$

Choice expressions and choice types

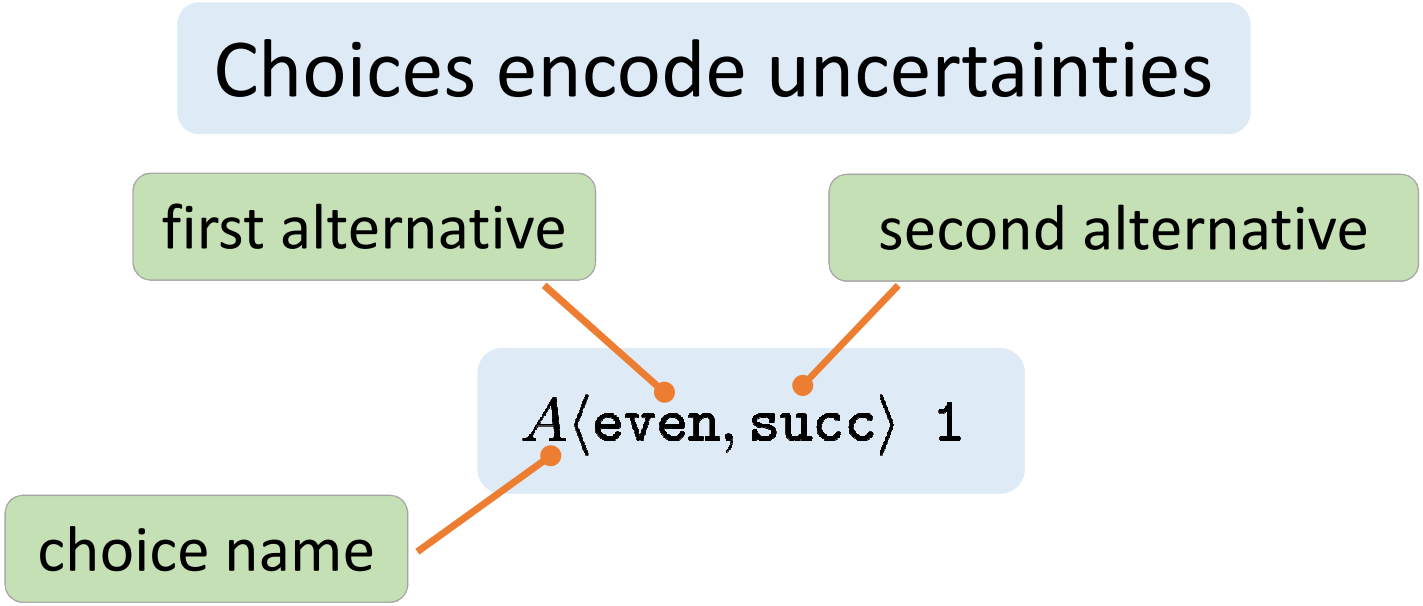
Choices encode uncertainties



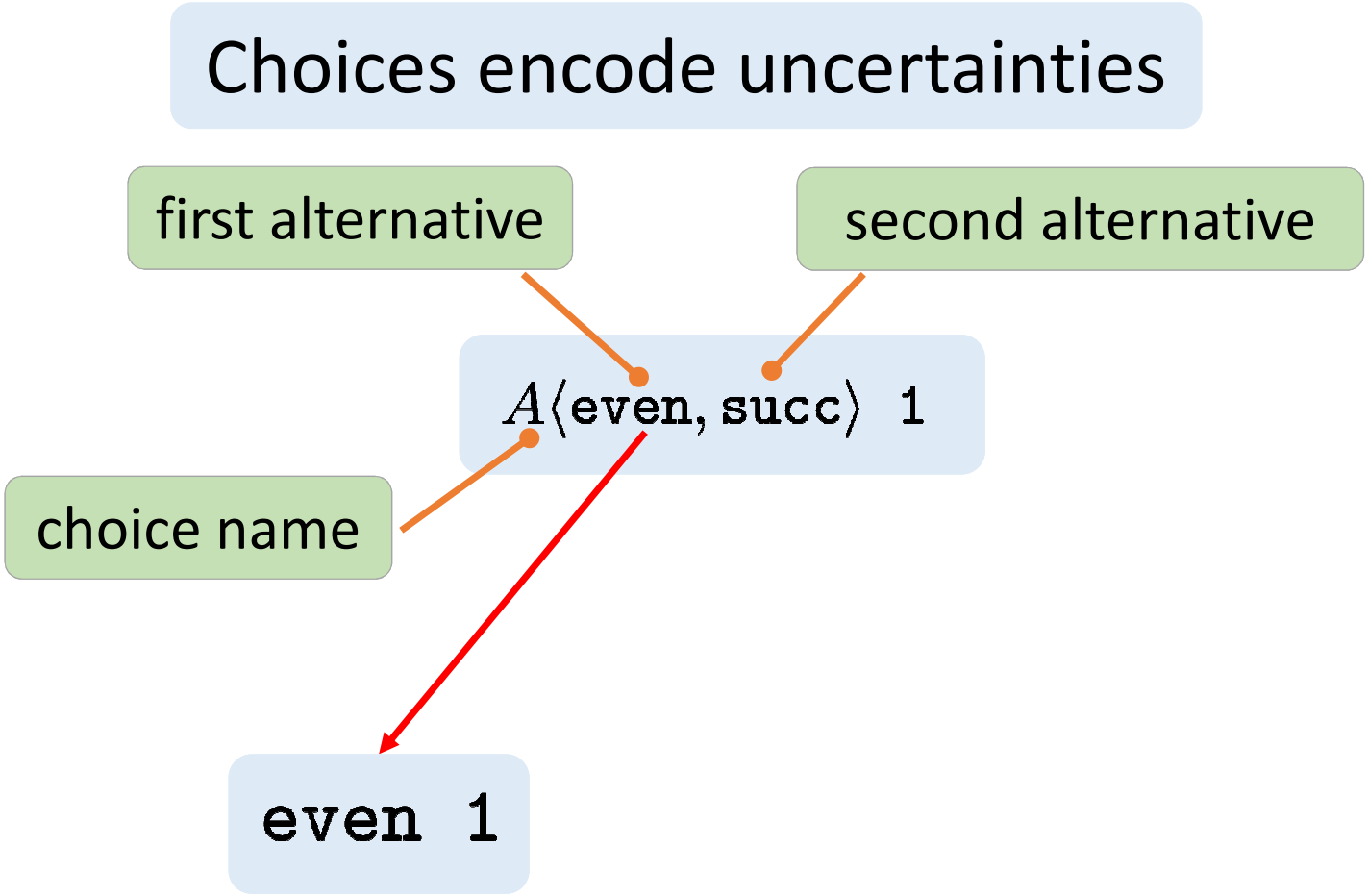
Choice expressions and choice types



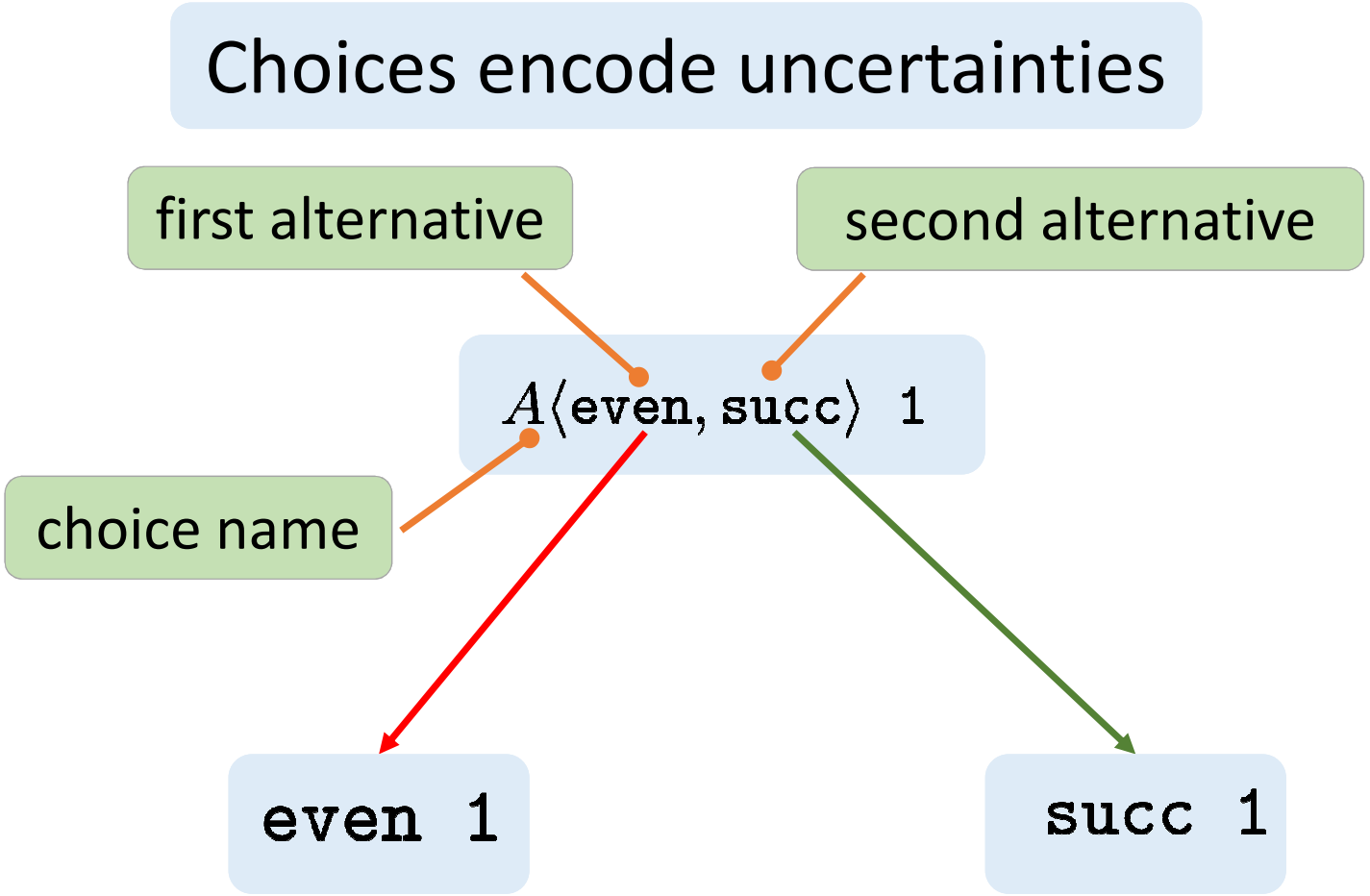
Choice expressions and choice types



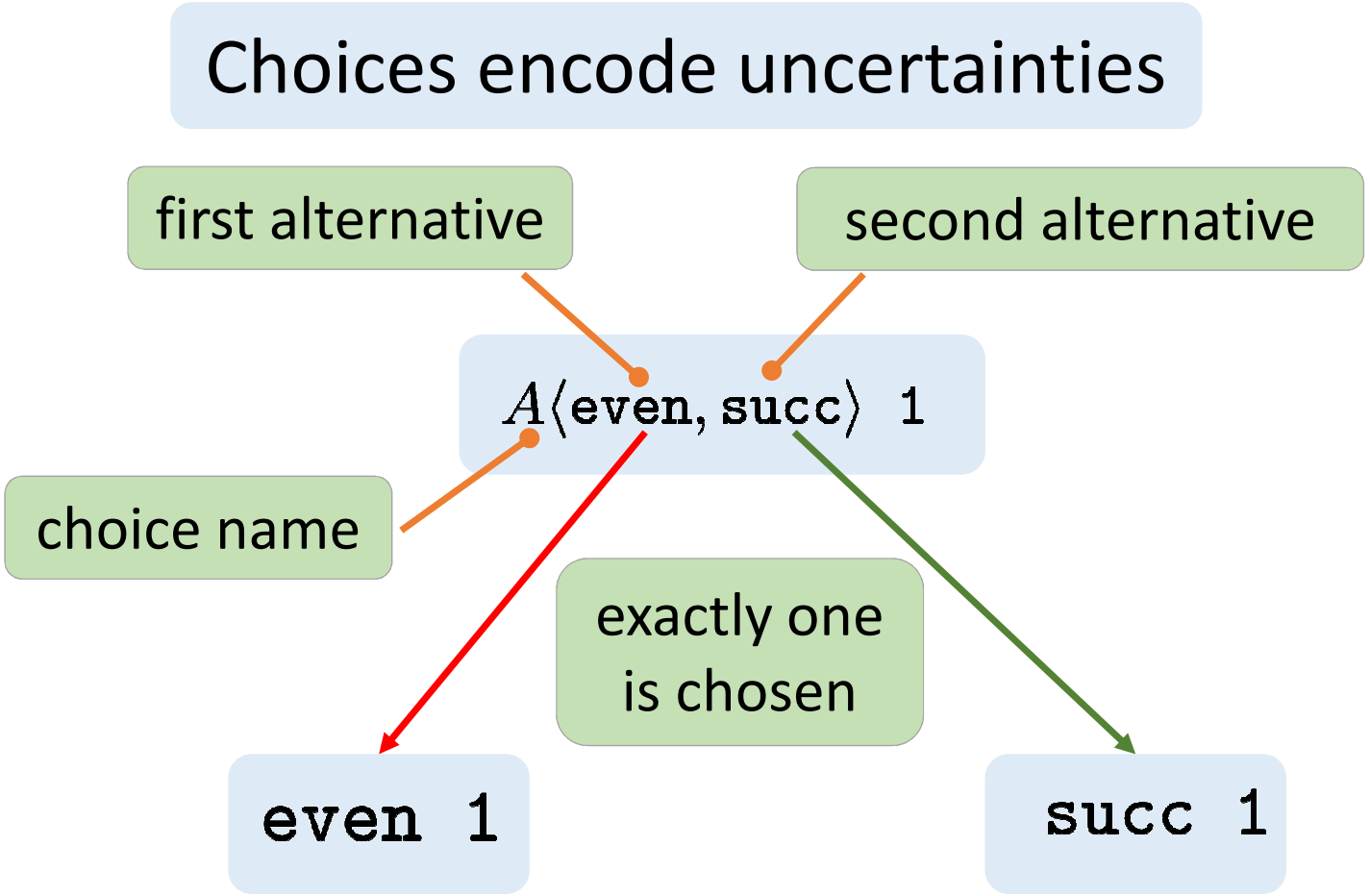
Choice expressions and choice types



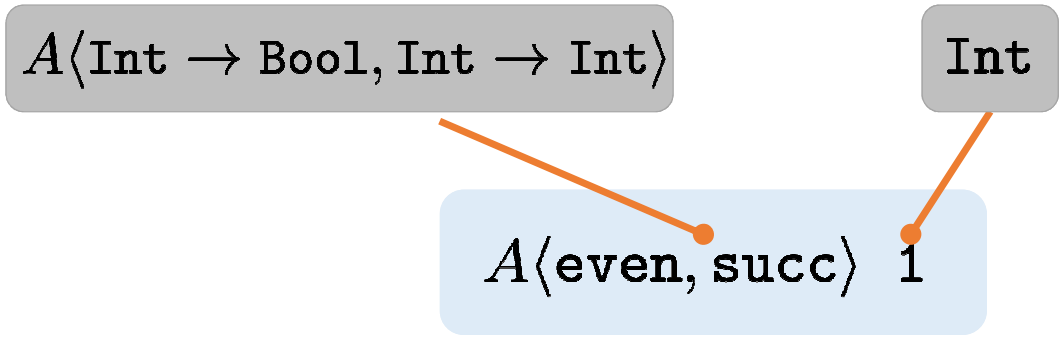
Choice expressions and choice types



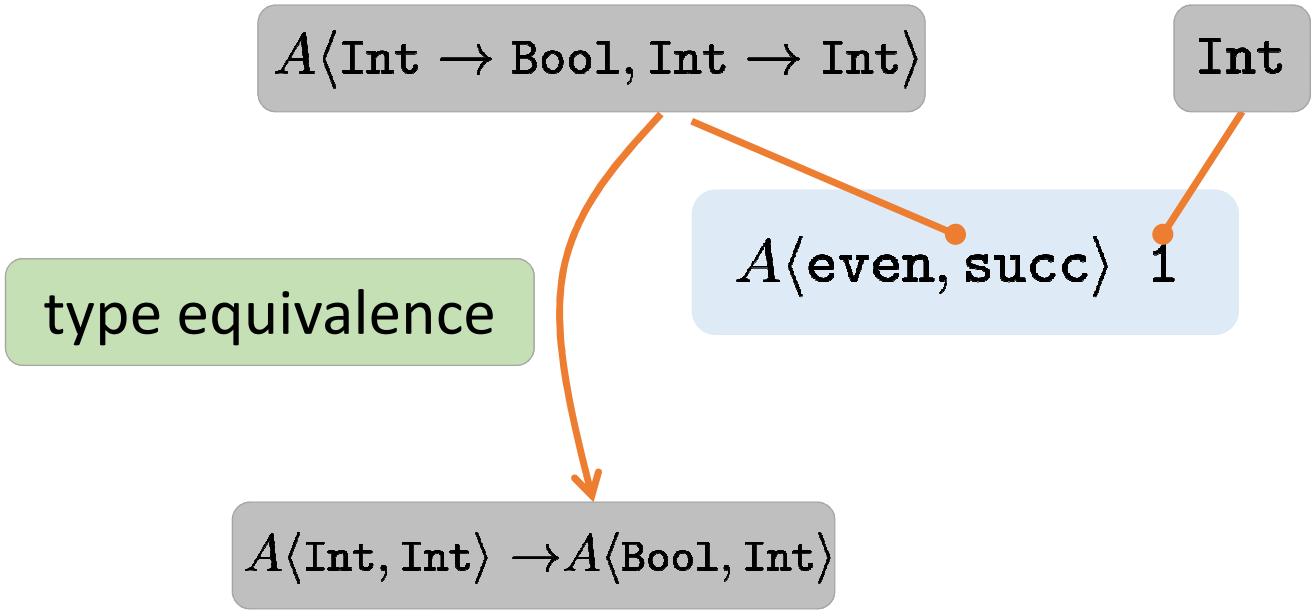
Choice expressions and choice types



Choice expressions and choice types

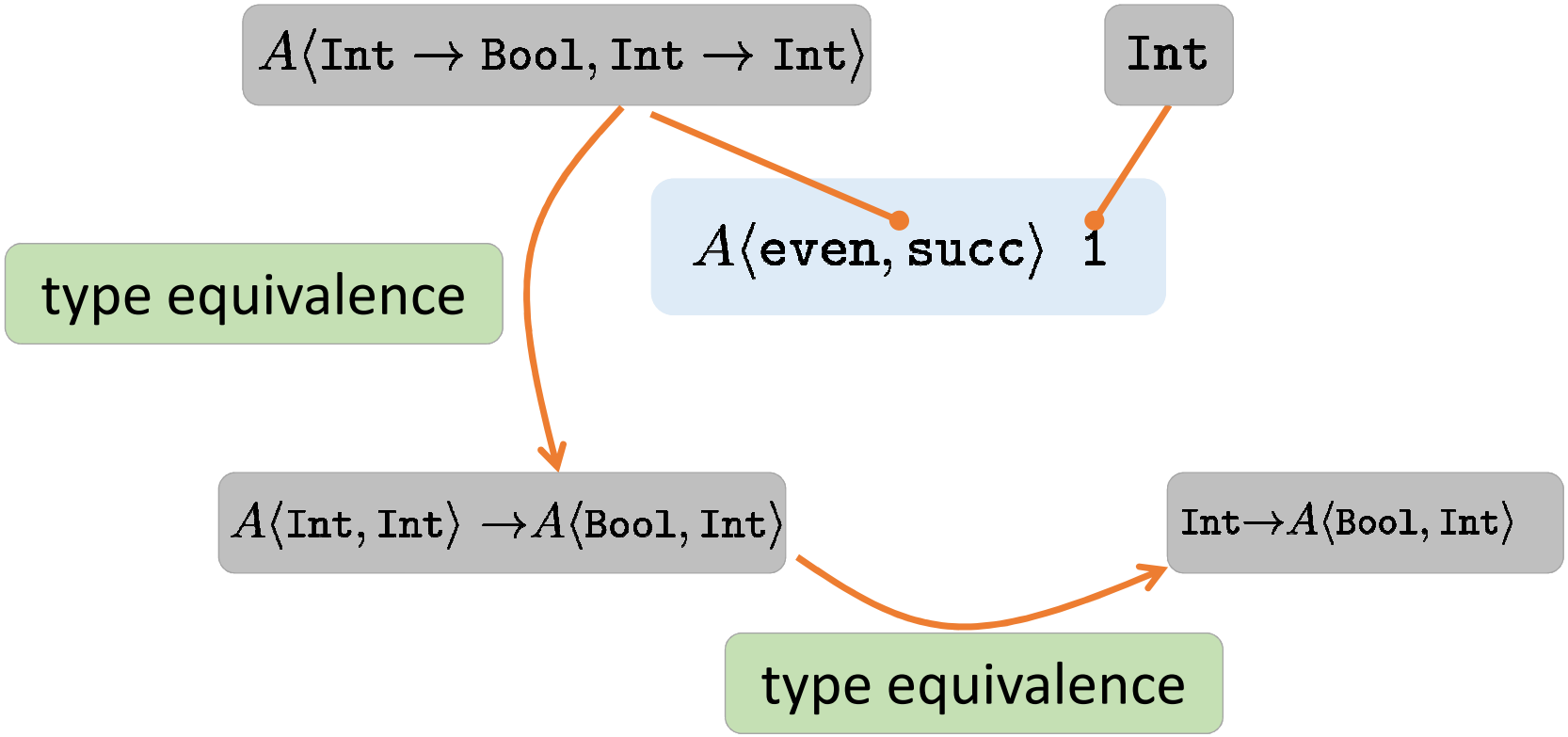


Choice expressions and choice types



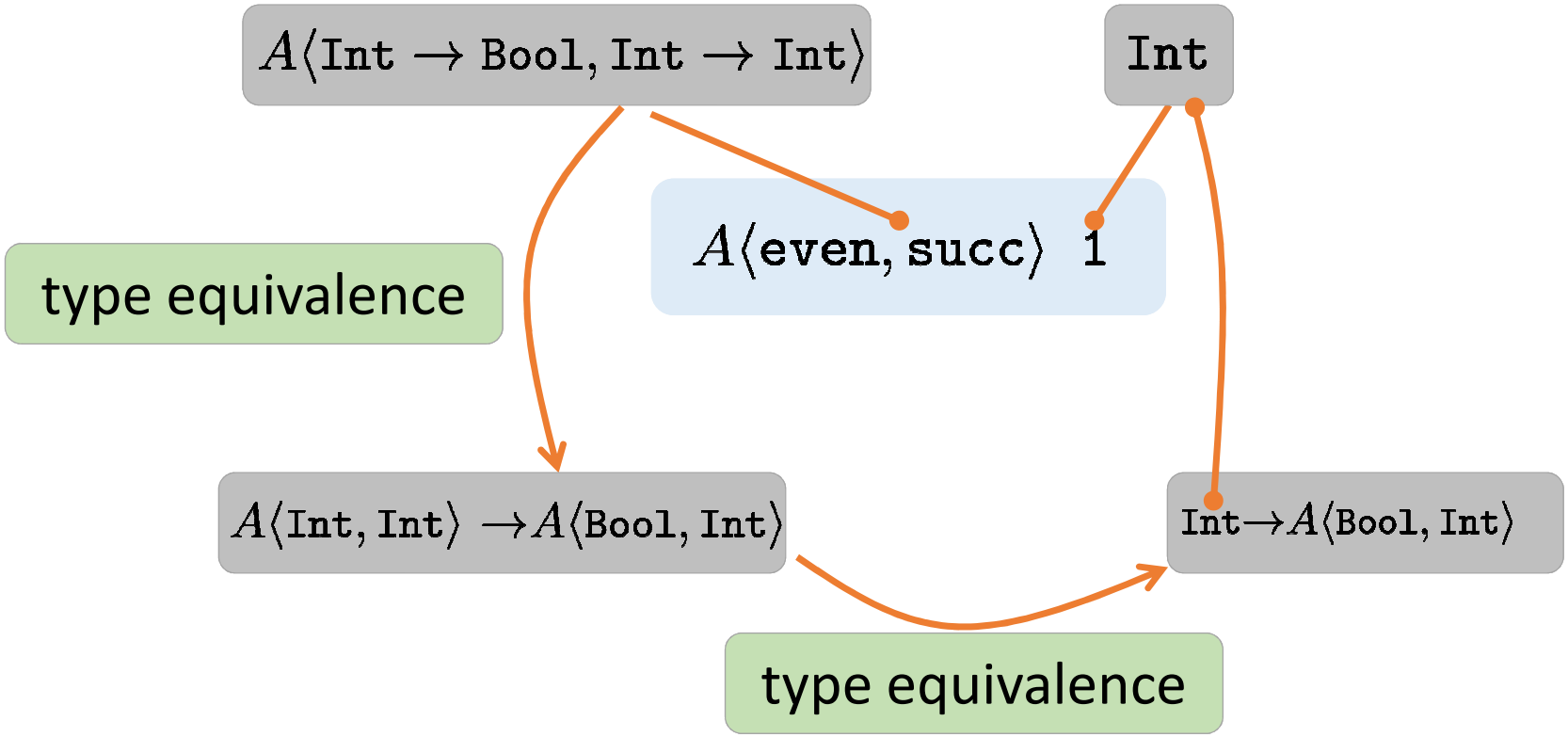
Sheng Chen, Martin Erwig and Eric Walkingshaw. Extending Type Inference to Variational Programs. TOPLAS 2014

Choice expressions and choice types



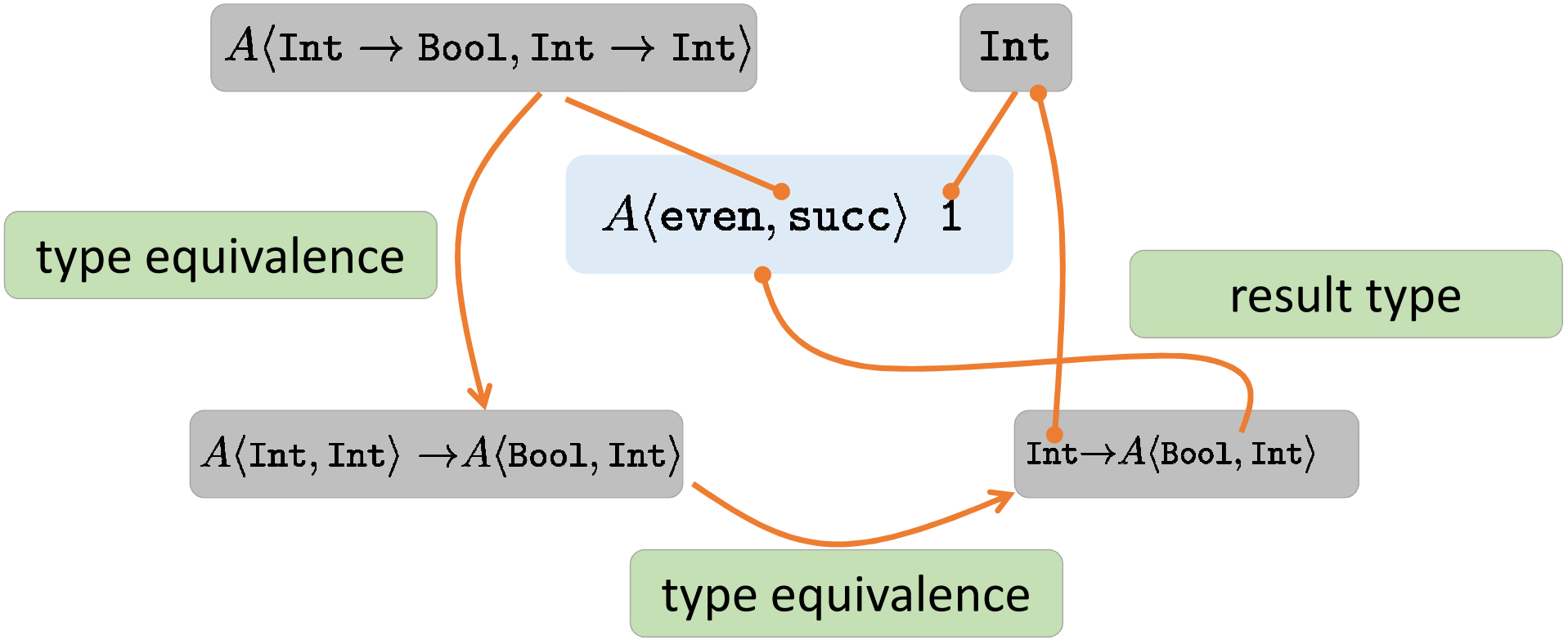
Sheng Chen, Martin Erwig and Eric Walkingshaw. Extending Type Inference to Variational Programs. TOPLAS 2014

Choice expressions and choice types



Sheng Chen, Martin Erwig and Eric Walkingshaw. Extending Type Inference to Variational Programs. TOPLAS 2014

Choice expressions and choice types

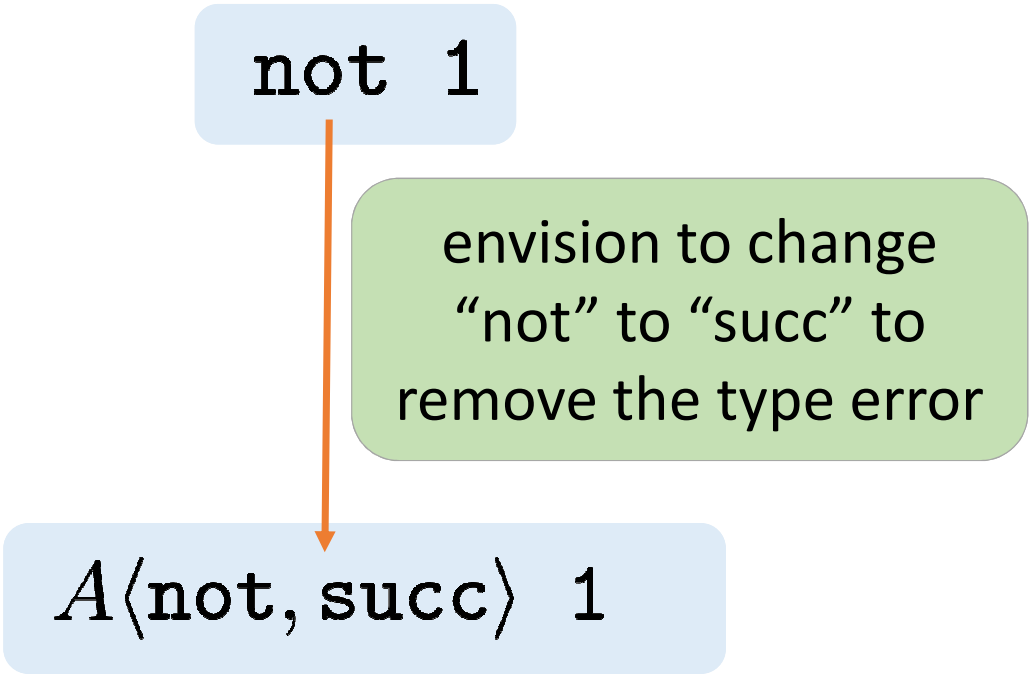


Sheng Chen, Martin Erwig and Eric Walkingshaw. Extending Type Inference to Variational Programs. TOPLAS 2014

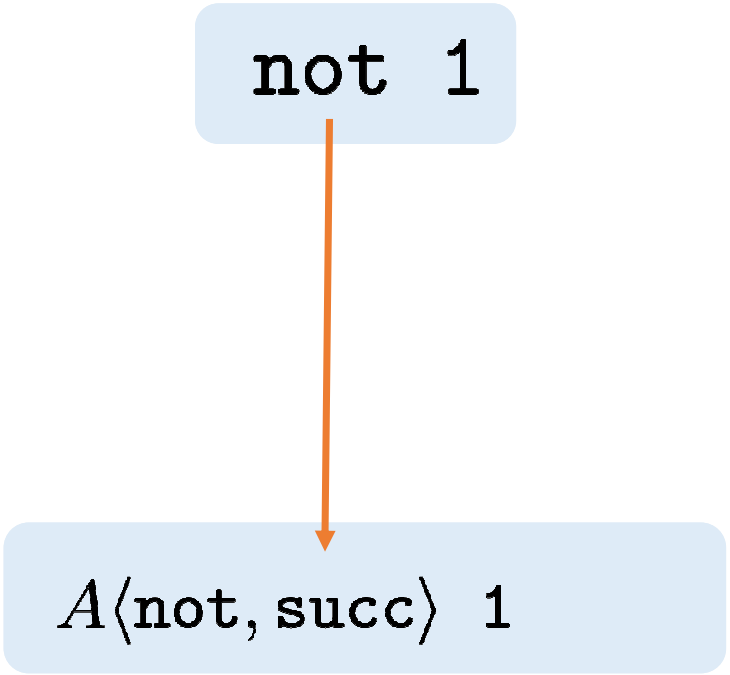
Remedy ill-typed expressions

```
not 1
```

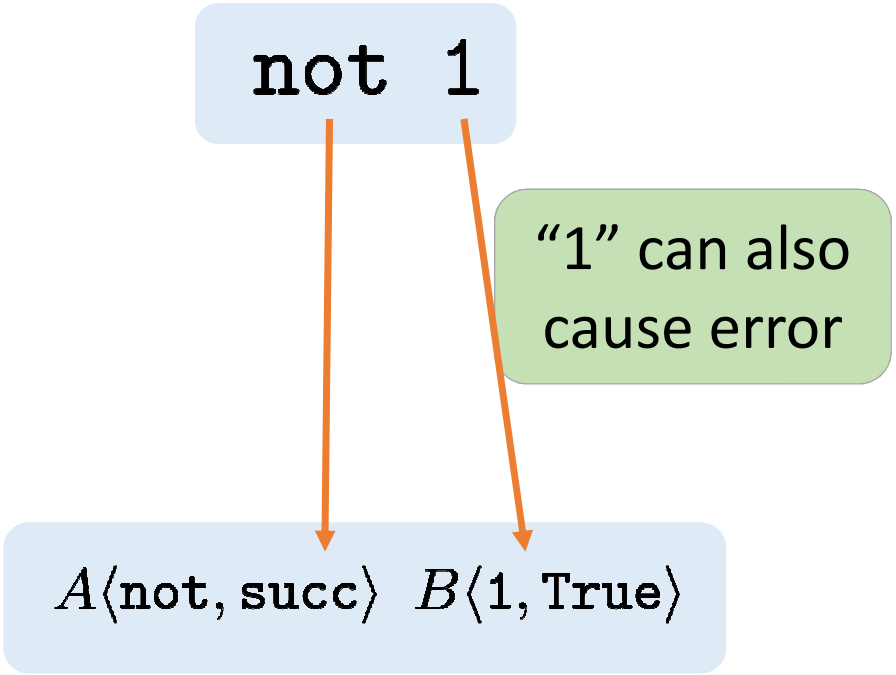
Remedy ill-typed expressions



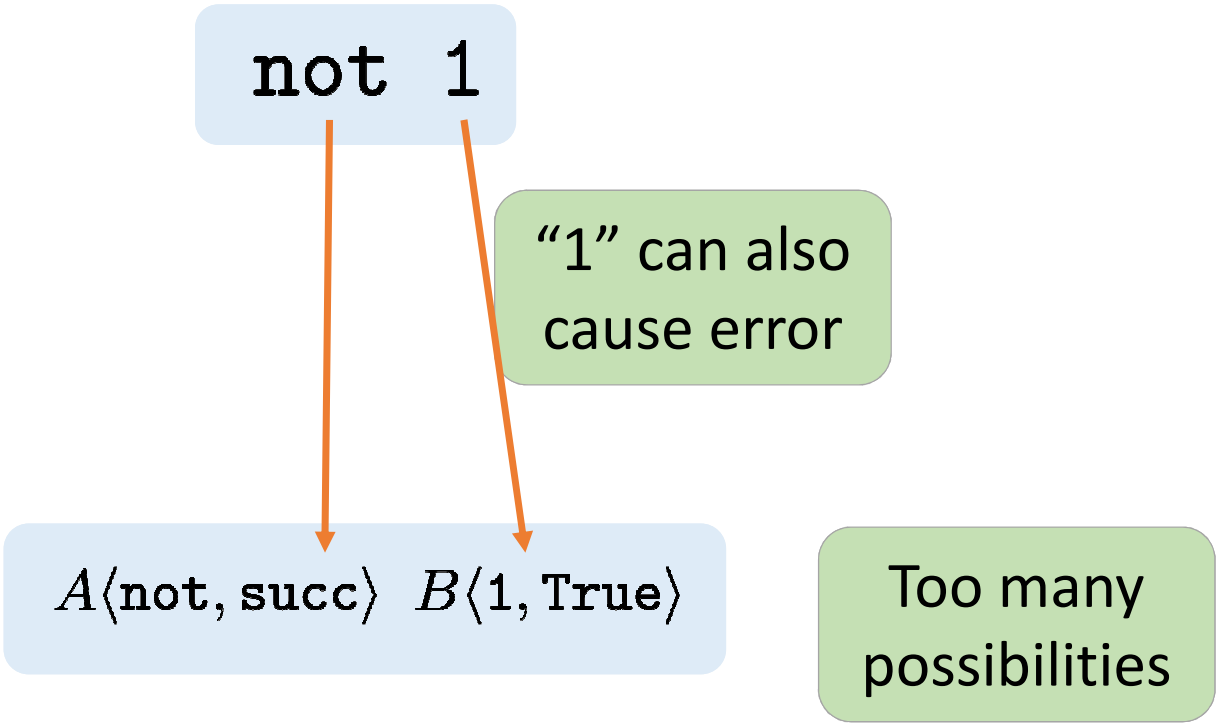
Do it systematically



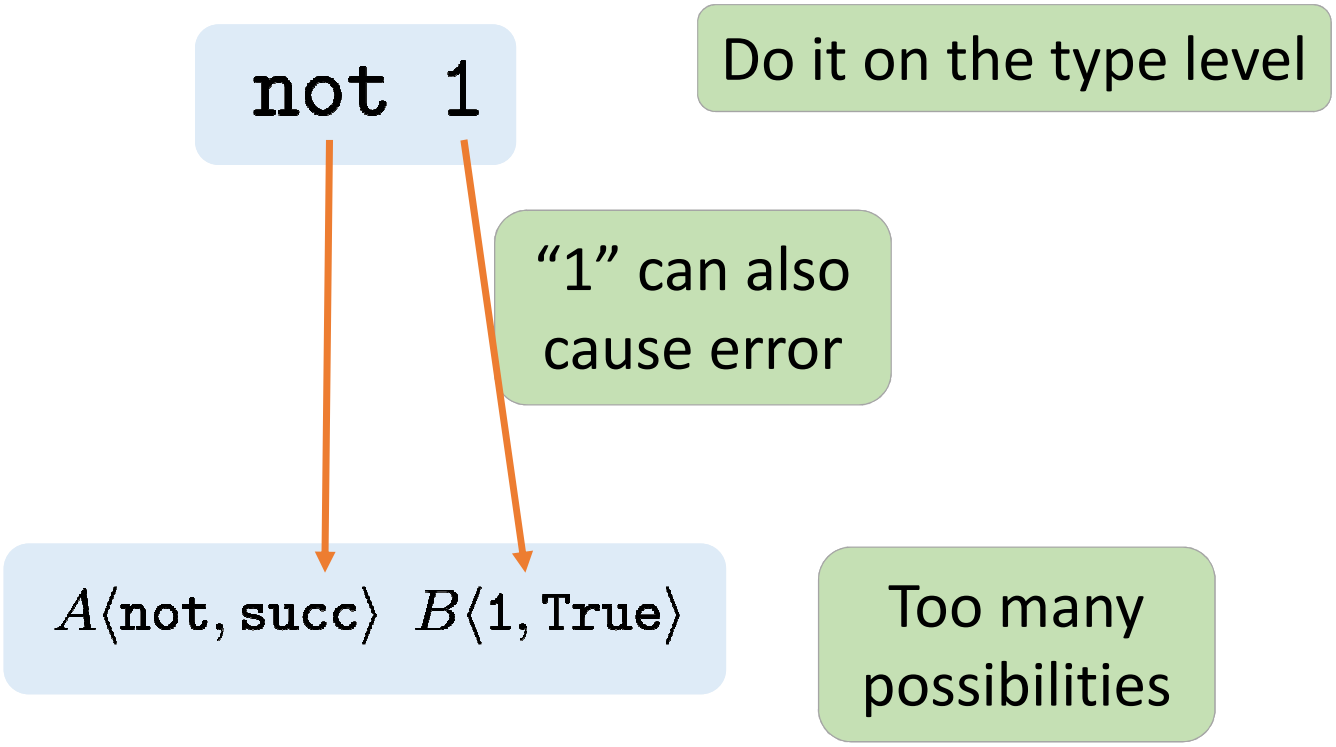
Do it systematically



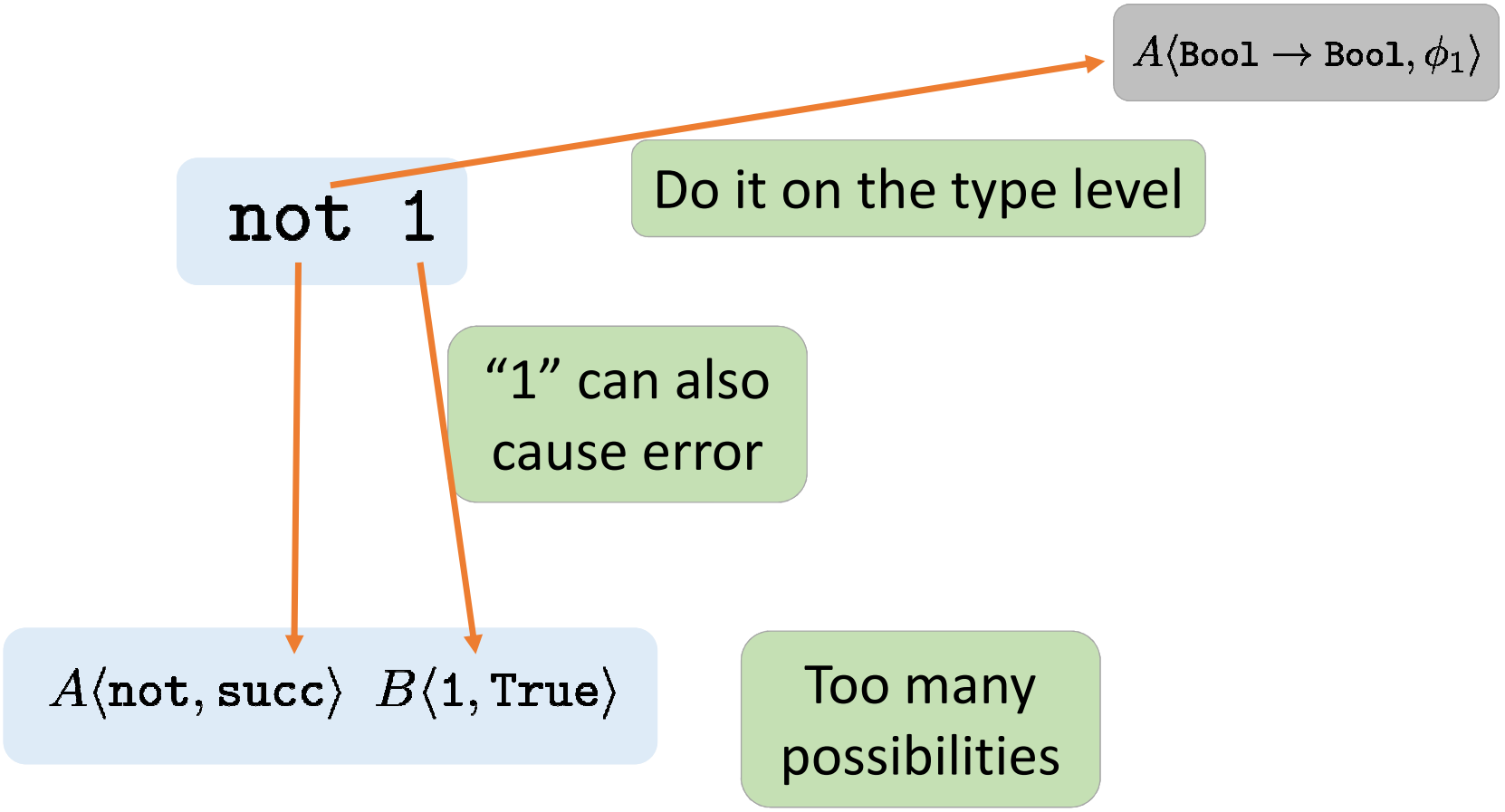
Do it systematically



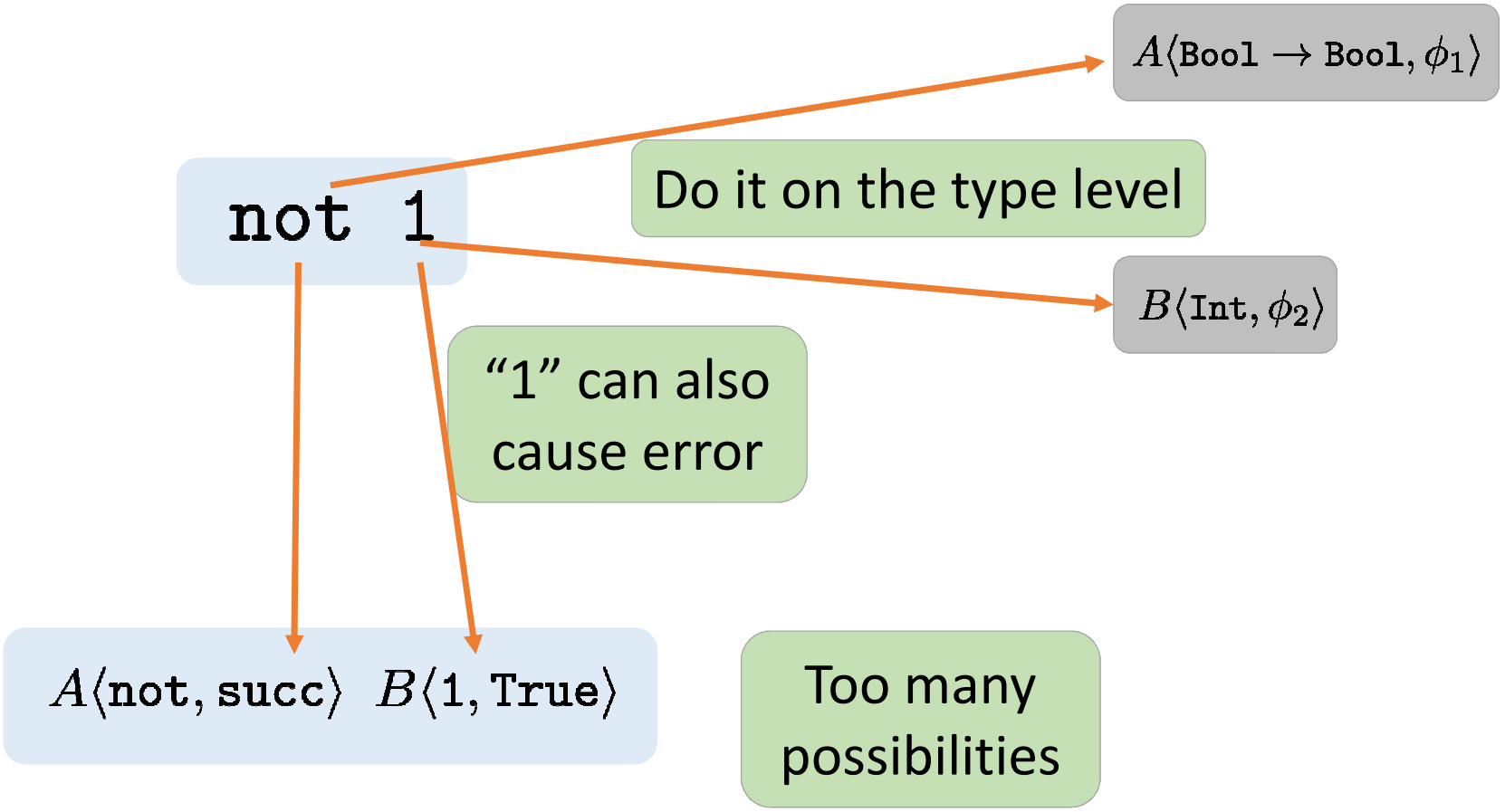
Do it systematically



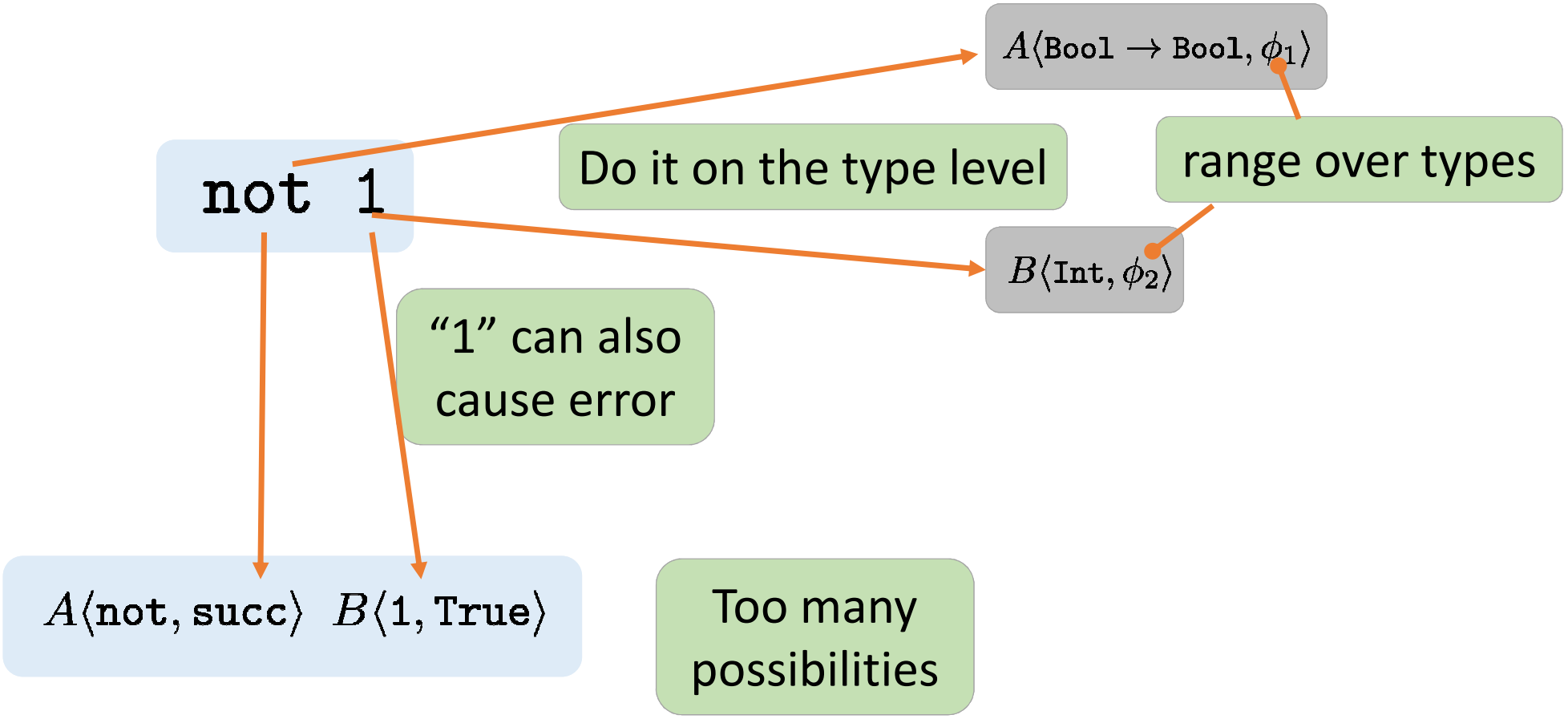
Do it systematically



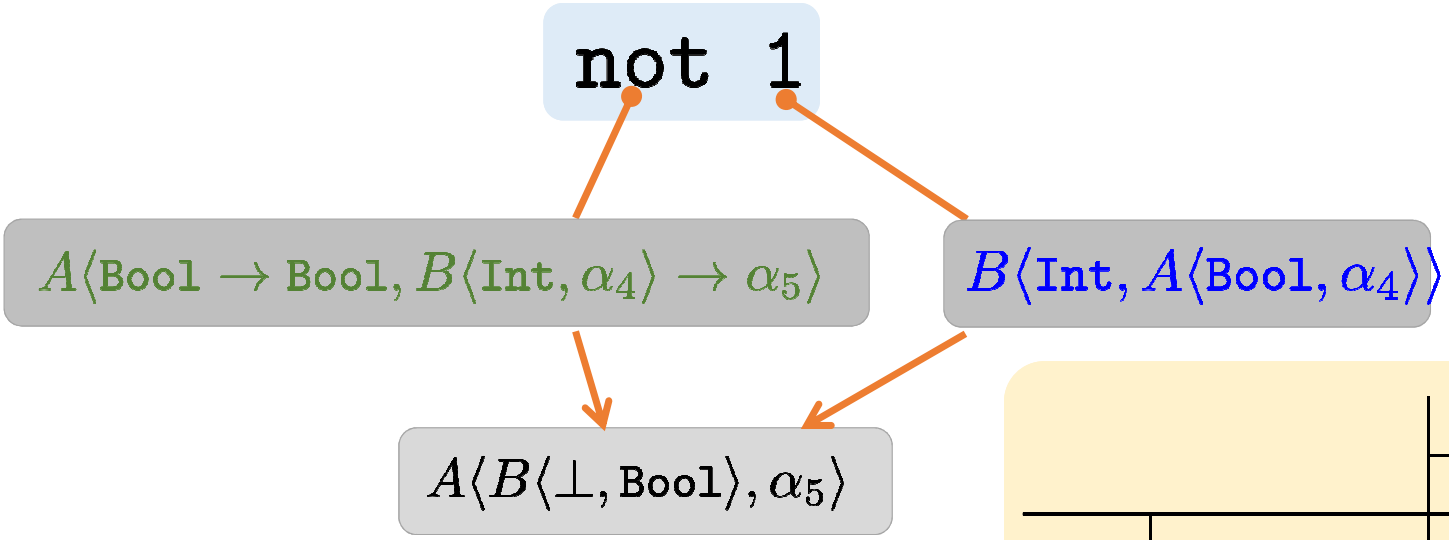
Do it systematically



Do it systematically

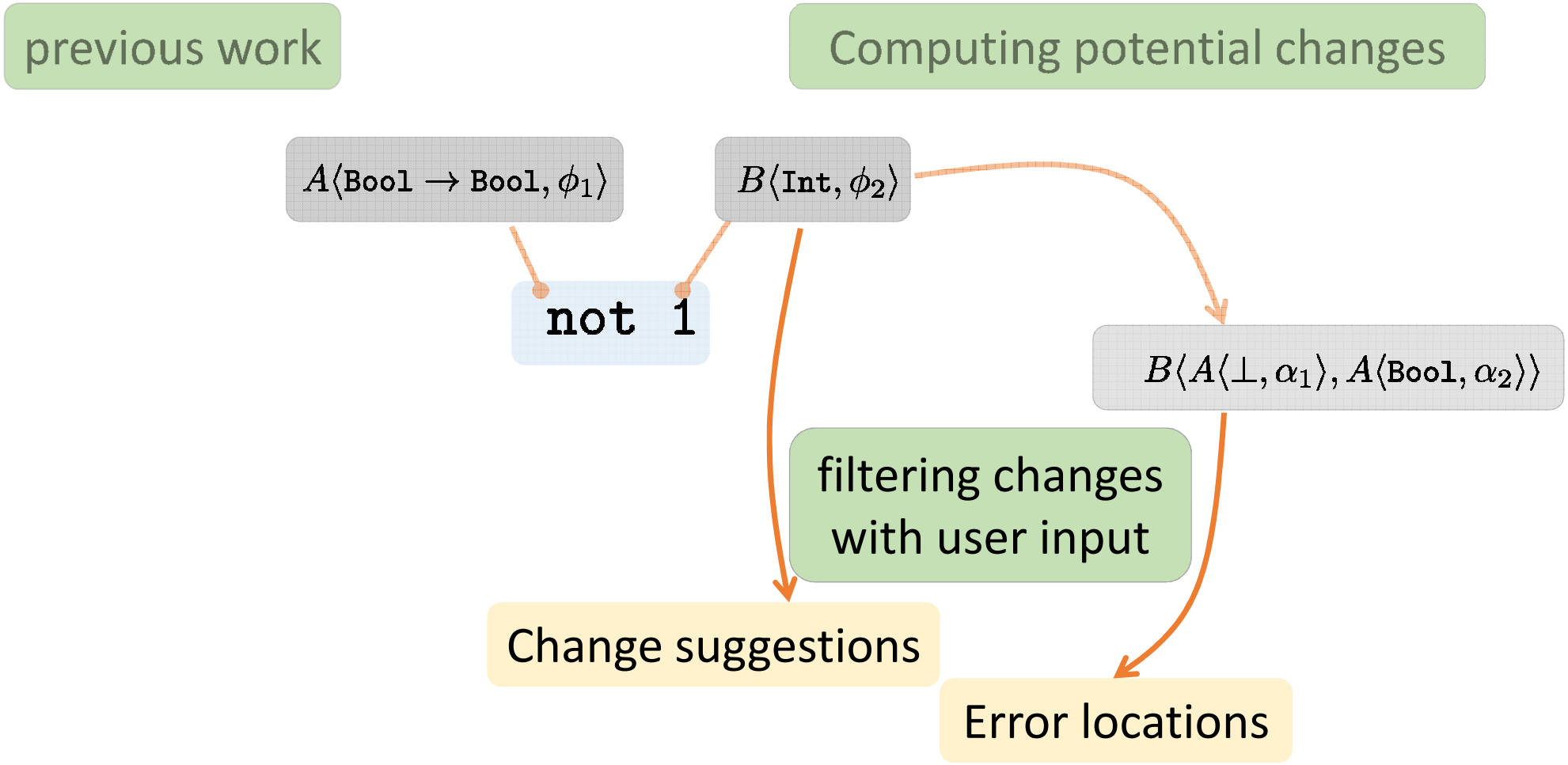


Computing all potential fixes



		1	
		no change	*change*
not	*no change*	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Int}$ $\bot$	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Bool}$ $\texttt{Bool}$
	change	$\texttt{Int} \to \alpha_5$ $\texttt{Int}$ α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Outline


previous work
Computing potential changes
$A\langle \texttt{Bool} \to \texttt{Bool}, \phi_1 \rangle$
$B\langle \texttt{Int}, \phi_2 \rangle$
not 1
$B\langle A\langle \bot, \alpha_1 \rangle, A\langle \texttt{Bool}, \alpha_2 \rangle \rangle$
filtering changes with user input
Change suggestions
Error locations

Combat another exponential problem

```
not 1
```

		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Combat another exponential problem

```
not 1
```

Given a user input, we need to examine each fix to filter out correct fixes

		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Combat another exponential problem

```
not 1
```

Given a user input, we need to examine each fix to filter out correct fixes

		1	
		no change	*change*

`not`	*no change*	`Bool` → `Bool`
		`Int`
		⊥
			`Bool` → `Bool`
			`Bool`
			`Bool`

	change	`Int` → α_5 `Int` α_5	α_4 → α_5 α_4 α_5

exponential in expression size

Combat another exponential problem

```
not 1
```

Given a user input, we need to examine each fix to filter out correct fixes

		1	
		no change	*change*

`not`	*no change*	`Bool` → `Bool`
		`Int`
		⊥
			`Bool` → `Bool`
			`Bool`
			`Bool`

	change	`Int` → α_5 `Int` α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Order fixes

Type change lattice

```
not 1
```

$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$

Type change lattice

```
not 1
```

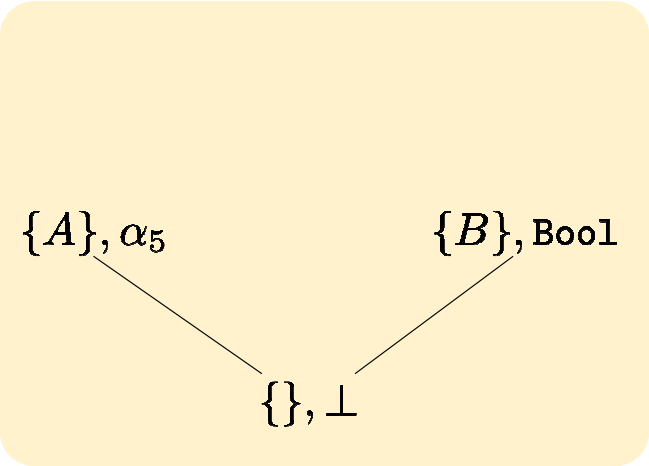
$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$

$\{\}, \bot$

Type change lattice

```
not 1
```

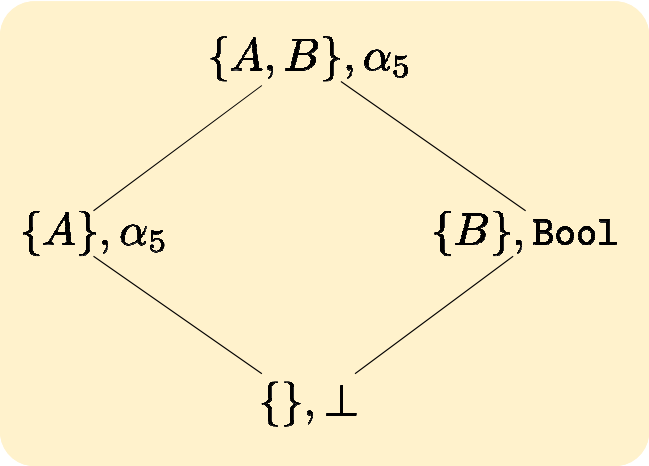
$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$



Type change lattice

`not 1`

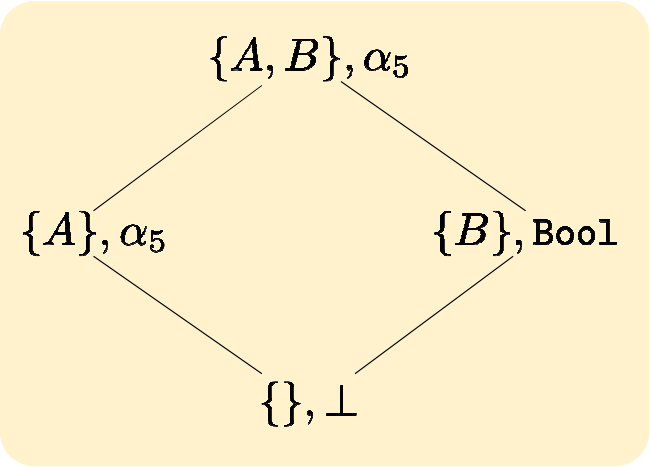
$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$



Type change lattice

```
not 1
```

$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$



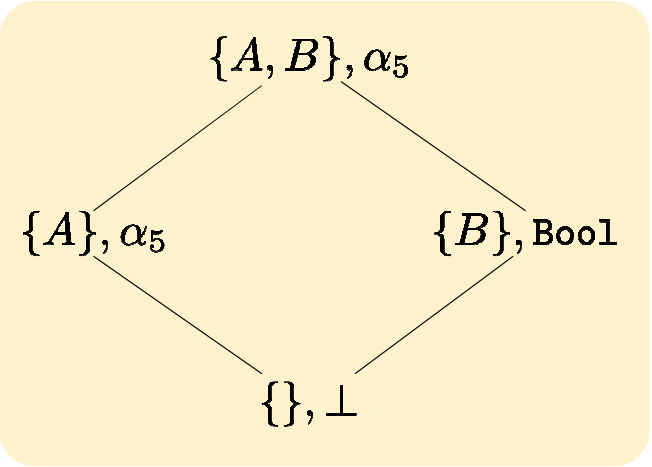
Size of TCL can be exponential in the size of expression

Type change lattice

`not 1`

$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$

More general types



Size of TCL can be exponential in the size of expression

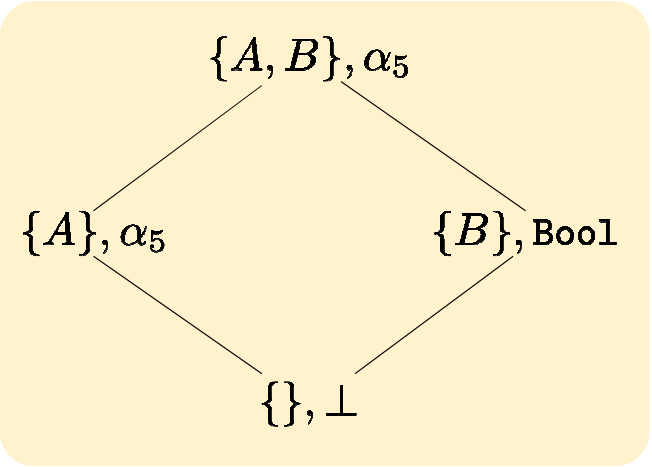
Type change lattice

`not 1`

$A\langle B\langle \bot, \texttt{Bool}\rangle, \alpha_5\rangle$

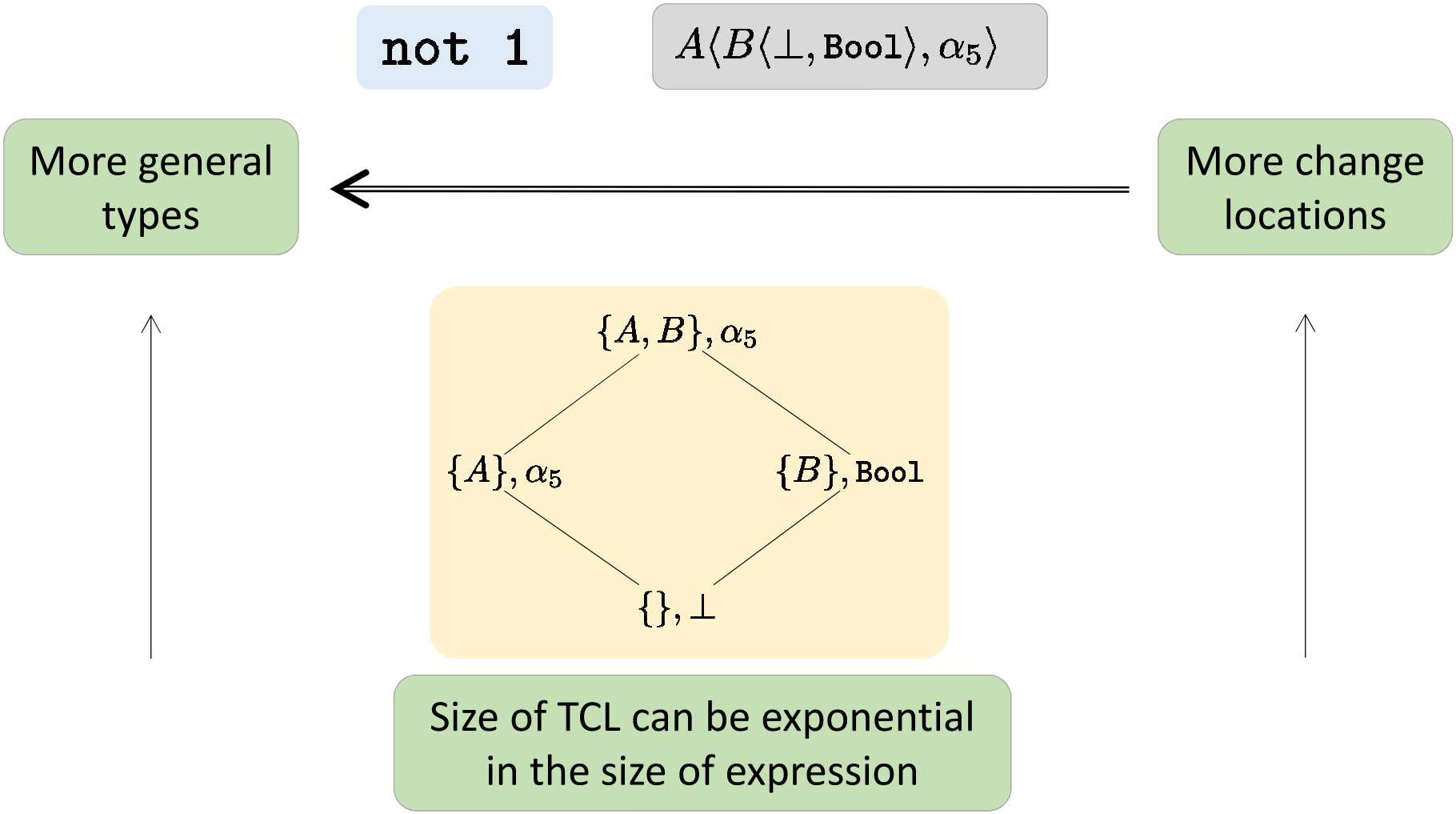
More general types

More change locations



Size of TCL can be exponential in the size of expression

Type change lattice



Important property

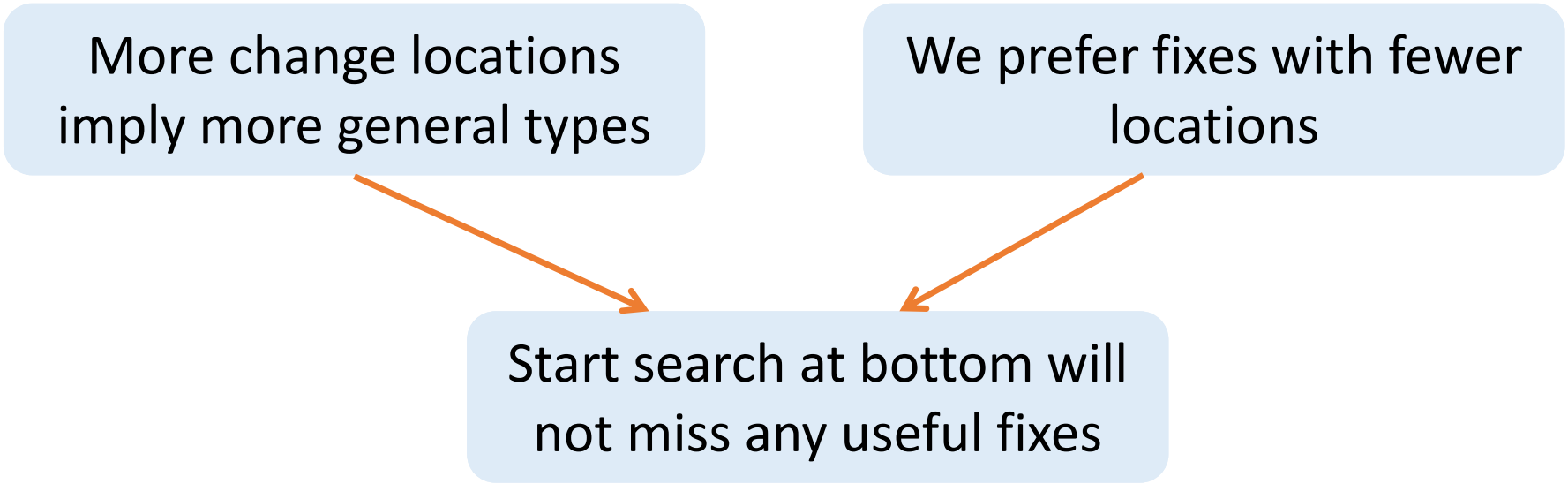
More change locations
imply more general types

Important property

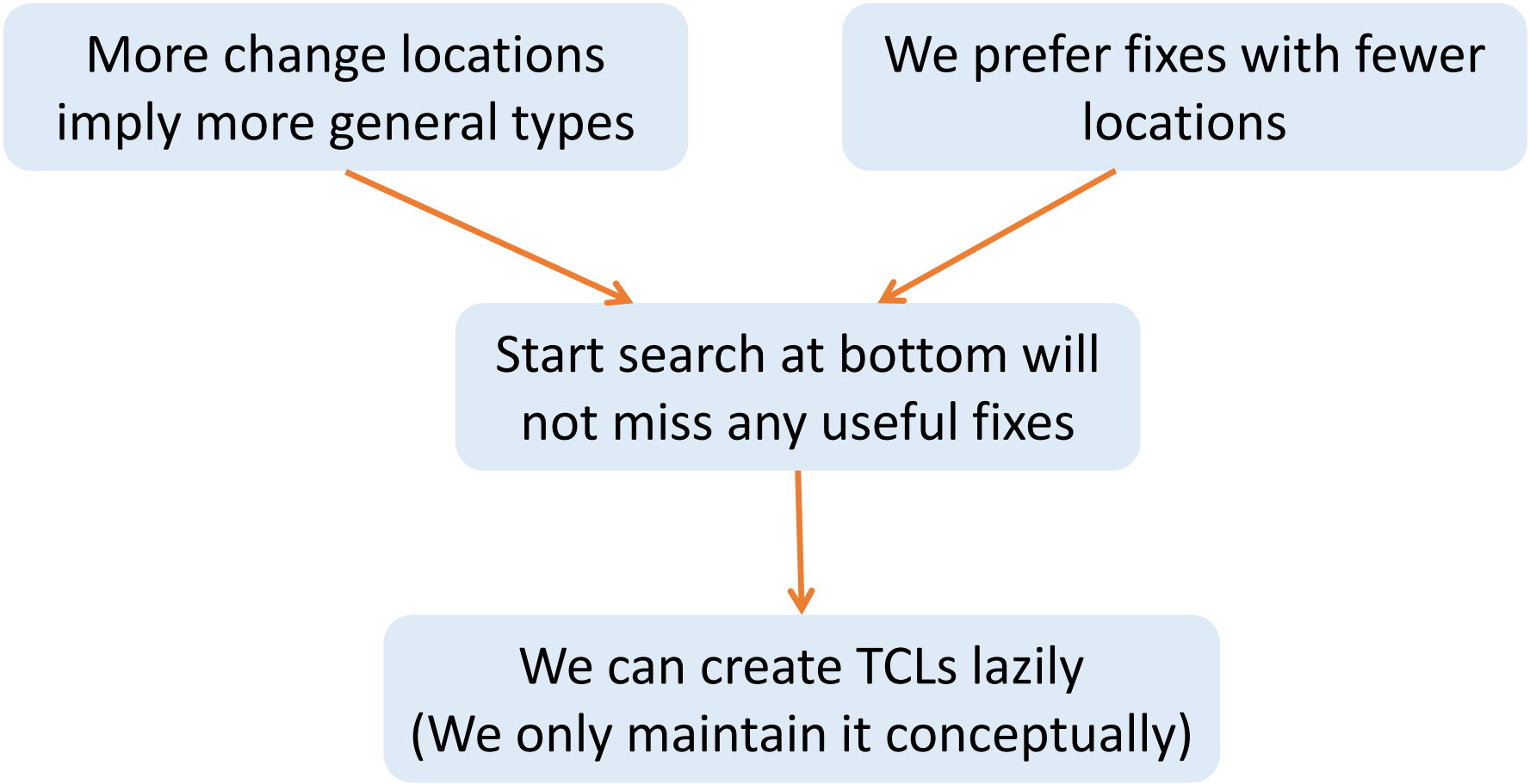
More change locations imply more general types

We prefer fixes with fewer locations

Important property



Important property



Climbing TCLs

```
not 1
```

		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`

		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`

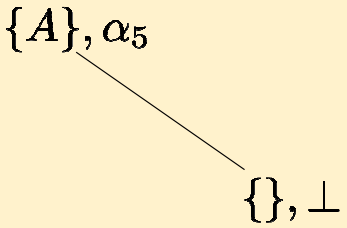
$\{\}, \bot$

		1	
		no change	*change*
not	*no change*	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Int}$ $\bot$	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Bool}$ $\texttt{Bool}$
	change	$\texttt{Int} \to \alpha_5$ $\texttt{Int}$ α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`

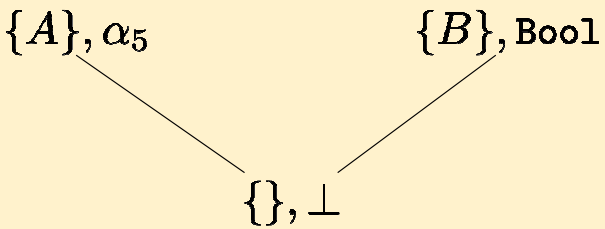


		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`

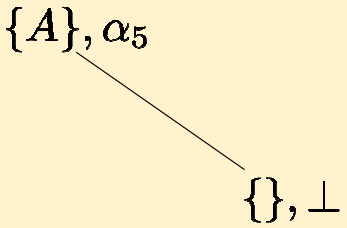


		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`

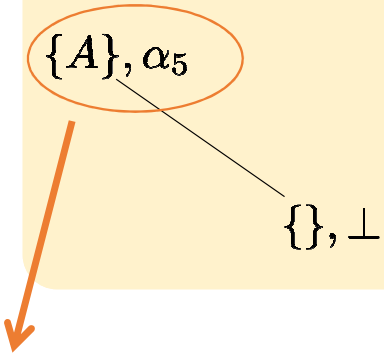


		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Int`



		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

```
Potential fixes:

1  Change not
     from type Bool -> Bool
     to type Int -> Int
There are no other one-change fixes.
Show two-change fixes? (y/n)
```

Climbing TCLs

```
not 1
```

		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Bool`

		1	
		no change	*change*
not	*no change*	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Int}$ $\bot$	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Bool}$ $\texttt{Bool}$
	change	$\texttt{Int} \to \alpha_5$ $\texttt{Int}$ α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Bool`

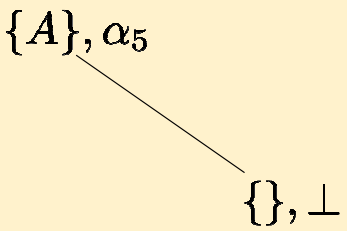
$\{\}, \bot$

		1	
		no change	*change*
not	*no change*	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Int}$ $\bot$	$\texttt{Bool} \to \texttt{Bool}$ $\texttt{Bool}$ $\texttt{Bool}$
	change	$\texttt{Int} \to \alpha_5$ $\texttt{Int}$ α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Bool`

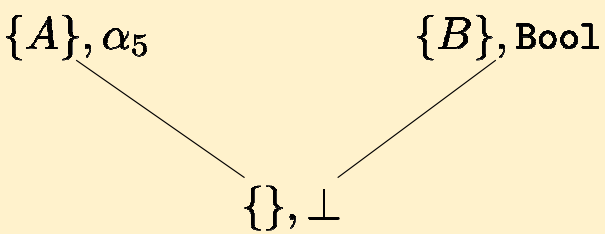


		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

`not 1`

Expected result type: `Bool`



		1	
		no change	*change*

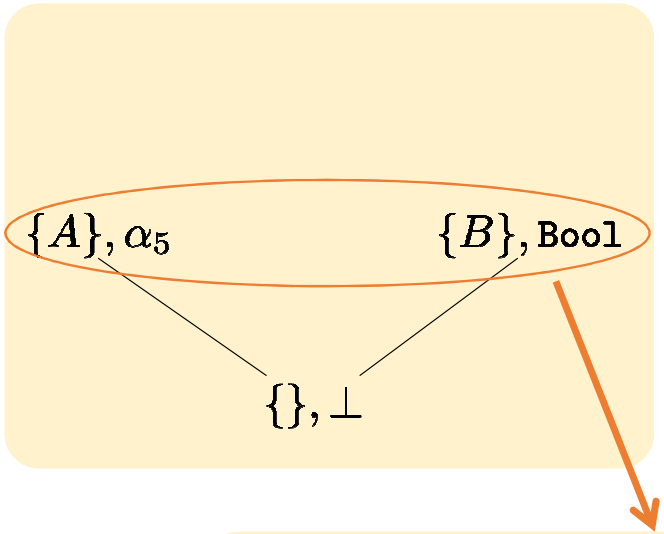
`not`	*no change*	`Bool` → `Bool`
		`Int`
		`Bool`
			`Bool` → `Bool`
			`Bool`
			`Bool`

	change	`Int` → α_5 `Int` α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

Climbing TCLs

```
not 1
```

Expected result type: `Bool`



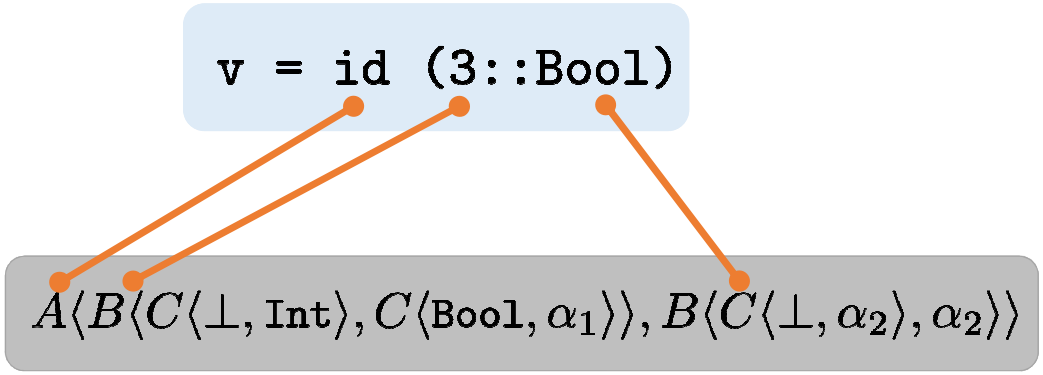
		1	
		no change	*change*
not	*no change*	Bool → Bool Int ⊥	Bool → Bool Bool Bool
	change	Int → α_5 Int α_5	$\alpha_4 \to \alpha_5$ α_4 α_5

```
Potential fixes:

1  Change 1 from type Int to type Bool
2  Change not
     from type Bool -> Bool
     to type Int -> Bool
There are no other one-change fixes.
Show two-change fixes? (y/n)
```

Debugging type errors in annotations

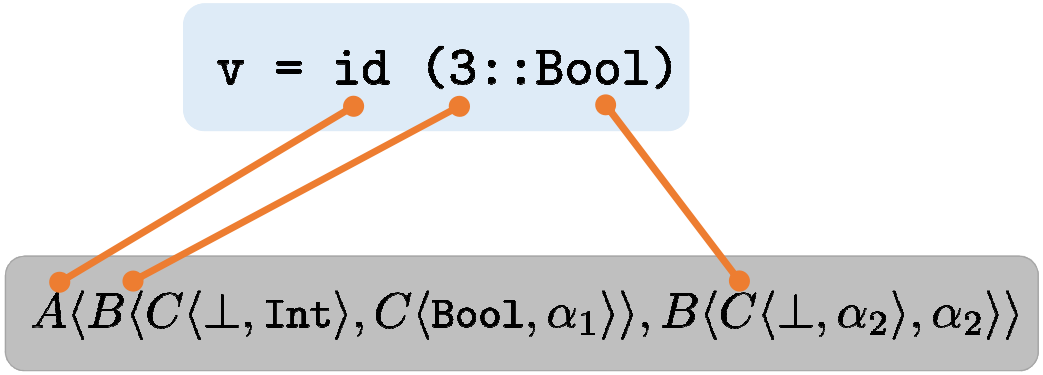
```
v = id (3::Bool)
```

Debugging type errors in annotations



Debugging type errors in annotations

Expected result type: `Int`



Debugging type errors in annotations

Expected result type: `Int`

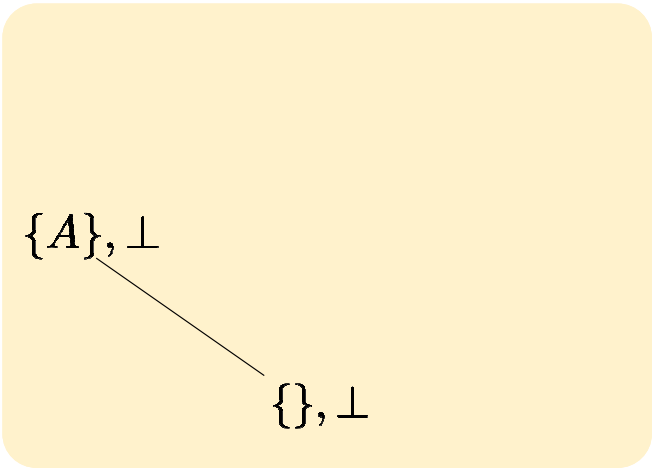
$\{\}, \bot$

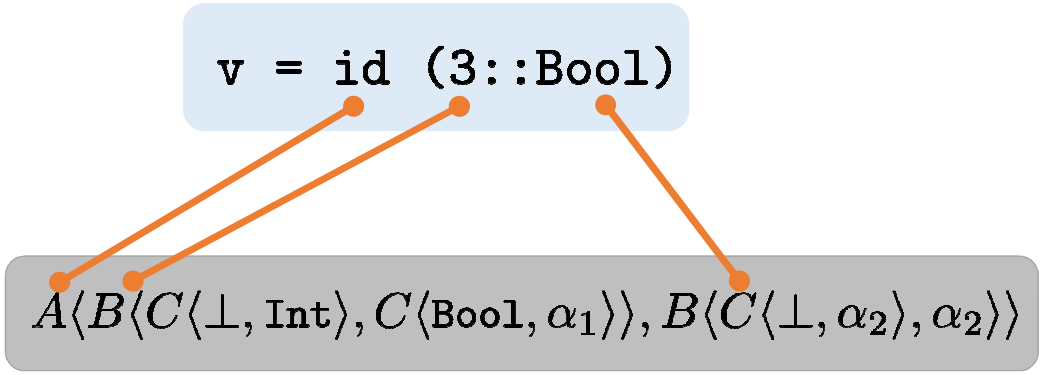
```
v = id (3::Bool)
```

$A\langle B\langle C\langle \bot, \texttt{Int}\rangle, C\langle \texttt{Bool}, \alpha_1\rangle\rangle, B\langle C\langle \bot, \alpha_2\rangle, \alpha_2\rangle\rangle$

Debugging type errors in annotations

Expected result type: `Int`





Debugging type errors in annotations

Expected result type: `Int`

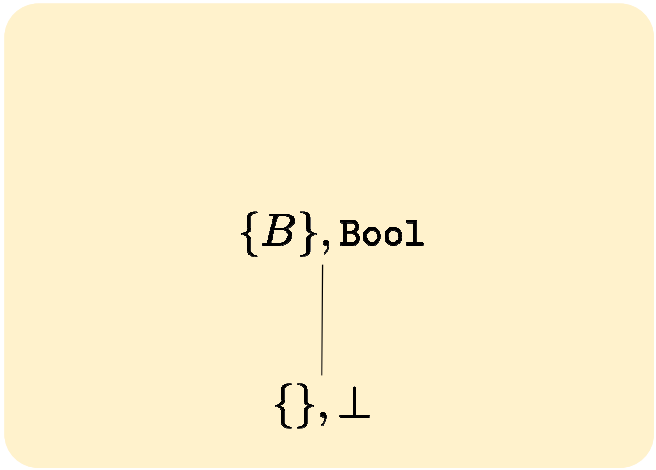
$\{\}, \bot$

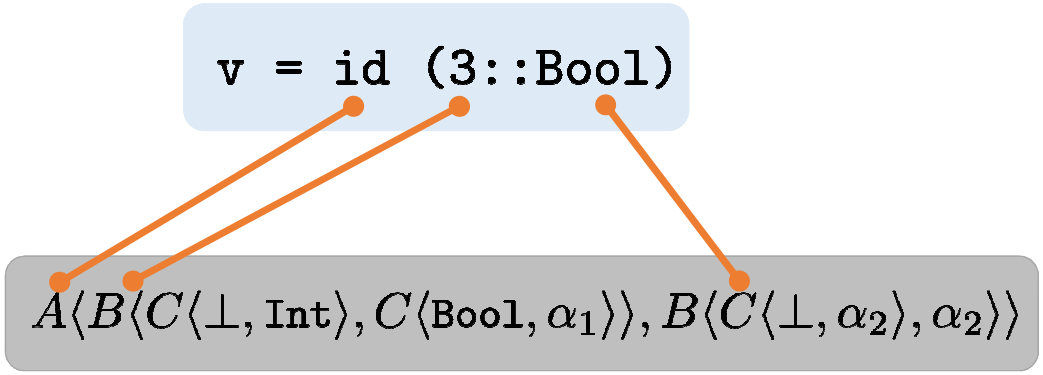
```
v = id (3::Bool)
```

$A\langle B\langle C\langle \bot, \texttt{Int}\rangle, C\langle \texttt{Bool}, \alpha_1\rangle\rangle, B\langle C\langle \bot, \alpha_2\rangle, \alpha_2\rangle\rangle$

Debugging type errors in annotations

Expected result type: `Int`





Debugging type errors in annotations

Expected result type: `Int`

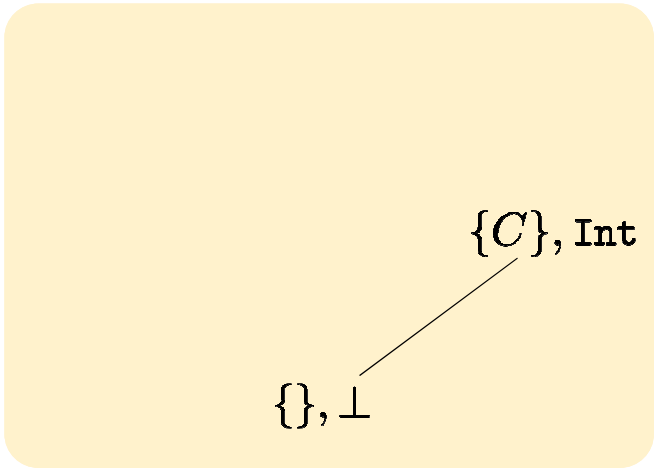
$\{\}, \bot$

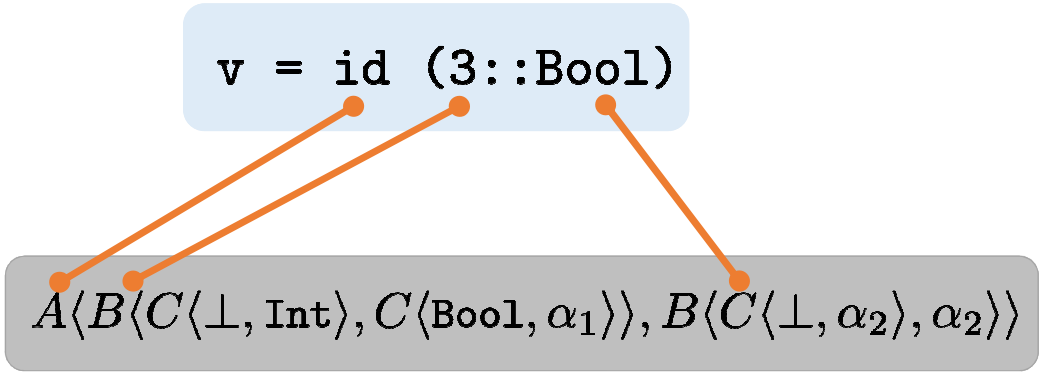
```
v = id (3::Bool)
```

$A\langle B\langle C\langle \bot, \texttt{Int}\rangle, C\langle \texttt{Bool}, \alpha_1\rangle\rangle, B\langle C\langle \bot, \alpha_2\rangle, \alpha_2\rangle\rangle$

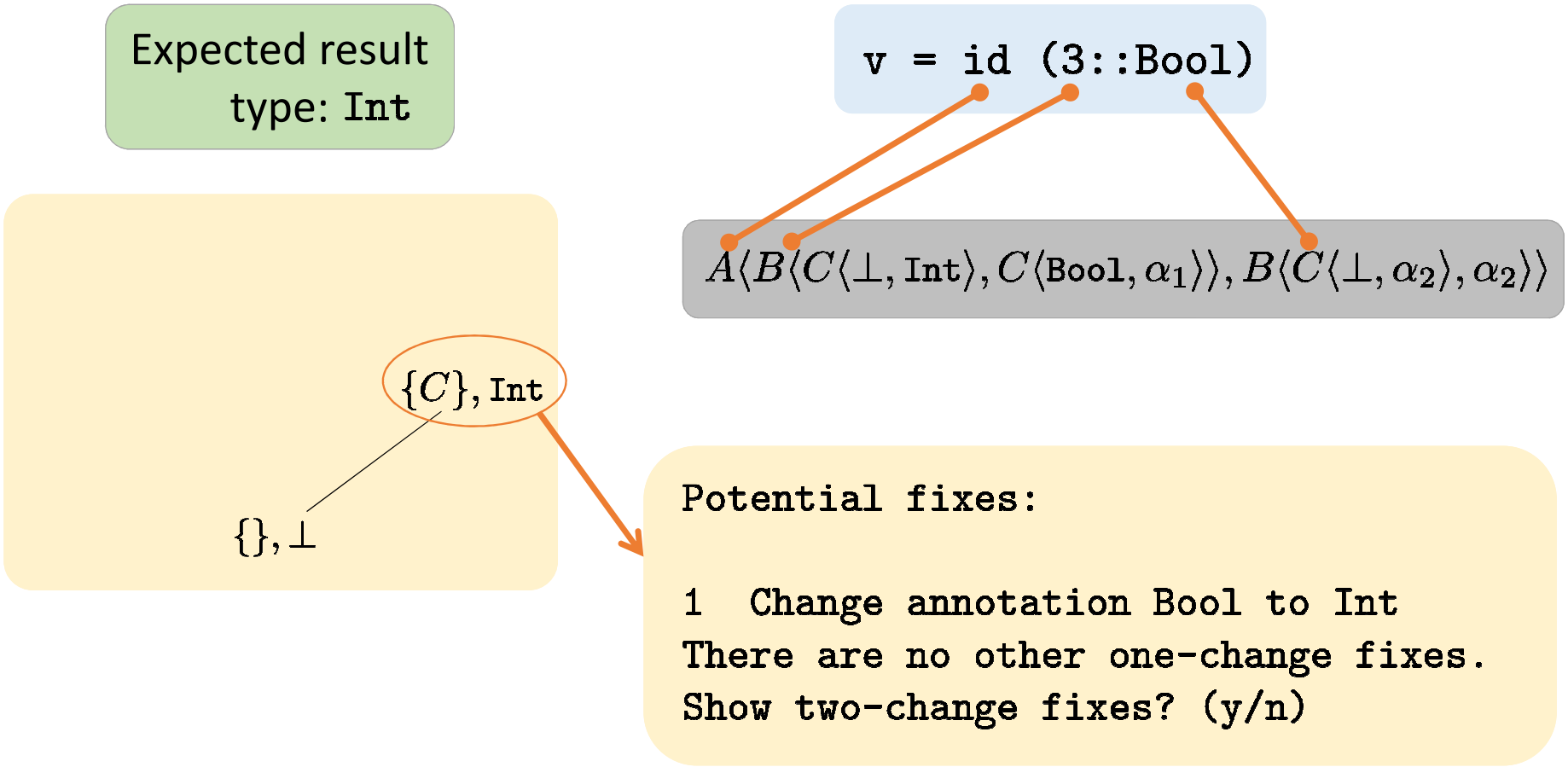
Debugging type errors in annotations

Expected result type: `Int`





Debugging type errors in annotations



Evaluation programs

Used earlier collected 86 programs from 22 publications

Evaluation programs

Used earlier collected 86 programs from 22 publications

Translated to the syntax presented in the paper

Evaluation programs

Used earlier collected 86 programs from 22 publications

Translated to the syntax presented in the paper

Each example exhibits a particular difficulty for type error diagnosing

Evaluation programs

```
\f y x -> f (y x) (y 3) (not x)
```

```
plot f dx dy oy =
    let fxs = getYs f dx
        ys = map (\y-> fromIntegral (y-oy)*dy) [maxY,maxY-1..minY]
        rows = map (doRow fxs) ys
    in unlines rows
    where
        doRow [] r = ""
        doRow (y:ys) r = (if y < r && y > (r-dy) then *
                          else  ) : doRow r ys
        getYs f dx = [ f ((centre x * dx)) | x <- [minX..maxX] ]
             where centre = (+) .5
```

```
idStack stk = pop (push undefined stk)
push top stk = (top:stk)
pop (top,stk) = stk
empty = []
```

```
split xs =
  case xs of
    [] -> ([],[])
    [x] -> ([], x)
    (x:y:zs) ->
      let (xs, ys) = split zs
      in (x:xs, y:ys)
```

```
reverse (x:xs) = reverse xs ++ x
last xs = head (reverse xs)
init = reverse . tail . reverse
rotateR xs = last xs : init xs
```

Evaluation results

	86 examples with Oracle				
	1	2	3	≥ 4	never
CF typing	67%	80%	88%	92%	8%
GTD	83%	90%	92%	92%	8%

Percentage of programs whose type errors could be removed with at most *n* suggestions

Evaluation results

	86 examples with Oracle				
	1	2	3	≥ 4	never
CF typing	67%	80%	88%	92%	8%
GTD	83%	90%	92%	92%	8%

Percentage of programs whose type errors could be removed with at most *n* suggestions

GTD overhead compared to counter-factual typing is less than 0.5s

In the paper

Formalization of the type system

In the paper

Formalization of the type system

Reliability of type annotations

In the paper

Formalization of the type system

Reliability of type annotations

Debugging type errors in annotations

In the paper

Formalization of the type system

Reliability of type annotations

Debugging type errors in annotations

More about TCLs

Future work

More expressive type systems

Detecting specification failures

Programming by search

Conclusions

GTD

An effective type debugging method guided by user input

Conclusions

GTD

An effective type debugging method guided by user input

Main idea

Finding all potential fixes and filtering them with the user expected type

Conclusions