Improving type-error messages in functional languages

Bastiaan Heeren
Universiteit Utrecht

January 5, 2001

Contents

- Introduction
- Constraints
- Type inference rules
- Solving constraints
- Solving inconsistencies
- Future work
- Conclusion

Introduction

Example of an erroneous expression:

```
f = \x -> case x of
             0 -> False
             1 -> “one”
             2 -> “two”
             3 -> “three”
```

Error message produced by Hugs:

```
ERROR “example.hs” (line 1): Type error in case expression
*** Term           : “one”
*** Type           : String
*** Does not match : Bool
```

Introduction

Problems in current type inferencing algorithms:

- local approach
- error message is only a brief explanation
- only one error message is reported

Improvements:

- global approach
- a better explanation of the type conflict
- multiple error messages are reported

Expression language

Expression language:

```
data Expr         = Variable    String
                  | Literal     String
                  | Apply       Expr Expr
                  | Lambda      String Expr
                  | Case        Expr Alternatives
                  | Let         String Expr Expr

data Alternatives = Empty
                  | Alternative Expr Expr Alternatives
```

- no syntactic sugar
- simplified let expression

Type language

Type language:

```
data Type = TVar  Int
          | TCon  String
          | TApp  Type Type
```

- type arrow ($\rightarrow$) is represented as a constant
- no quantifiers

Constraints

Equality constraint:

- represents the unification of two types

for example : $a \equiv Int \rightarrow b$

Instance constraint:

- represents an instantiation of a (polymorphic) type
- contains a set of monomorphic variables

for example : $a <_{\varnothing} Int \rightarrow b$

Type inference rules

[VAR]

$$\frac{a \text{ is fresh}}{\vdash\ x : a\ ,\ [x \rightarrow a]}$$

no constraints

[LIT]

$$\frac{}{\vdash\ \text{literal} : \text{primitive type}\ ,\ \emptyset}$$

no constraints

[APP]

$$\frac{\begin{array}{l} a \text{ is fresh} \\ \vdash\ f : tf\ ,\ Af \\ \vdash\ e : te\ ,\ Ae \end{array}}{\vdash\ (f\ e) : a\ ,\ Af \cup Ae}$$

$tf \equiv te \rightarrow a$

[ABS]

$$\frac{\begin{array}{l} a \text{ is fresh} \\ \vdash\ e : t\ ,\ A \end{array}}{\vdash\ (\backslash x \rightarrow e) : a \rightarrow t\ ,\ A \setminus x}$$

$\{\ s \equiv a \mid (x \rightarrow s) \in A\ \}$

Type inference rules

[CASE]

$$\frac{\begin{array}{ll} \text{a and b are fresh} & \\ \vdash\ p : tp\ ,\ Ap & \\ \vdash\ p_i : tp_i\ ,\ Ap_i & \text{for } i \in [1..n] \\ \vdash\ e_i : te_i\ ,\ Ae_i & \text{for } i \in [1..n] \end{array}}{\begin{array}{l} \vdash\ (\text{case } p \text{ of } p_1 \rightarrow e_1; \ldots ; p_n \rightarrow e_n;) : b\ , \\ \quad Ap \cup (Ae_1 - Ap_1) \cup \ldots \cup (Ae_n - Ap_n) \end{array}}$$

$a \equiv tp$

$a \equiv tp_i$ for $i \in [1..n]$

$b \equiv te_i$ for $i \in [1..n]$

$\{\ s \equiv t \mid (x \rightarrow s) \in Ap_i, (x \rightarrow t) \in Ae_i\ \}$ for $i \in [1..n]$

Type inference rules

[LET]

$$\frac{\begin{array}{l} a \text{ is fresh} \\ \vdash\ e : te\ ,\ Ae \\ \vdash\ b : tb\ ,\ Ab \end{array}}{\vdash\ (\text{let } x = e \text{ in } b) : tb\ ,\ (Ae \cup Ab) \setminus x}$$

$a \equiv te$

$\{\ s \equiv a \ \mid (x \to s) \in Ae\ \}$

$\{\ s <_M a \ \mid (x \to s) \in Ab\ \}$

where $M = Ae \setminus x$

Example

$\overline{\qquad\qquad\qquad\qquad}$ [VAR] $\overline{\qquad\qquad\qquad\qquad}$ [VAR]

$\vdash\ i : t3\ ,\ [i \rightarrow t3]$ $\qquad$ $\vdash\ i : t4\ ,\ [i \rightarrow t4]$

$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$ [APP]

$\vdash\ i\ i : t5\ ,\ [i \rightarrow t3,\ i \rightarrow t4]$ $\qquad t3 \equiv t4 \rightarrow t5$

$\overline{\qquad\qquad\qquad\qquad}$ [VAR]

$\vdash\ x : t0\ ,\ [x \rightarrow t0]$

$\overline{\qquad\qquad\qquad\qquad}$ [ABS]

$\vdash\ \backslash x \rightarrow x : t1 \rightarrow t0\ ,\ \varnothing$ $\qquad t1 \equiv t0$

$t3 <_{\varnothing} t2$

$t4 <_{\varnothing} t2$

$t2 \equiv t1 \rightarrow t0$

$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$ [LET]

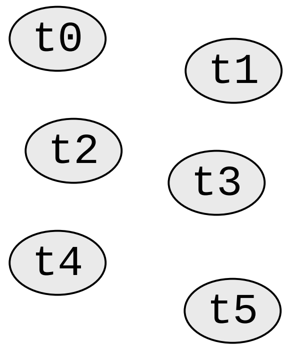
$\vdash\ \text{let } i = \backslash x \rightarrow x \text{ in } i\ i : t5\ ,\ \varnothing$

Solving constraints

Equality graph:
- vertex : type variable or type constant
- edge : equality constraint

Each type variable occurs exactly once in a vertex

Initial state for the expression “let i = \x → x in i i”

result type	set of constraints	equality graph
t5	#1 : $t1 \equiv t0$ #2 : $t2 \equiv t1 \to t0$ #3 : $t3 \equiv t4 \to t5$ #4 : $t3 <_{\varnothing} t2$ #5 : $t4 <_{\varnothing} t2$	
errors		

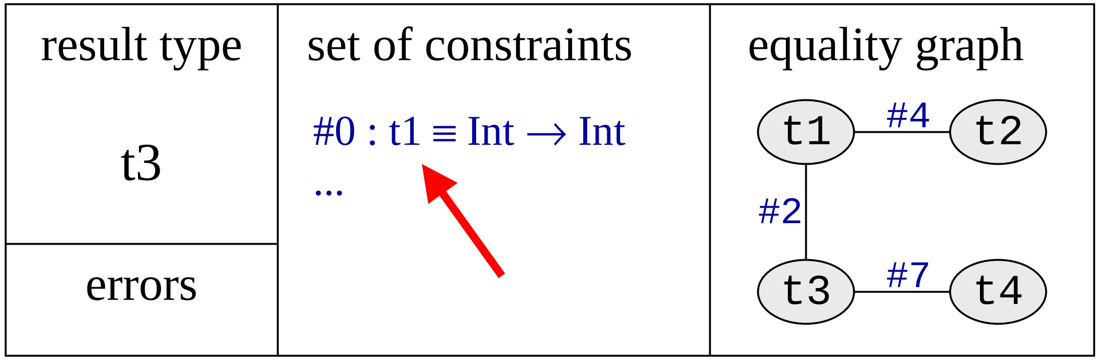
Solving constraints

Equality constraints: a rule for each combination

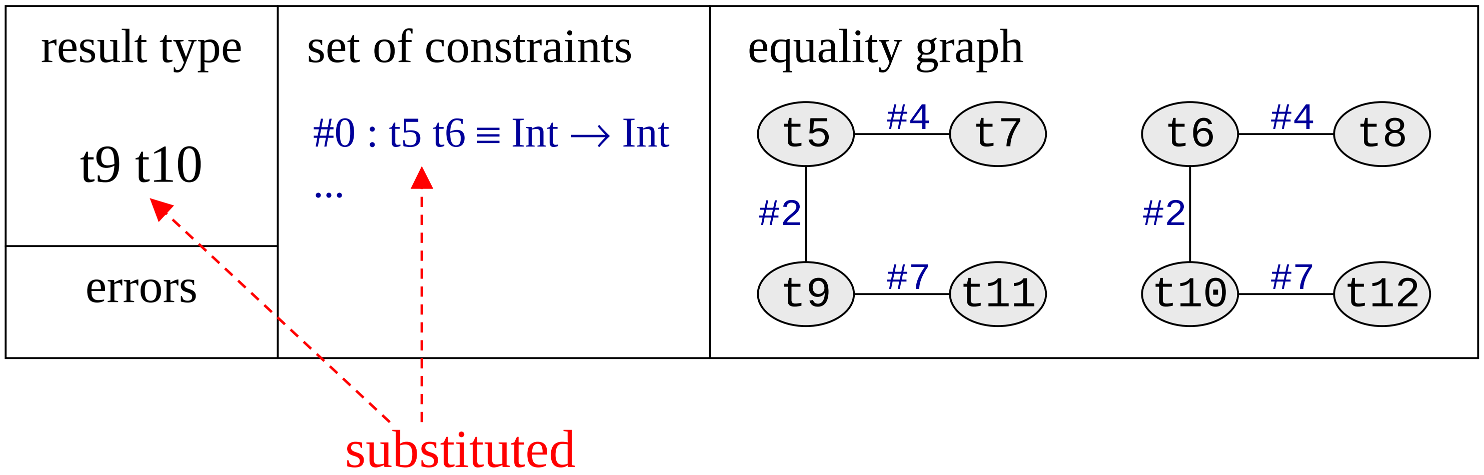
`TCon c1 ≡ TCon c2` if c1 and c2 are equal then remove else error	`TCon c ≡ TVar v` add an edge between c and v
`TCon c ≡ TApp t1 t2` error	`TVar v1 ≡ TVar v2` add an edge between v1 and v2

`TApp t1 t2 ≡ TApp t3 t4`
split constraint into
`(t1 ≡ t3)` and `(t2 ≡ t4)`
	`TVar v ≡ TApp t1 t2`
	decomposition of v

Solving constraints



Substitution: [t1 := t5 t6, t2 := t7 t8, t3 := t9 t10, t4 := t11 t12]



Solving constraints

Instance constraints:

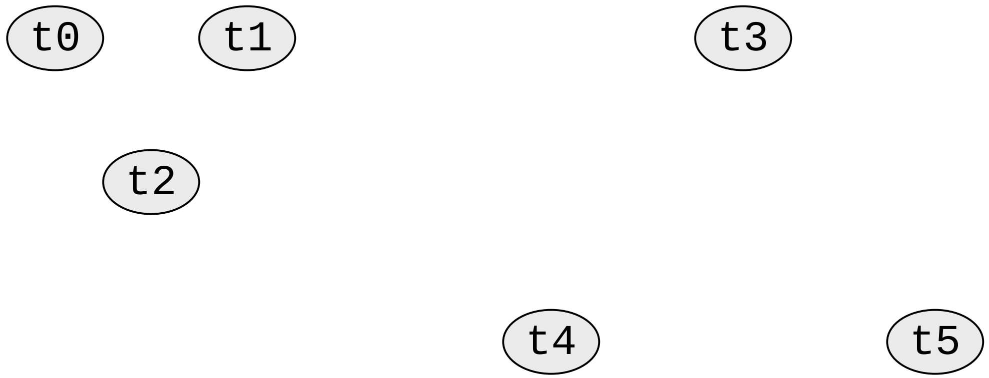
`t1 <M t2` is changed into an equality constraint as soon as t2 is *fixed*
(a *fixed* type does not change during the rest of the computation)

- A type is fixed if all the type variables it contains are fixed.
- A type variable is fixed if:
  - it does not occur in an equality constraint
  - it does not occur on the left hand side of an instance constraint
  - all the type variables in its *connected component* are fixed

Example

simplify →

t5			
#1	t1	$\equiv$	t0
#2	t2	$\equiv$	$t1 \rightarrow t0$
#3	t3	$\equiv$	$t4 \rightarrow t5$
#4	t3	$<\emptyset$	t2
#5	t4	$<\emptyset$	t2



Example

decompose →

t5			
#1	t1	≡	t0
#2	t2	≡	$t1 \to t0$
#3	t3	≡	$t4 \to t5$
#4	t3	$<\emptyset$	t2
#5	t4	$<\emptyset$	t2

substitution
$t2 := t6 \to t7$

Example

simplify → #2

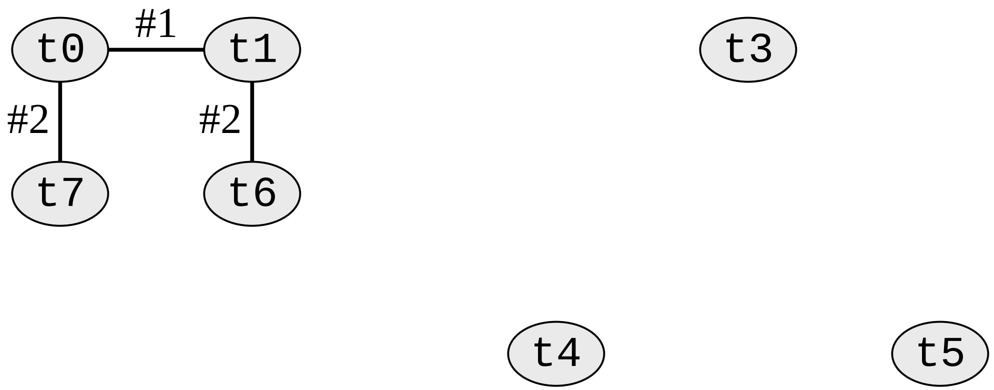
t5				
#1		t1	≡	t0
#2	t6 → t7		≡	t1 → t0
#3		t3	≡	t4 → t5
#4		t3	<∅	t6 → t7
#5		t4	<∅	t6 → t7

substitution
t2 := t6 → t7

Example

t5			
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t3	≡	t4 → t5
#4	t3	<∅	t6 → t7
#5	t4	<∅	t6 → t7

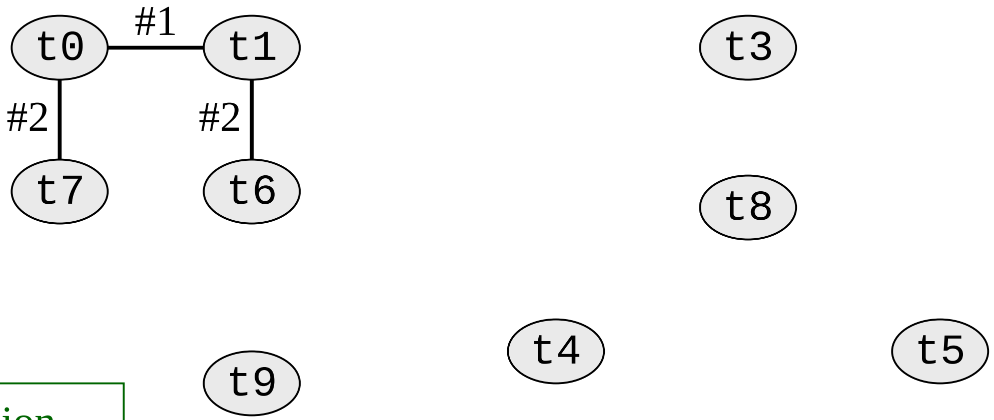
instantiate (→ #4, → #5)



Example

t5			
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t3	≡	t4 → t5
#4	t3	≡	t8 → t8
#5	t4	≡	t9 → t9

decompose → #3

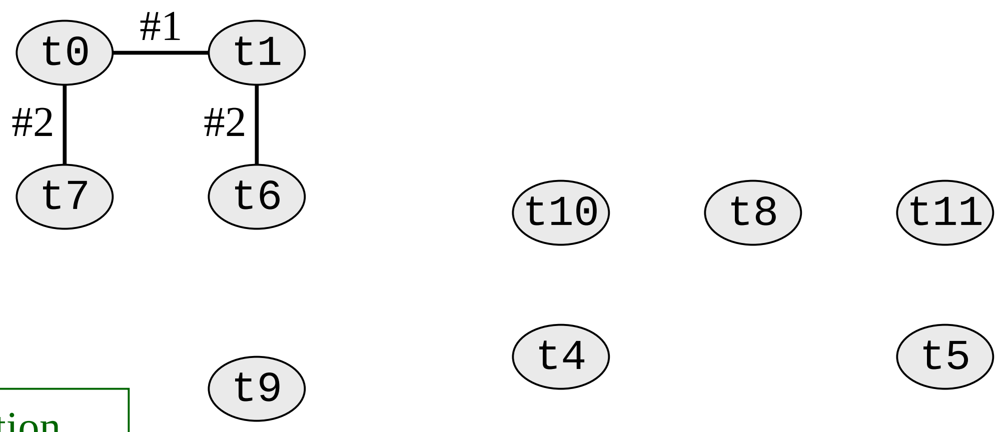


substitution
t3 := t10 → t11

Example

simplify → (#3)

t5			
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t10 → t11	≡	t4 → t5
#4	t10 → t11	≡	t8 → t8
#5	t4	≡	t9 → t9

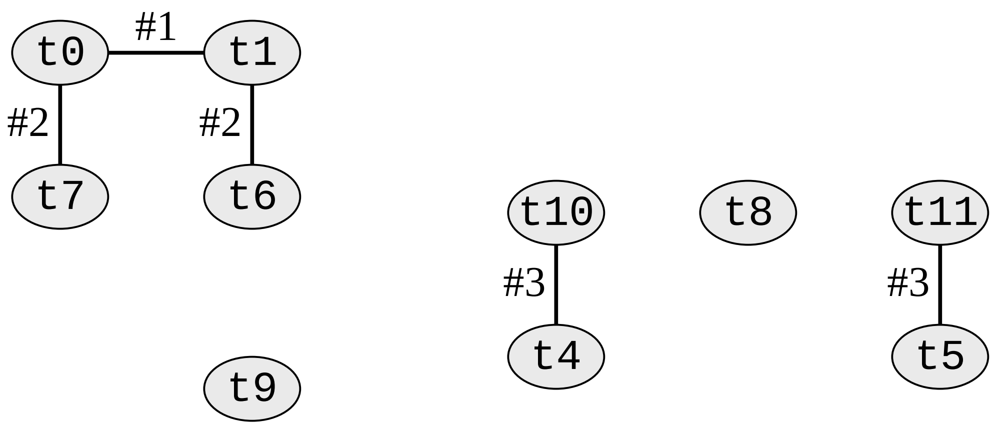


substitution
t3 := t10 → t11

Example

t5				
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t10 → t11	≡	t4 → t5
#4	t10 → t11	≡	t8 → t8
#5	t4	≡	t9 → t9

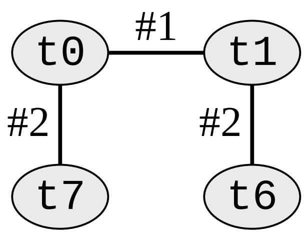
simplify → #4

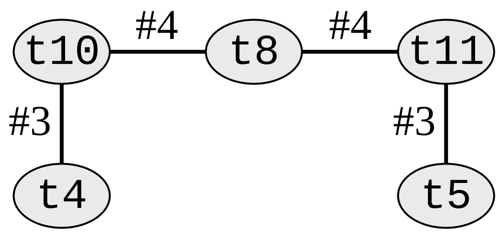


Example

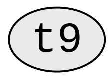
decompose →

t5			
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t10 → t11	≡	t4 → t5
#4	t10 → t11	≡	t8 → t8
#5	t4	≡	t9 → t9





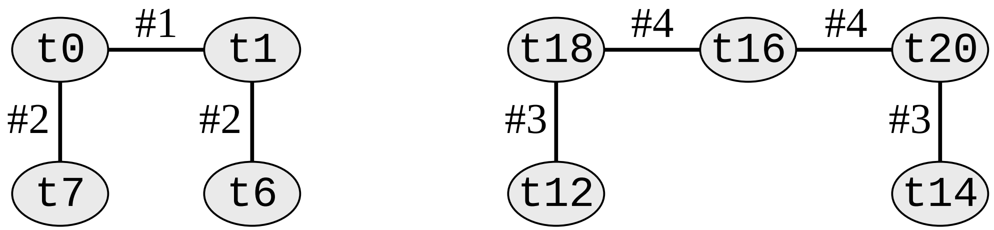
substitution	
t4	:= t12 → t13
t5	:= t14 → t15
t8	:= t16 → t17
t10	:= t18 → t19
t11	:= t20 → t21



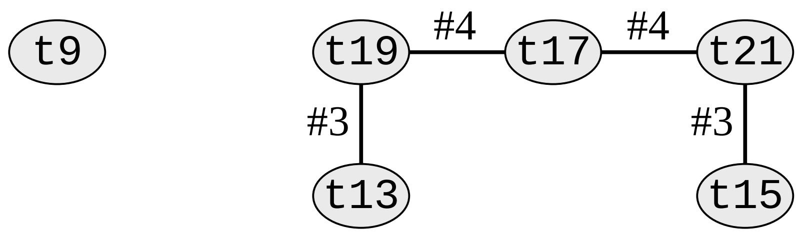
Example

t14 → t15			
#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t10 → t11	≡	t4 → t5
#4	t10 → t11	≡	t8 → t8
#5	t12 → t13	≡	t9 → t9

simplify → #5



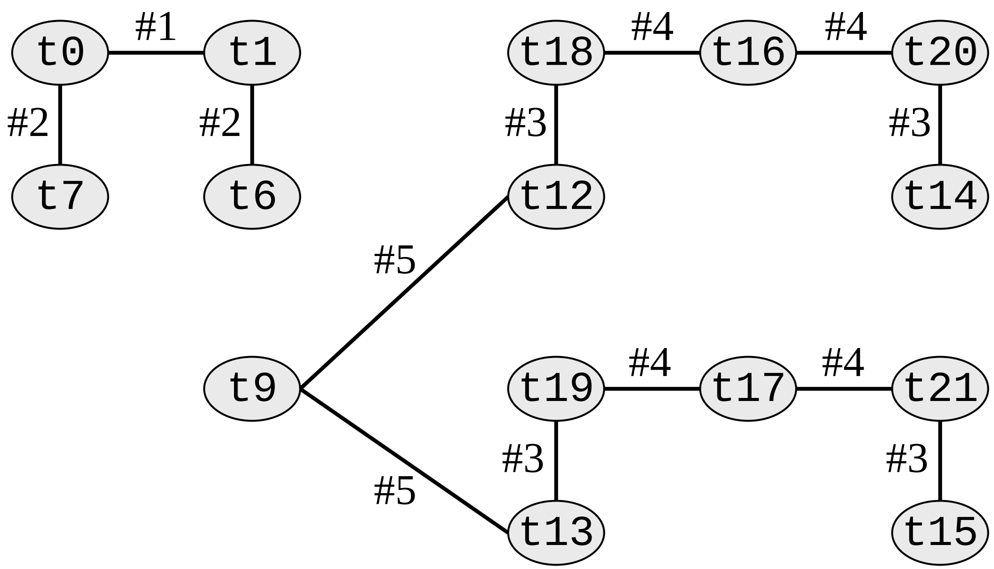
substitution
t4 := t12 → t13
t5 := t14 → t15
t8 := t16 → t17
t10 := t18 → t19
t11 := t20 → t21



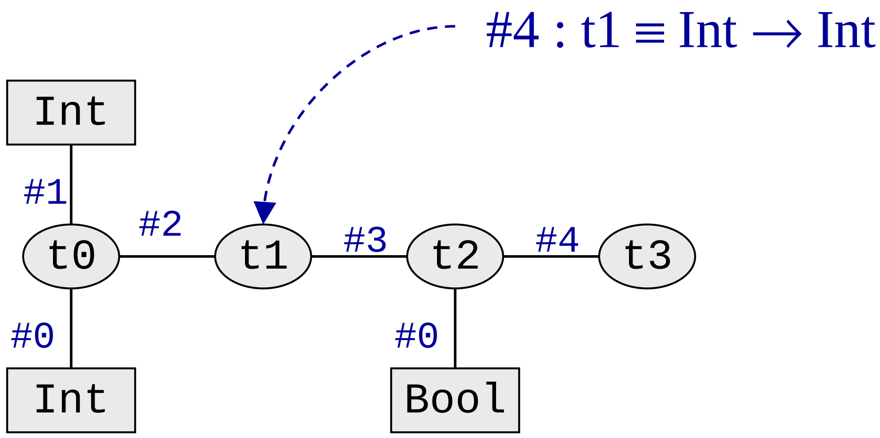
Example

result type → t14 → t15 → a → a

#1	t1	≡	t0
#2	t6 → t7	≡	t1 → t0
#3	t10 → t11	≡	t4 → t5
#4	t10 → t11	≡	t8 → t8
#5	t12 → t13	≡	t9 → t9



Solving inconsistencies



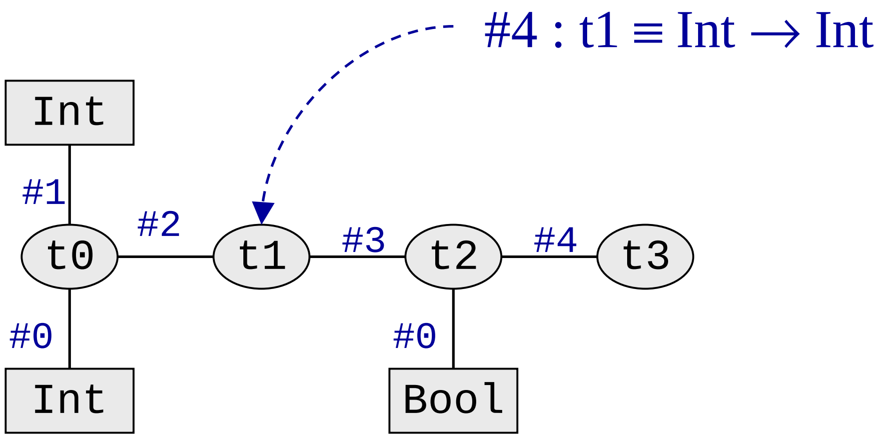
wrong paths:

- {#1,#2,#3,#0}
- {#0,#2,#3,#0}

decompose paths:

- {#4,#2,#1}
- {#4,#2,#0}
- {#4,#3,#0}

Solving inconsistencies



wrong paths:

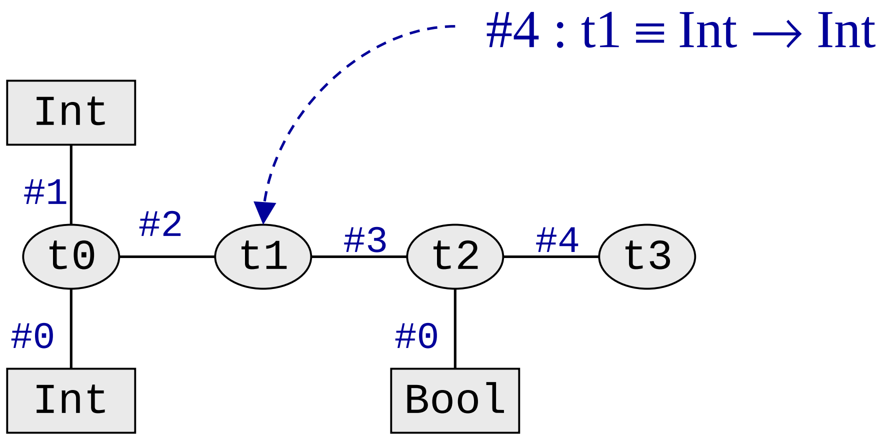
- {#1,#2,#3,#4}
- {#0,#2,#3,#0}

decompose paths:

- {#4,#2,#1}
- {#4,#2,#0}
- {#4,#3,#0}

minimal set
{#0,#1}
{#0,#2}
{#0,#4}
{#2,#3}
{#2,#4}
{#3,#4}

Solving inconsistencies



removal cost

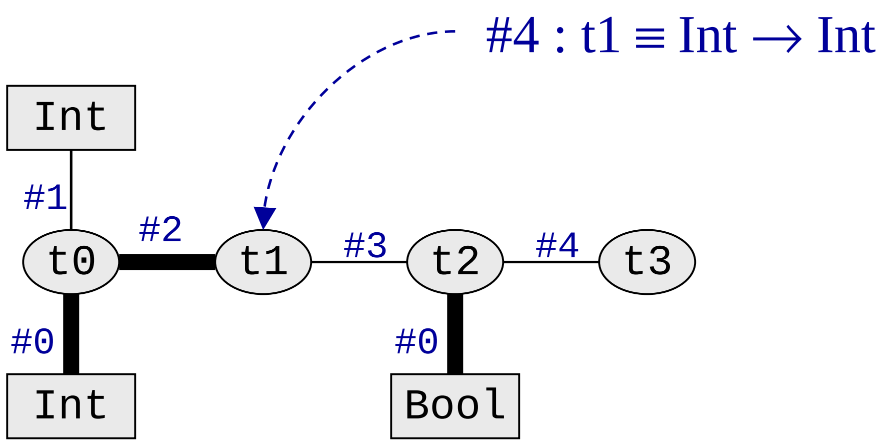
	good		
#0	1		
#1	1		
#2	-		
#3	-		
#4	-		

good paths:

- {#0,#1}

minimal set
{#0,#1}
{#0,#2}
{#0,#4}
{#2,#3}
{#2,#4}
{#3,#4}

Solving inconsistencies



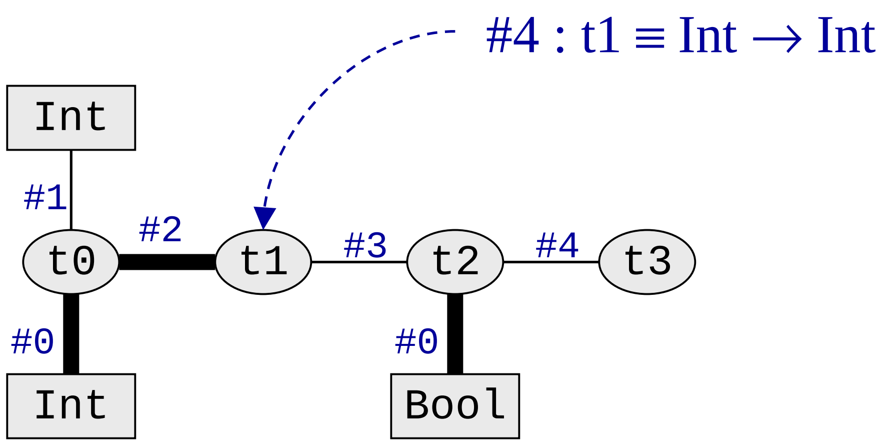
removal cost

	good	trust	
#0	1	5	
#1	1	1	
#2	-	5	
#3	-	1	
#4	-	1	

each constraint
has a trust value

minimal set
{#0,#1}
{#0,#2}
{#0,#4}
{#2,#3}
{#2,#4}
{#3,#4}

Solving inconsistencies



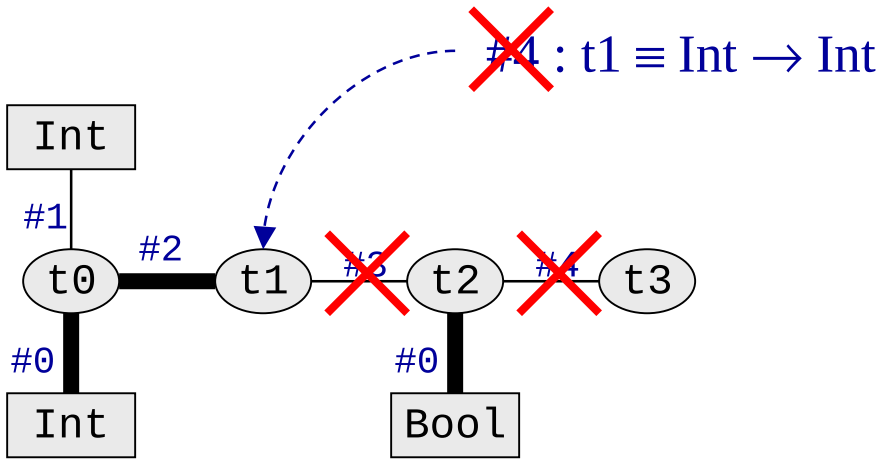
removal cost

	good	trust	cost
#0	1	5	10
#1	1	1	2
#2	-	5	5
#3	-	1	1
#4	-	1	1

(1 + good) * trust

minimal set
{#0,#1}
{#0,#2}
{#0,#4}
{#2,#3}
{#2,#4}
{#3,#4}

Solving inconsistencies



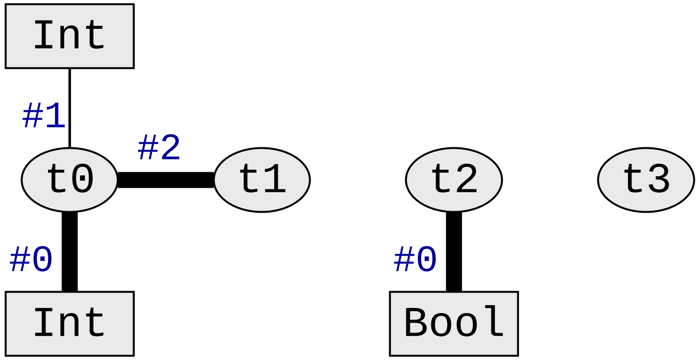
removal cost

	good	trust	cost
#0	1	5	10
#1	1	1	2
#2	-	5	5
#3	-	1	1
#4	-	1	1

remove the constraints in the set with the lowest total cost

minimal set	total cost
{#0,#1}	12
{#0,#2}	15
{#0,#4}	11
{#2,#3}	6
{#2,#4}	6
{#3,#4}	2 ←

Solving inconsistencies



the inconsistency is removed

Future work

- Explicit typing
- Type synonyms
- Type classes
- Kind inferencing
- Type tracing
- Advanced output
- BANE

Conclusion

- *Left-to-right* bias is completely removed
- Several heuristics increase the exactness of the error message considerably
- Possibility to add more heuristics