Building a Typed Scripting Language

Zachary Palmer

The Johns Hopkins University

April 16th, 2015

Scripting: Good and Bad

```
print "What number?"
number = raw_input();
if number < 4:
    print "Small!"
else:
    print "Not small!"
```

Scripting: Good and Bad

```
print "What number?"
number = raw_input();
if number < 4:
    print "Small!"
else:
    print "Not small!"
```

Scripting languages are...

- Terse and legible: easy to read and write

Scripting: Good and Bad

```
print "What number?"
number = raw_input();
if number < 4:
    print "Small!"
else:
    print "Not small!"
```

Scripting languages are...

- Terse and legible: easy to read and write
- Flexible

Scripting: Good and Bad

```
print "What number?"
number = raw_input();
if number < 4:
    print "Small!"
else:
    print "Not small!"
```

Scripting languages are...

- Terse and legible: easy to read and write
- Flexible
- High-level

Scripting: Good and Bad

```
print "What number?"
number = raw_input();
if number < 4:
    print "Small!"
else:
    print "Not small!"
```

Scripting languages are...

- Terse and legible: easy to read and write
- Flexible
- High-level
- Error-prone

Scripting: Good and Bad

```java
import java.util.*;
public class SmallnessDetector {
    public static void main(String[] arg) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("What number?");
        int number = scanner.nextLine();
        if (number < 4) {
            System.out.println("Small!");
        } else {
            System.out.println("Not small!");
        }
    }
}
```

Scripting: Good and Bad

```java
import java.util.*;
public class SmallnessDetector {
    public static void main(String[] arg) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("What number?");
        int number = scanner.nextLine();
        if (number < 4) {
            System.out.println("Small!");
        } else {
            System.out.println("Not small!");
        }
    }
}
```

- Expression has type `String`
- Variable declared with type `int`

The Best of Both Worlds

Why can't we just create a type system?

The Best of Both Worlds

Why can't we just create a type system? We have.

The Best of Both Worlds

Why can't we just create a type system? We have.

Rubydust

Flow

MyPy

TeJaS

. . .

Diamondback Ruby

. . .

PHP+QB

Hack

TypeScript

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

```
x = 5
print x + 1
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

int

```
x = 5
print x + 1
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

int

```
x = 5
print x + 1
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

int

```
x = 5
print x + 1
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

```python
fs = [str, str.strip, len]
xs = [True, " very ", "ab"]
for (f,x) in zip(fs,xs):
    print f(x)
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

```
fs = [str, str.strip, len]
xs = [True, " very ", "ab"]
for (f,x) in zip(fs,xs):
    print f(x)
```

```
x = 5
locals()[raw_input()] = "foo"
print x + 1
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

```
fs = [str, str.strip, len]
xs = [True, " very ", "ab"]
for (f,x) in zip(fs,xs):
    print f(x)
```

```
x = 5
locals()[raw_input()] = "foo"
print x + 1
```

```
exec(open(raw_input()).read())
```

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Observations:

- Fundamentally dynamic operations

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Observations:

- Fundamentally dynamic operations
- Contrived – not necessary for “scripting”

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Observations:

- Fundamentally dynamic operations
- Contrived – not necessary for "scripting"

- Alternative: **build a typed scripting language from scratch**

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Observations:

- Fundamentally dynamic operations
- Contrived – not necessary for "scripting"

- Alternative: **build a typed scripting language from scratch**
  - Include "scripty" expressiveness

Retrofitting is Challenging

- Retrofitting: appealing, but very hard.
- Scripting languages: designed without type systems in mind.

Observations:

- Fundamentally dynamic operations
- Contrived – not necessary for "scripting"

- Alternative: **build a typed scripting language from scratch**
  - Include "scripty" expressiveness
  - Avoid dynamic operations

This Talk

Building a Typed Scripting Language

This Talk

Building a Typed Scripting Language

- Primary contribution: synthesis

This Talk

Building a Typed Scripting Language

- Primary contribution: synthesis
  - Underused type theory

This Talk

Building a Typed Scripting Language

- Primary contribution: synthesis
  - Underused type theory
  - Some abstract interpretation

This Talk

Building a Typed Scripting Language

- Primary contribution: synthesis
  - Underused type theory
  - Some abstract interpretation
  - New type theory

This Talk

Building a Typed Scripting Language

- Primary contribution: synthesis
  - Underused type theory
  - Some abstract interpretation
  - New type theory
- Thesis: *It is possible to construct a language which has the static analyzability of traditional languages and the flexibility of scripting languages.*

Outline

- Duck Type Inference
- Conditional Reasoning
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Outline

- Duck Type Inference
- Conditional Reasoning
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Duck Typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

James Whitcomb Riley

Duck Typing

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

James Whitcomb Riley

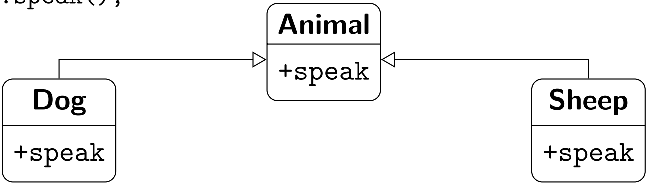
Duck typing: categorizing data based on its exhibited properties (as opposed to by explicit grouping).

Java: No Duck Typing

```java
public interface Animal {
    public void speak();
}
public class Dog implements Animal {
    public void speak() {
        System.out.println("Woof!");
    }
}
...
Animal animal = new Dog();
animal.speak();
animal = new Sheep();
animal.speak();
```

Java: No Duck Typing

```java
public interface Animal {
    public void speak();
}
public class Dog implements Animal {
    public void speak() {
        System.out.println("Woof!");
    }
}
...
Animal animal = new Dog();
animal.speak();
animal = new Sheep();
animal.speak();
```

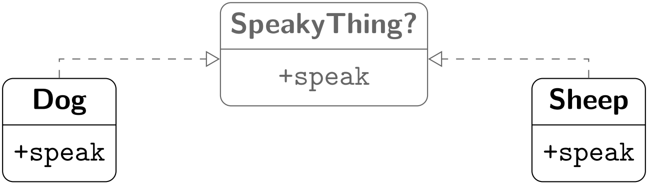


Python: Duck Typing

```python
class Dog:
    def speak(self): print "Woof!"
...
animal = Dog();
animal.speak();
animal = Sheep();
animal.speak();
```

Python: Duck Typing

```
class Dog:
    def speak(self): print "Woof!"
...
animal = Dog();
animal.speak();
animal = Sheep();
animal.speak();
```



How do we type it?

Constraint Types

Let's use constraint types!

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha \backslash \{\alpha \leq \texttt{has speak()}\}$

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha \backslash \{\alpha \leq \text{has } \texttt{speak()}\}$
- Expressive:
  - Union types: $\alpha \backslash \{\alpha \geq \texttt{int}\,, \alpha \geq \texttt{char}\}$
  - Recursive types: $\alpha_1 \backslash \{\texttt{nil} \leq \alpha_1, \texttt{cons}\ \ \alpha_2\ \alpha_1 \leq \alpha_1\}$
  - etc.

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha \backslash \{\alpha \leq \text{has } \texttt{speak()}\}$
- Expressive:
  - Union types: $\alpha \backslash \{\alpha \geq \texttt{int}, \alpha \geq \texttt{char}\}$
  - Recursive types: $\alpha_1 \backslash \{\texttt{nil} \leq \alpha_1, \texttt{cons}\ \alpha_2\ \alpha_1 \leq \alpha_1\}$
  - etc.
- Inhabited iff deductive closure is consistent

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha\backslash\{\alpha \leq \text{has } \texttt{speak()}\}$
- Expressive:
  - Union types: $\alpha\backslash\{\alpha \geq \texttt{int}, \alpha \geq \texttt{char}\}$
  - Recursive types: $\alpha_1\backslash\{\texttt{nil} \leq \alpha_1, \texttt{cons}\ \alpha_2\ \alpha_1 \leq \alpha_1\}$
  - etc.
- Inhabited iff deductive closure is consistent
  - $\alpha_1\backslash\{\alpha_2 \geq \texttt{int}, \alpha_1 \geq \alpha_2, \texttt{char} \geq \alpha_1\}$

Constraint Types

Let's use constraint types!

- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha \backslash \{\alpha \leq \text{has } \texttt{speak()}\}$
- Expressive:
  - Union types: $\alpha \backslash \{\alpha \geq \texttt{int}, \alpha \geq \texttt{char}\}$
  - Recursive types: $\alpha_1 \backslash \{\texttt{nil} \leq \alpha_1, \texttt{cons}\ \alpha_2\ \alpha_1 \leq \alpha_1\}$
  - etc.
- Inhabited iff deductive closure is consistent
  - $\alpha_1 \backslash \{\alpha_2 \geq \texttt{int}, \alpha_1 \geq \alpha_2, \texttt{char} \geq \alpha_1\}$
  - $\alpha_1 \backslash \{\alpha_2 \geq \texttt{int}, \alpha_1 \geq \alpha_2, \texttt{char} \geq \alpha_1, \texttt{char} \geq \texttt{int}\}$

Constraint Types

Let's use constraint types!

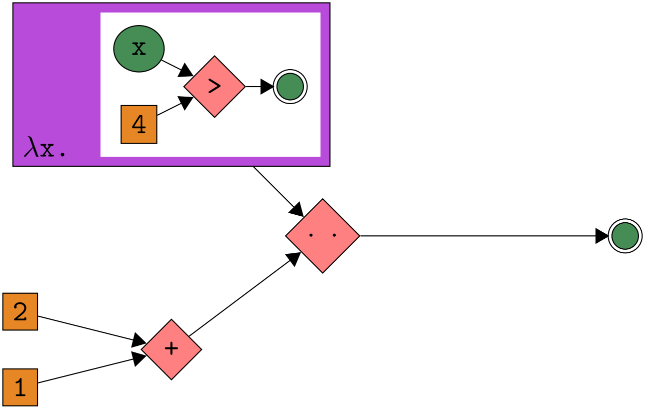
- Commonly used to type existing scripting languages
  - Well-suited to type inference
- Type described by restrictions on variables
  - $\alpha \backslash \{\alpha \leq \text{has } \texttt{speak()}\}$
- Expressive:
  - Union types: $\alpha \backslash \{\alpha \geq \texttt{int}, \alpha \geq \texttt{char}\}$
  - Recursive types: $\alpha_1 \backslash \{\texttt{nil} \leq \alpha_1, \texttt{cons}\ \alpha_2\ \alpha_1 \leq \alpha_1\}$
  - etc.
- Inhabited iff deductive closure is consistent
  - $\alpha_1 \backslash \{\alpha_2 \geq \texttt{int}, \alpha_1 \geq \alpha_2, \texttt{char} \geq \alpha_1\}$
  - $\alpha_1 \backslash \{\alpha_2 \geq \texttt{int}, \alpha_1 \geq \alpha_2, \texttt{char} \geq \alpha_1, \texttt{char} \geq \texttt{int}\}$
  - $\texttt{char} \geq \texttt{int}$ is false, so this type does not exist!

Intuition

```
(λx. x>4) (2+1)
```

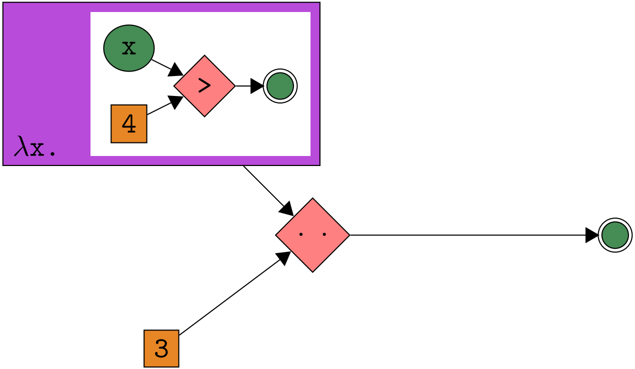
Intuition

```
(λx. x>4) (2+1)
```



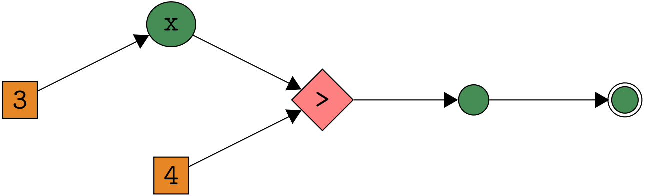
Intuition

```
(λx. x>4) (2+1)
```



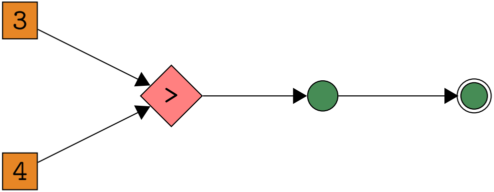
Intuition

```
(λx. x>4) (2+1)
```



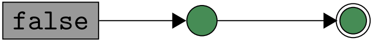
Intuition

```
(λx. x>4) (2+1)
```



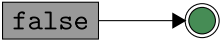
Intuition

```
(λx. x>4) (2+1)
```



Intuition

```
(λx. x>4) (2+1)
```



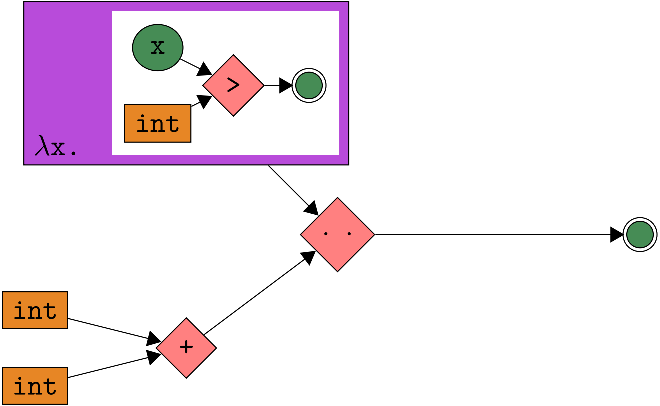
Intuition

```
(λx. x>4) (2+1)
```

$$\left\{\begin{array}{l} \alpha_{\mathsf{z}} \geq \alpha_{\mathsf{f}}\ \alpha_{\mathsf{a}}, \\ \alpha_{\mathsf{a}} \geq \texttt{int} \mathbin{\textsf{+}} \texttt{int} \\ \alpha_{\mathsf{f}} \geq \alpha_{\mathsf{x}} \rightarrow \alpha_{\mathsf{r}} \setminus \{\alpha_{\mathsf{r}} \geq \alpha_{\mathsf{x}} \mathbin{\textsf{>}} \texttt{int}\} \end{array}\right\}$$

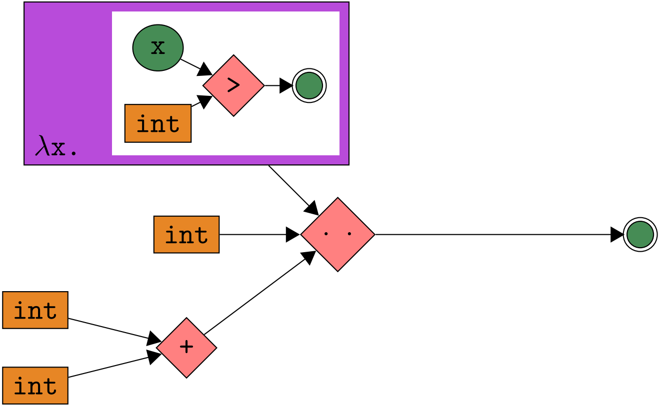
Intuition

```
(λx. x>4) (2+1)
```



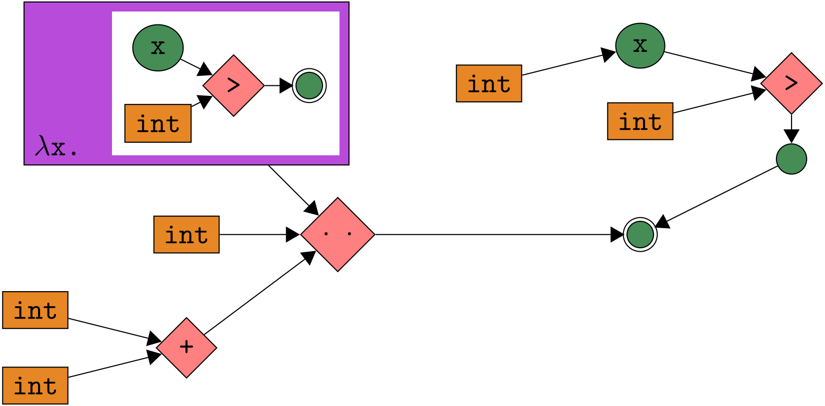
Intuition

```
1 (λx. x>4) (2+1)
```



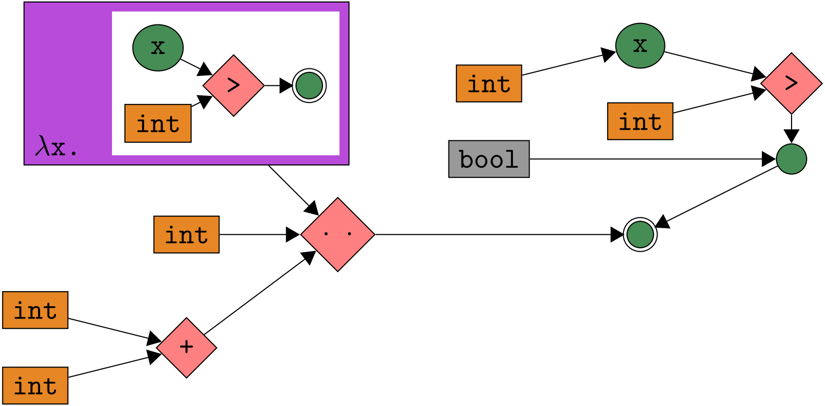
Intuition

```
1 (λx. x>4) (2+1)
```



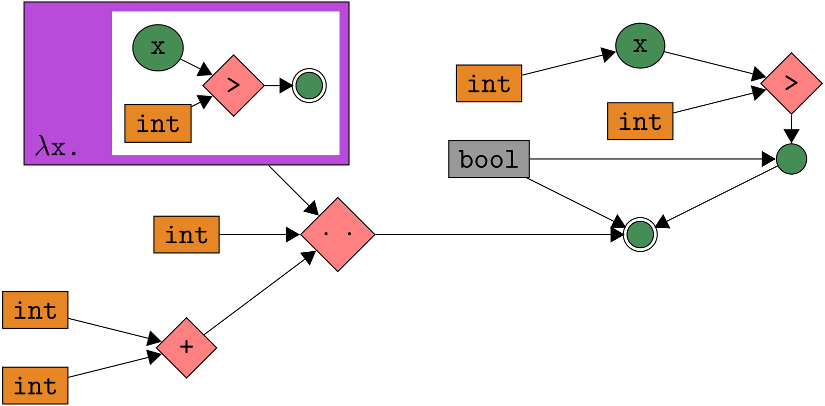
Intuition

```
1 (λx. x>4) (2+1)
```



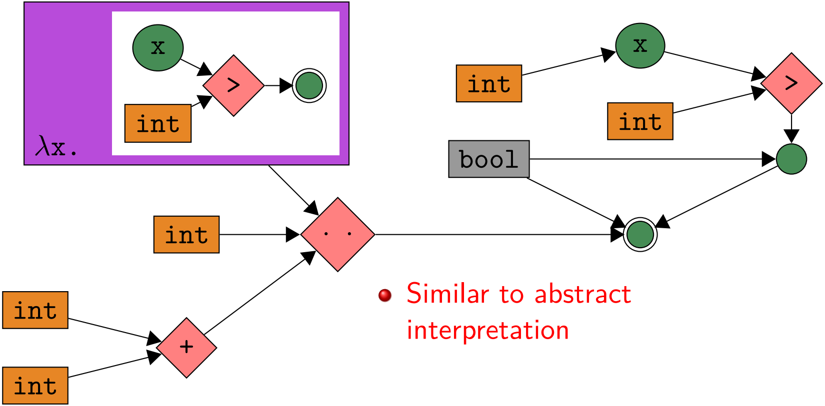
Intuition

```
(λx. x>4) (2+1)
```



Intuition

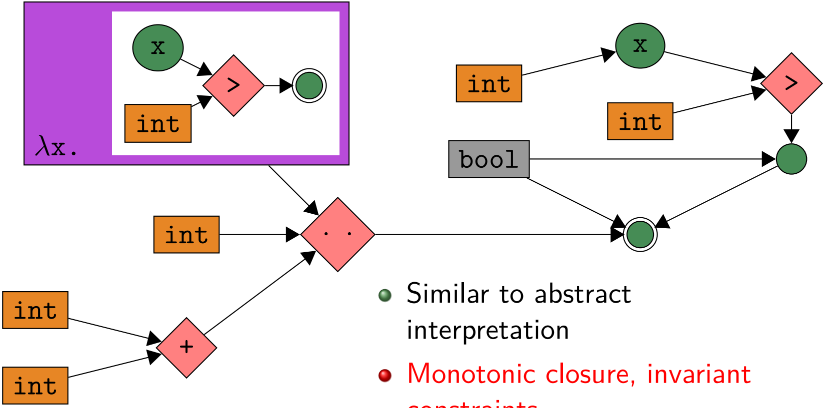
```
1 (λx. x>4) (2+1)
```



- Similar to abstract interpretation

Intuition

```
1 (λx. x>4) (2+1)
```



- Similar to abstract interpretation
- Monotonic closure, invariant constraints

Outline

- Duck Type Inference
- Conditional Reasoning
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Conditional Reasoning

- Soundness of code based on case analysis

Conditional Reasoning

- Soundness of code based on case analysis
- Common tactic in scripting

Conditional Reasoning

- Soundness of code based on case analysis
- Common tactic in scripting
- Works particularly well with duck typing

Conditional Reasoning

```
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

$\longrightarrow$ function list

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

⟶ function list

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

function list

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

⟶ function list

Conditional Reasoning

```
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

→ function list

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

⟶ function list

⟶ function

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

function list

function

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

$\longrightarrow$ function list

$\longrightarrow$ function

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

⟶ function list

⟶ function

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

$\longrightarrow$ function list

$\longrightarrow$ function

Conditional Reasoning

```python
def processHooks(tgt, data):
    if callable(tgt):
        fns = [tgt]
    else:
        fns = tgt
    for fn in fns:
        data = fn(data)
    return data
```

$\longrightarrow$ function list

$\longrightarrow$ function

- Program analyses with conditional reasoning: “path-sensitive”

How do we type it?

Typing Path-Sensitivity

- Soundness reasoning by case analysis on value

Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types

Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

Typing Path-Sensitivity

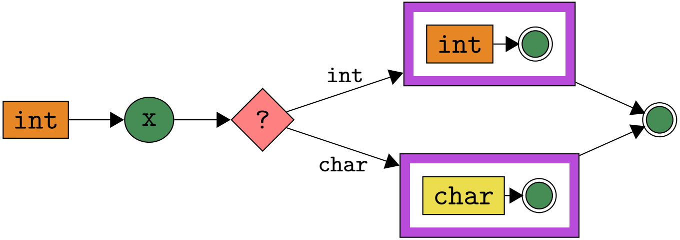
- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

```
let x = 4 in
case x of
    int -> 0
    char -> 'a'
```

Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

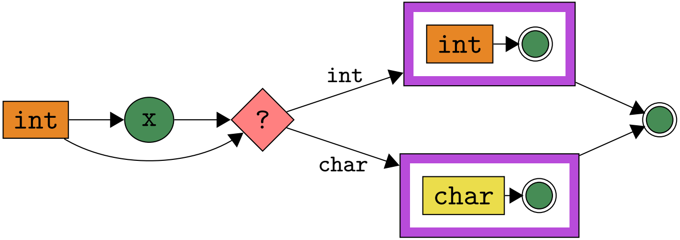
```
let x = 4 in
case x of
    int -> 0
    char -> 'a'
```



Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

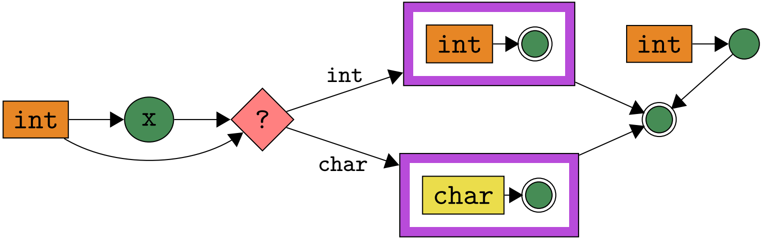
```
let x = 4 in
case x of
    int -> 0
    char -> 'a'
```



Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

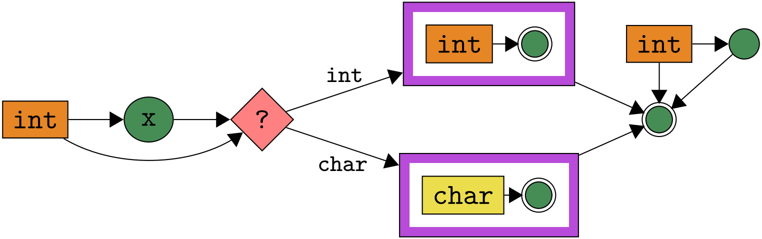
```
let x = 4 in
case x of
    int -> 0
    char -> 'a'
```



Typing Path-Sensitivity

- Soundness reasoning by case analysis on value
- Thus, use case analysis types
- Use only those constraints associated with correct case

```
let x = 4 in
case x of
     int -> 0
     char -> 'a'
```



Outline

- Duck Type Inference
- Conditional Reasoning
  - Filtered Types
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    int -> x + 1
    z -> 0
```

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
     int -> x + 1
     z -> 0
```

known to be an int!

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    int -> x + 1
    z -> 0
```

known to be an int!

- We want refinement on case analysis

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    int -> x + 1
    z -> 0
```

known to be an int!

- We want refinement on case analysis, but
  - We'd like to preserve the invariance of types

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    int -> x + 1
    z -> 0
```

known to be an int!

- We want refinement on case analysis, but
  - We'd like to preserve the invariance of types and
  - We have to keep decidability in mind

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    y * int -> y + 1
    z -> 0
```

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    y * int -> y + 1
    z -> 0
```

- $\alpha_{\mathtt{y}} \geq \alpha_{\mathtt{x}} \cap \mathtt{int}$

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    y * int -> y + 1
    z -> 0
```

- $\alpha_{\mathtt{y}} \geq \alpha_{\mathtt{x}} \cap \mathtt{int}$
- General intersections are infeasible [?]

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    y * int -> y + 1
    z -> 0
```

- $\alpha_{\mathtt{y}} \geq \alpha_{\mathtt{x}} \cap \mathtt{int}$
- General intersections are infeasible [?]
- Filtered types: $\tau \cap \pi \cap \pi \cap \ldots$

Filtered Types

```
let x = (if somebool then 4 else 'z') in
case x of
    y * int -> y + 1
    z -> 0
```

- $\alpha_{\mathtt{y}} \geq \alpha_{\mathtt{x}} \cap \mathtt{int}$
- General intersections are infeasible [**?**]
- Filtered types: $\tau \cap \pi \cap \pi \cap \ldots$
- Also includes negation (top level only): $\alpha_{\mathtt{z}} \geq 1 - \mathtt{int}$

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

$\mathbf{fun} \cup [\mathbf{fun}]$

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

$\mathbf{fun} \cup [\mathbf{fun}]$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap \mathbf{fun}$

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

$\mathbf{fun} \cup [\mathbf{fun}]$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap \mathbf{fun} = \mathbf{fun}$

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

$\mathbf{fun} \cup [\mathbf{fun}]$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap \mathbf{fun} = \mathbf{fun}$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap (1 - \mathbf{fun})$

Filtered Types

```
def processHooks(tgt, data):
    let fns =
        case tgt of
            f * fun -> [f]
            x -> x
    for fn in fns:
        fn(data)
    return data
```

$\mathbf{fun} \cup [\mathbf{fun}]$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap \mathbf{fun} = \mathbf{fun}$

$(\mathbf{fun} \cup [\mathbf{fun}]) \cap (1 - \mathbf{fun}) = [\mathbf{fun}]$

Outline

- Duck Type Inference
- Conditional Reasoning
    - Filtered Types
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Contextual Reasoning

```
def strange(b):
    if b: return (lambda n: n+1)
    else: return 4
f = strange(True)
x = strange(False)
print f(x)
```

- `f` is a function from integers to integers

Contextual Reasoning

```python
def strange(b):
    if b: return (lambda n: n+1)
    else: return 4
f = strange(True)
x = strange(False)
print f(x)
```

- `f` is a function from integers to integers
- `x` is a single integer value

Contextual Reasoning

```python
def strange(b):
    if b: return (lambda n: n+1)
    else: return 4
f = strange(True)
x = strange(False)
print f(x)
```

- `f` is a function from integers to integers
- `x` is a single integer value
- Return types are different based on invocation context

Contextual Reasoning

```
def strange(b):
    if b: return (lambda n: n+1)
    else: return 4
f = strange(True)
x = strange(False)
print f(x)
```

- `f` is a function from integers to integers
- `x` is a single integer value
- Return types are different based on invocation context
- Program analyses with contextual reasoning: "context-sensitive"

How do we type it?

Context Sensitivity

- Let-bound polymorphism
- Existing program analyses [?]
- Conditional constraints [?]

Context Sensitivity

- Let-bound polymorphism (too weak)
- Existing program analyses [**?**]
- Conditional constraints [**?**]

Context Sensitivity

- Let-bound polymorphism (too weak)
- Existing program analyses [**?**] (too brittle, not suited)
- Conditional constraints [**?**]

Context Sensitivity

- Let-bound polymorphism (too weak)
- Existing program analyses [**?**] (too brittle, not suited)
- Conditional constraints [**?**] (too coarse)

Context Sensitivity

- Let-bound polymorphism (too weak)
- Existing program analyses [**?**] (too brittle, not suited)
- Conditional constraints [**?**] (too coarse)
- Call-site polymorphism [**?**, **?**]

Call-Site Polymorphism

- **All** functions inferred polymorphic types

Call-Site Polymorphism

- **All** functions inferred polymorphic types
- Polyinstantiation at call sites

Call-Site Polymorphism

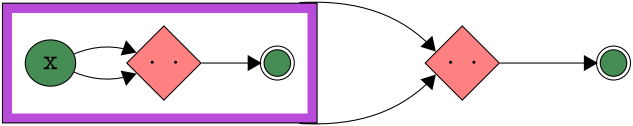
- **All** functions inferred polymorphic types
- Polyinstantiation at call sites
- Expressiveness:

```
let f x =
  let g y = y in
  g;;
let h = f();;
let q = (h 1, h 'z');;
```

```
def f() = fun y -> y
def h = f()
def q = (h 1, h 'z')
```

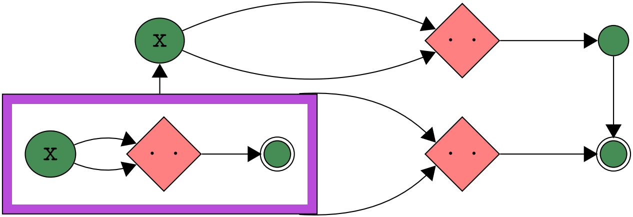
Bounding Call-Site Polymorphism

```
(λx. x x) (λx. x x)
```



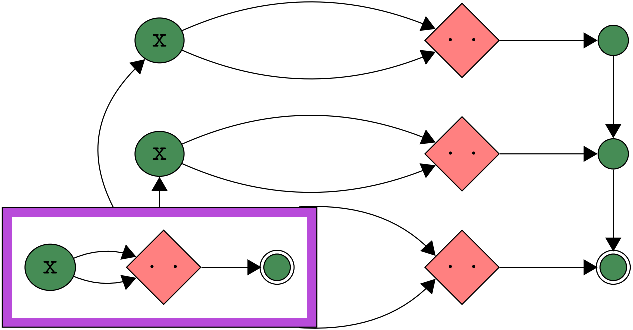
Bounding Call-Site Polymorphism

```
(λx. x x) (λx. x x)
```



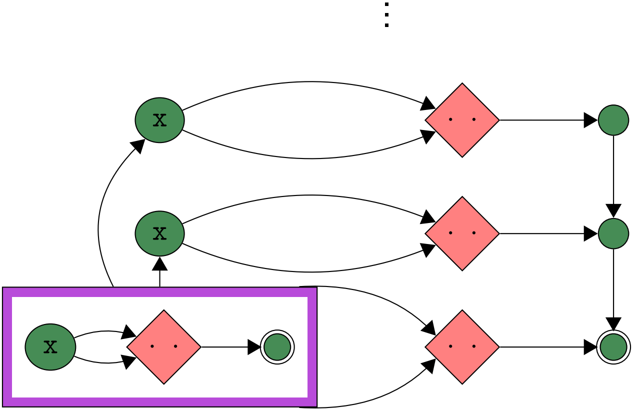
Bounding Call-Site Polymorphism

```
(λx. x x) (λx. x x)
```

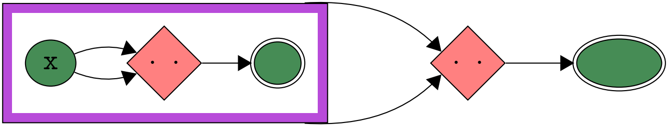


Bounding Call-Site Polymorphism

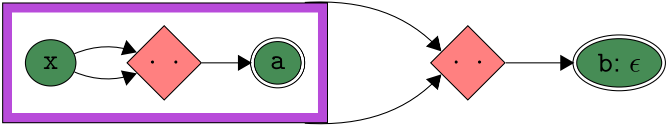
```
(λx. x x) (λx. x x)
```



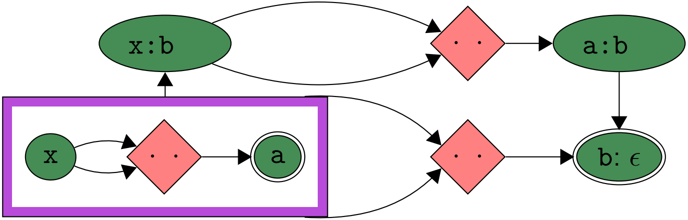
Bounding Call-Site Polymorphism



Bounding Call-Site Polymorphism

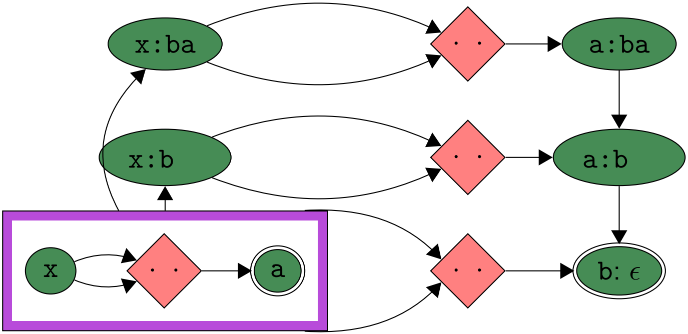


Bounding Call-Site Polymorphism

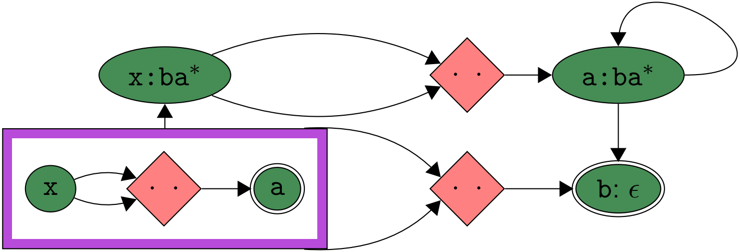


Bounding Call-Site Polymorphism

⋮

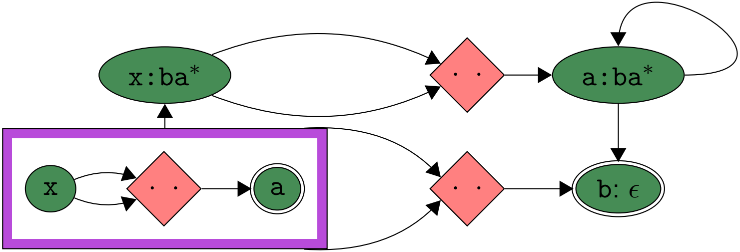


Bounding Call-Site Polymorphism



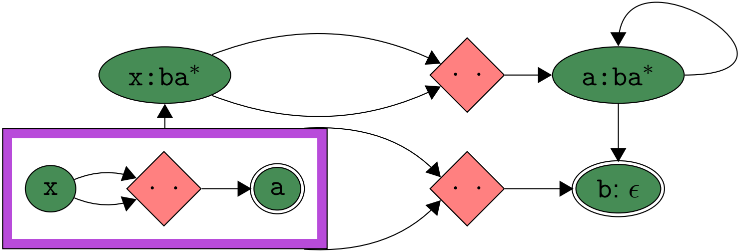
Bounding Call-Site Polymorphism

- Variables named by program point and **contour**



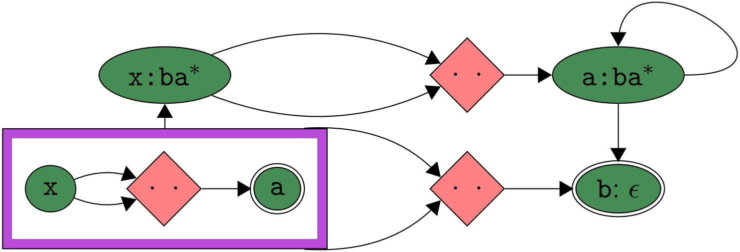
Bounding Call-Site Polymorphism

- Variables named by program point and **contour**
- Contour: restricted regular expression



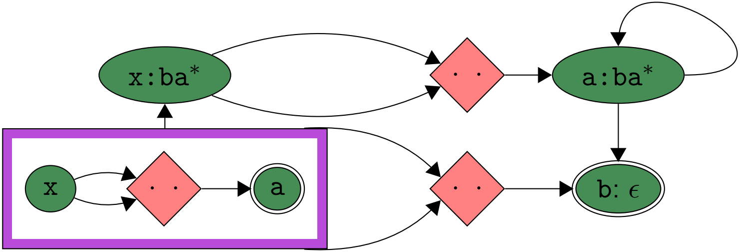
Bounding Call-Site Polymorphism

- Variables named by program point and **contour**
- Contour: restricted regular expression
  - Concatenation of literals and Kleene closures over literals



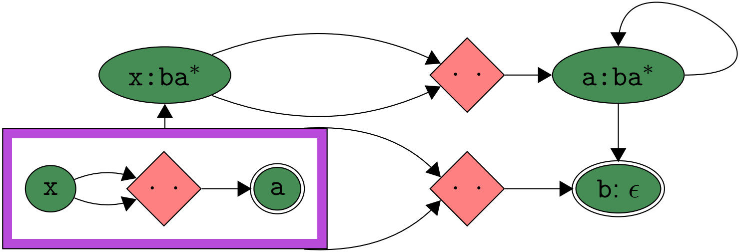
Bounding Call-Site Polymorphism

- Variables named by program point and **contour**
- Contour: restricted regular expression
  - Concatenation of literals and Kleene closures over literals
  - Each literal token may appear at most once



Bounding Call-Site Polymorphism

- Variables named by program point and **contour**
- Contour: restricted regular expression
  - Concatenation of literals and Kleene closures over literals
  - Each literal token may appear at most once
- Contours are joined at recursion



Outline

- Duck Type Inference
- Conditional Reasoning
  - Filtered Types
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Flexible Data Model

Scripting languages have flexible data models:

```python
class MyClass:
    def msg(self):
        print "Foo"
obj = MyClass()
obj.msg() # Prints "Foo"
obj.msg = types.MethodType(lambda s: print "Bar", obj)
obj.msg() # Prints "Bar"
```

Flexible Data Model

Scripting languages have flexible data models:

```
class MyClass:
    def msg(self):
        print "Foo"
obj = MyClass()
obj.msg() # Prints "Foo"
obj.msg = types.MethodType(lambda s: print "Bar", obj)
obj.msg() # Prints "Bar"
```

- Mutating/adding methods
- Multiple inheritance
- Dynamic mixins
- etc.

How do we type it?

Encoding

- Reduces complexity of type system

Encoding

- Reduces complexity of type system
- Ensures consistency in reasoning

Encoding

- Reduces complexity of type system
- Ensures consistency in reasoning
- No artificial distinctions (e.g. strategy object $\cong$ function)

Encoding

- Reduces complexity of type system
- Ensures consistency in reasoning
- No artificial distinctions (e.g. strategy object $\cong$ function)
- Programmer can work "under the hood" as necessary

Encoding

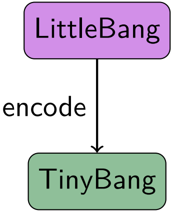
LittleBang

Encoding

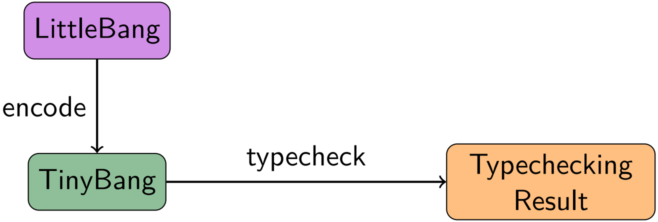
LittleBang

TinyBang

Encoding



Encoding


LittleBang
encode
TinyBang
typecheck
Typechecking Result

TinyBang Language Features

- Primitives (e.g. 5)

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
  - `fun int -> 0` matches any integer

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
  - `fun int -> 0` matches any integer
  - `fun x:int -> x` is integer identity

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
  - `fun int -> 0` matches any integer
  - `fun x:int -> x` is integer identity
  - `(fun x:int -> x + 1) 4` returns 5

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `‘A 5`)
- Partial functions
  - `fun int -> 0` matches any integer
  - `fun x:int -> x` is integer identity
  - `(fun x:int -> x + 1) 4` returns 5
  - `(fun x -> x + 1) "badness"` crashes

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
  - `fun int -> 0` matches any integer
  - `fun x:int -> x` is integer identity
  - `(fun x:int -> x + 1) 4` returns 5
  - `(fun x -> x + 1) "badness"` crashes
  - `(fun int -> 0) "badness"` also crashes

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [**?**]

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [**?**]
  - "Normal" records: `{Name="Ann", Age=43}`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5``)
- Partial functions
- Onions: type-indexed records [**?**]
  - "Normal" records: `{Name="Ann", Age=43}`
  - `4 & "word"` is an onion with an `int` and a `str`

TinyBang Language Features

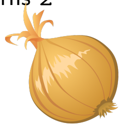
- Primitives (e.g. `5`)
- Labels (e.g. `‘A 5`)
- Partial functions
- Onions: type-indexed records [**?**]
  - “Normal” records: `{Name="Ann", Age=43}`
  - `4 & "word"` is an onion with an `int` and a `str`
  - Onions of labels: records/structs
    - `‘Name "Ann" & ‘Age 43`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `‘A 5`)
- Partial functions
- Onions: type-indexed records [**?**]
  - “Normal” records: `{Name="Ann", Age=43}`
  - `4 & "word"` is an onion with an `int` and a `str`
  - Onions of labels: records/structs
    - `‘Name "Ann" & ‘Age 43`
  - Functions match onions by type
    - `(fun ‘Name n -> n) (‘Name "Ann" & ‘Age 43)` returns `"Ann"`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `‘A 5`)
- Partial functions
- Onions: type-indexed records [?]
  - “Normal” records: `{Name="Ann", Age=43}`
  - `4 & "word"` is an onion with an `int` and a `str`
  - Onions of labels: records/structs
    - `‘Name "Ann" & ‘Age 43`
  - Functions match onions by type
    - `(fun ‘Name n -> n) (‘Name "Ann" & ‘Age 43)` returns `"Ann"`
  - **Leftmost** onion element wins
    - `(fun ‘A x -> x) (‘A 2 & ‘B 3 & ‘A 4)` returns 2



TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [**?**]
- Onion dispatch: leftmost function wins

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [?]
- Onion dispatch: leftmost function wins
  - `(int -> 0) & (char -> 'a')` matches `int` or `char` onions

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [?]
- Onion dispatch: leftmost function wins
  - `(int -> 0) & (char -> 'a')` matches `int` or `char` onions
  - `((int -> 0) & (char -> 'a')) (5)` $\Longrightarrow$ `0`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [**?**]
- Onion dispatch: leftmost function wins
  - `(int -> 0) & (char -> 'a')` matches `int` or `char` onions
  - `((int -> 0) & (char -> 'a')) (5)` $\Longrightarrow$ `0`
  - `((int -> 0) & (char -> 'a')) ('z')` $\Longrightarrow$ `'a'`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [?]
- Onion dispatch: leftmost function wins
  - `(int -> 0) & (char -> 'a')` matches `int` or `char` onions
  - `((int -> 0) & (char -> 'a')) (5)` $\Longrightarrow$ `0`
  - `((int -> 0) & (char -> 'a')) ('z')` $\Longrightarrow$ `'a'`
  - `((int -> 0) & (char -> 'a')) (5 & 'z')` $\Longrightarrow$ `0`

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `‘A 5`)
- Partial functions
- Onions: type-indexed records [**?**]
- Onion dispatch: leftmost function wins
  - `(int -> 0) & (char -> ’a’)` matches `int` or `char` onions
  - `((int -> 0) & (char -> ’a’)) (5)` $\Longrightarrow$ `0`
  - `((int -> 0) & (char -> ’a’)) (’z’)` $\Longrightarrow$ `’a’`
  - `((int -> 0) & (char -> ’a’)) (5 & ’z’)` $\Longrightarrow$ `0`
  - `((int -> 0) & (char -> ’a’)) (‘A 5 & ‘B 4)` crashes

TinyBang Language Features

- Primitives (e.g. `5`)
- Labels (e.g. `` `A 5 ``)
- Partial functions
- Onions: type-indexed records [**?**]
- Onion dispatch: leftmost function wins
- That's it!

Why These Features?

Together, onions and partial functions can encode:

Why These Features?

Together, onions and partial functions can encode:

- Records
- Conditionals (on `'True ()` and `'False ()`)
- Variant-based objects
- Operator overloading
- Classes, inheritance, subclasses, etc.
- Mixins, dynamic functional object extension
- First-class cases
- Optional arguments
- etc.

Why These Features?

Together, onions and partial functions can encode:

- Records
- Conditionals (on `'True ()` and `'False ()`)
- Variant-based objects
- Operator overloading
- Classes, inheritance, subclasses, etc.
- Mixins, dynamic functional object extension
- First-class cases
- Optional arguments
- etc.

And we can type them!

Encoding Example

LittleBang

```
def inc(x,y=1):
    return x + y
print inc(3,5)
print inc(7)
```

Encoding Example

LittleBang

```
def inc(x,y=1):
    return x + y
print inc(3,5)
print inc(7)
```

TinyBang

```
let inc = fun a * `x x ->
            let y = ( (fun `y v -> v)
                    & (fun _ -> 1)     ) a
            in x + y
print (inc (`x 3 & `y 5))
print (inc (`x 7))
```

Working Under the Hood

```
let obj = if somebool
            then object {
                   m(s:str) = print(s)
                 }
            else object {
                   inc(x:int) = x + 1
                 }
obj.m("hello") # static type error if no m

def dynamic(msg): throw MethodError()
(obj & dynamic).m("hello") # exception if no m
```

Working Under the Hood

```
let obj = if somebool
            then
              fun (`msg `m () * `s (s * str)) -> print(s)

            else
              fun (`msg `inc () * `x (x * int)) -> x + 1

obj (`msg `m () & `s "hello") # static type error if no m

let dynamic = fun msg -> throw MethodError()
(obj & dynamic) (`msg `m () & `s "hello") # exception if no m
```

What We Don't Get

TinyBang is:

- Aware of conditionals
- Aware of calling context
- Flexible on data

What We Don't Get

TinyBang is:

- Aware of conditionals
- Aware of calling context
- Flexible on data

TinyBang isn't:

- Perfect on recursion
- Modular
- Aware of time

What We Don't Get

TinyBang is:

- Aware of conditionals
- Aware of calling context
- Flexible on data

TinyBang isn't:

- Perfect on recursion
- Modular
- Aware of time

```
def c = 4
c = "hello"
(str -> "") c
```

What We Don't Get

TinyBang is:

- Aware of conditionals
- Aware of calling context
- Flexible on data

TinyBang isn't:

- Perfect on recursion
- Modular
- Aware of time

```
def c = 4
c = "hello"
(str -> "") c
```

- No typestate

What We Don't Get

TinyBang is:

- Aware of conditionals
- Aware of calling context
- Flexible on data

TinyBang isn't:

- Perfect on recursion
- Modular
- Aware of time

```
def c = 4
c = "hello"
(str -> "") c
```

- No typestate
- No mutable monkeypatching (use functional extension!)

Outline

- Duck Type Inference
- Conditional Reasoning
  - Filtered Types
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

Typechecking Process

To typecheck a LittleBang program:

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form

Typechecking Process

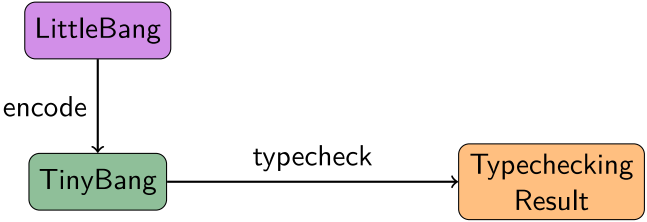
To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
    - Like assembly language: lots of small steps

Typechecking Process

To typecheck a LittleBang program:

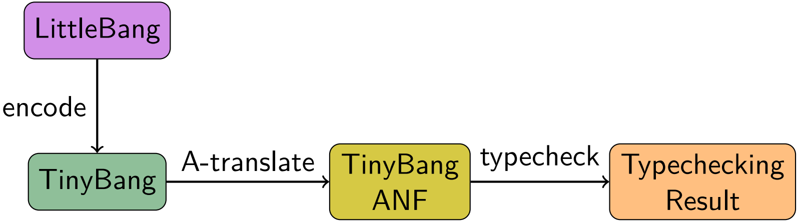
- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
    - Like assembly language: lots of small steps



Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
    - Like assembly language: lots of small steps



Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\mathtt{x}} \geq \mathtt{int}$

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\mathtt{x}} \geq \mathtt{int}$
- Perform deductive logical closure

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\texttt{x}} \geq \texttt{int}$
- Perform deductive logical closure
  - Reach every conclusion we can from what we've learned

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\mathtt{x}} \geq \mathtt{int}$
- Perform deductive logical closure
  - Reach every conclusion we can from what we've learned
  - Similar to running the program

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\mathtt{x}} \geq \mathtt{int}$
- Perform deductive logical closure
  - Reach every conclusion we can from what we've learned
  - Similar to running the program
- Check consistency of result

Typechecking Process

To typecheck a LittleBang program:

- Encode LittleBang operations into TinyBang
- *A-translate* TinyBang into A-normal form
  - Like assembly language: lots of small steps
- Derive type constraints from program
  - e.g. `x=5` implies $\alpha_{\mathtt{x}} \geq \mathtt{int}$
- Perform deductive logical closure
  - Reach every conclusion we can from what we've learned
  - Similar to running the program
- Check consistency of result
  - If the program is bad, we will reach a false conclusion

Synthesis

- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \texttt{`A}\, \alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `` `A int ``

Synthesis

- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \texttt{`A}\,\alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `` `A int ``
- Corresponds to A-normalized expression grammar

Synthesis

- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \texttt{`A}\, \alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `` `A int ``
- Corresponds to A-normalized expression grammar
- In general, correspondence between evaluation and type systems
  - value $\leftrightarrow$ type
  - clause $\leftrightarrow$ constraint
  - expression $\leftrightarrow$ constraint type
  - operational semantics rule $\leftrightarrow$ constraint closure rule
  - evaluation $\leftrightarrow$ constraint closure

Synthesis

- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \texttt{`A}\, \alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `` `A int ``
- Corresponds to A-normalized expression grammar
- In general, correspondence between evaluation and type systems
  - value $\leftrightarrow$ type
  - clause $\leftrightarrow$ constraint
  - expression $\leftrightarrow$ constraint type
  - operational semantics rule $\leftrightarrow$ constraint closure rule
  - evaluation $\leftrightarrow$ constraint closure
- Forms adjoint (as in abtract interpretation)

Synthesis

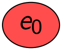
- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \text{`A}\, \alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `A int
- Corresponds to A-normalized expression grammar
- In general, correspondence between evaluation and type systems
  - value $\leftrightarrow$ type
  - clause $\leftrightarrow$ constraint
  - expression $\leftrightarrow$ constraint type
  - operational semantics rule $\leftrightarrow$ constraint closure rule
  - evaluation $\leftrightarrow$ constraint closure
- Forms adjoint (as in abtract interpretation)
- Proof of soundness: simulation rather than progress & preservation

Synthesis

- TinyBang type grammar is shallow (unusual)
  - e.g. $\alpha_1 \backslash \{ \texttt{`A}\, \alpha_2 \leq \alpha_1, \texttt{int} \leq \alpha_2 \}$
  - not `` `A int``
- Corresponds to A-normalized expression grammar
- In general, correspondence between evaluation and type systems
  - value $\leftrightarrow$ type
  - clause $\leftrightarrow$ constraint
  - expression $\leftrightarrow$ constraint type
  - operational semantics rule $\leftrightarrow$ constraint closure rule
  - evaluation $\leftrightarrow$ constraint closure
- Forms adjoint (as in abtract interpretation)
- Proof of soundness: simulation rather than progress & preservation
- Difference: all facts gathered here are invariant

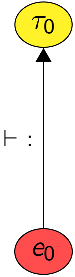
Proof of Soundness

Progress & Preservation



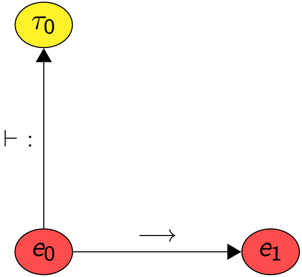
Proof of Soundness

Progress & Preservation



Proof of Soundness

Progress & Preservation



Proof of Soundness

Progress & Preservation

$$\begin{array}{ccc} \tau_0 & & \tau_1 \\ \uparrow{\scriptstyle \vdash :} & & \uparrow{\scriptstyle \vdash :} \\ e_0 & \xrightarrow{\longrightarrow} & e_1 \end{array}$$

Proof of Soundness

Progress & Preservation

$$
\begin{tikzcd}
\tau_0 \arrow[r, dashed, "\geq"] & \tau_1 \\
e_0 \arrow[u, "\vdash :"] \arrow[r, "\longrightarrow"] & e_1 \arrow[u, "\vdash :"']
\end{tikzcd}
$$

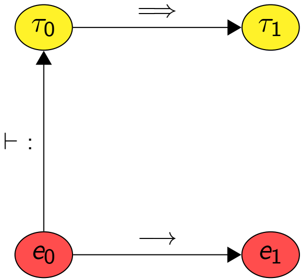
Proof of Soundness

Progress & Preservation

$$\begin{tikzcd}
\tau_0 \arrow[rr, dashed, "\geq"] & & \tau_1 \arrow[rr, dashed] & & \cdots \\
e_0 \arrow[u, "\vdash :"] \arrow[rr, "\longrightarrow"] & & e_1 \arrow[u, "\vdash :"] \arrow[rr] & & \cdots
\end{tikzcd}$$

Proof of Soundness

Simulation



Proof of Soundness

Simulation

$$
\begin{tikzcd}
\tau_0 \arrow[r, "\Longrightarrow"] & \tau_1 \\
e_0 \arrow[u, "\vdash :"] \arrow[r, "\longrightarrow"] & e_1 \arrow[u, dashed, "\vdash :"]
\end{tikzcd}
$$

Proof of Soundness

Simulation

$$
\begin{tikzcd}
\tau_0 \arrow[r, "\Longrightarrow"] & \tau_1 \arrow[r] & \cdots \\
e_0 \arrow[u, "\vdash :"] \arrow[r, "\longrightarrow"] & e_1 \arrow[u, dashed, "\vdash :"] \arrow[r] & \cdots
\end{tikzcd}
$$

Not Shown Here

- Union alignment invariants

Not Shown Here

- Union alignment invariants
- The interesting encodings (dynamic mixins, subclasses, etc.)

Not Shown Here

- Union alignment invariants
- The interesting encodings (dynamic mixins, subclasses, etc.)
- Novel object extension properties [**?**]

Outline

- Duck Type Inference
- Conditional Reasoning
  - Filtered Types
- Contextual Reasoning
- Flexible Data Model
- Formal Development
- What's Left?
- Conclusion

What's Left?

- Current Research

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch
- Future Research

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch
- Future Research
  - Type error processing

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch
- Future Research
  - Type error processing
  - Modularity

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch
- Future Research
  - Type error processing
  - Modularity
  - User-specified encodings / language features

What's Left?

- Current Research
  - Function patterns (e.g. `int ~> int`)
    - Encode type signatures
    - First overloading of higher-order functions
    - Potentially solves modularity issues
  - Chronological constraints
    - Enable flow-sensitivity (awareness of time)
    - Possibly improves typechecking performance
  - Layout calculus
    - Hari's dissertation focus
    - Efficient onion memory layout and dispatch
- Future Research
  - Type error processing
  - Modularity
  - User-specified encodings / language features
  - Type-aware script metaprogramming

Outline

Conclusions

Conclusions

We can build a typed scripting language from scratch.

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing
- Feature encodings keep type system simple

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing
- Feature encodings keep type system simple
  - Asymmetry of onions allows encoding of subclasses, overloading, etc.

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing
- Feature encodings keep type system simple
  - Asymmetry of onions allows encoding of subclasses, overloading, etc.
- Path-sensitivity (via onion dispatch) enables case-based reasoning

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing
- Feature encodings keep type system simple
  - Asymmetry of onions allows encoding of subclasses, overloading, etc.
- Path-sensitivity (via onion dispatch) enables case-based reasoning
- Context-sensitivity (via call-site polymorphism) allows complex, intuitive soundness arguments

Conclusions

We can build a typed scripting language from scratch.

- Subtype constraints allow for duck typing
- Feature encodings keep type system simple
  - Asymmetry of onions allows encoding of subclasses, overloading, etc.
- Path-sensitivity (via onion dispatch) enables case-based reasoning
- Context-sensitivity (via call-site polymorphism) allows complex, intuitive soundness arguments
- Exploit connection to abstract interpretation while remaining in type theory

Thanks!

- Scott F. Smith (advisor)
- Alexander Rozenstheyn (collaborator)
- Pottayil Harisanker Menon (collaborator)
- Rebekah Palmer (wife, best friend)
- JHU Computer Science Department
- All those people who did all that research

Questions?

Bibliography

Typechecking Example

Typechecking by Example

First, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

Typechecking by Example

First, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

Then, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 'z' else 1)
```

Encoding and A-normalization

Preparing to Typecheck

LittleBang

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

Preparing to Typecheck

LittleBang

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

TinyBang

```
let b = ... in
1 + 2 + ( ((`True () -> 5)
          &(`False () -> 1)) b)
```

Preparing to Typecheck

LittleBang

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

TinyBang

```
let b = ... in
1 + 2 + ( ((`True () -> 5)
           &(`False () -> 1)) b)
```

TinyBang ANF

```
b = ...;
x1 = 1;
x2 = 2;
x3 = + x1 x2;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ...;
x1 = 1;
x2 = 2;
x3 = + x1 x2;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = `False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = `False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
r2’ = 1; x7 = r2’;
x8 = + x3 x7;
```

An Aside: Execution

```
b = `False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
r2' = 1; x7 = r2';
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
r2’ = 1; x7 = 1;
x8 = + x3 x7;
```

An Aside: Execution

```
b = ‘False ...;
x1 = 1;
x2 = 2;
x3 = 3;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
r2’ = 1; x7 = 1;
x8 = 4;
```

Initial Alignment

```
b = ...;
x1 = 1;
x2 = 2;
x3 = + x1 x2;
x4 = { p1 = (); p2 = `True p1 } -> { r1 = 5 };
x5 = { p3 = (); p4 = `False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

Initial Alignment

```
b = ...;
x1 = int;
x2 = int;
x3 = + x1 x2;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = int };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = int };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

- Replace primitive data with its type

Initial Alignment

$$\left\{\begin{array}{l}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \qquad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, \\
\alpha_{\texttt{x1}} \geq \texttt{int}\,, \\
\alpha_{\texttt{x2}} \geq \texttt{int}\,, \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \texttt{`True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{int}\}, \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \texttt{`False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}}\,\&\,\alpha_{\texttt{x5}}, \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}},
\end{array}\right\}$$

- Replace primitive data with its type
- Convert to constraints (for e.g. duck typing)

Perform Constraint Closure

$$\left\{\begin{array}{l}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\,\&\,\alpha_{\mathtt{x5}}, \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \\
\\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{l}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\,\&\,\alpha_{\mathtt{x5}}, \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \\
\\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \quad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \texttt{`True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{int}\}, & \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \texttt{`False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}} \,\&\, \alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, & \\
& \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\ \&\ \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\ \&\ \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \qquad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \texttt{`True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{int}\}, & \alpha_{\texttt{r1'}} \geq \texttt{int}, \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \texttt{`False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}}\,\&\,\alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r1'}}, \\
 & \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}} &
\end{array}
\right\}
$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \texttt{`True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x2}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x3}} \geq \texttt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \texttt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \texttt{()}\,, \alpha_{\mathtt{p2}} \geq \texttt{`True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \texttt{int}\}, & \alpha_{\mathtt{r1'}} \geq \texttt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \texttt{()}\,, \alpha_{\mathtt{p4}} \geq \texttt{`False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \texttt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \\
\alpha_{\mathtt{x8}} \geq \texttt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\ \&\ \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\ , \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\ , \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\ \&\ \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
& \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \texttt{`True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x2}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x3}} \geq \texttt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \texttt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \texttt{()}\,, \alpha_{\mathtt{p2}} \geq \texttt{`True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \texttt{int}\}, & \alpha_{\mathtt{r1'}} \geq \texttt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \texttt{()}\,, \alpha_{\mathtt{p4}} \geq \texttt{`False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \texttt{int}\}, & \alpha_{\mathtt{r2'}} \geq \texttt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \texttt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\,\&\,\alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \quad \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \texttt{`True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x2}} \geq \texttt{int}, & \\
\alpha_{\mathtt{x3}} \geq \texttt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \texttt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \texttt{()}\,, \alpha_{\mathtt{p2}} \geq \texttt{`True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \texttt{int}\}, & \alpha_{\mathtt{r1'}} \geq \texttt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \texttt{()}\,, \alpha_{\mathtt{p4}} \geq \texttt{`False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \texttt{int}\}, & \alpha_{\mathtt{r2'}} \geq \texttt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \qquad \alpha_{\mathtt{x7}} \geq \texttt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \texttt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \qquad \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \to \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1}'} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \to \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r2}'} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \mathbin{\&} \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \quad \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1}'}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2}'}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}, & \alpha_{\mathtt{x8}} \geq \mathtt{int}
\end{array}
\right\}
$$

Check Consistency

$$
\left\{
\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \mathtt{`True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \mathtt{`False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \mathtt{`True}\,\alpha_{\mathtt{p1}}\} \to \{\alpha_{\mathtt{r1}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \mathtt{`False}\,\alpha_{\mathtt{p3}}\} \to \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \qquad \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}, & \alpha_{\mathtt{x8}} \geq \mathtt{int}
\end{array}
\right\}
$$

✓

Typechecking by Example

First, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

✓

Then, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 'z' else 1)
```

Initial Alignment

```
b = ...;
x1 = 1;
x2 = 2;
x3 = + x1 x2;
x4 = { p1 = (); p2 = ‘True p1 } -> { r1 = ’z’ };
x5 = { p3 = (); p4 = ‘False p3 } -> { r2 = 1 };
x6 = x4 & x5;
x7 = x6 b;
x8 = + x3 x7;
```

Initial Alignment

$$\left\{\begin{array}{l}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}\,, \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}\,, \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}},
\end{array}\right\}$$

- Same as last time, but with a `char`

Perform Constraint Closure

$$
\left\{
\begin{array}{l}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \to \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \to \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \\
\\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}
\end{array}
\right\}
$$

Perform Constraint Closure

$$\left\{\begin{array}{l}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \\
\\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \quad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}, \alpha_{\texttt{p2}} \geq \texttt{`True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{char}\}, & \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}, \alpha_{\texttt{p4}} \geq \texttt{`False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}} \,\&\, \alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, & \\
 & \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$
\left\{
\begin{array}{lr}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq +\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq ()\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq ()\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq +\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}
\right\}
$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \qquad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq +\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq ()\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq ()\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq +\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\, \&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\ \&\ \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}\right\}$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\texttt{b}} \geq \text{`}\texttt{True}\ \ldots, \qquad \alpha_{\texttt{b}} \geq \text{`}\texttt{False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \text{`}\texttt{True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{char}\}, & \alpha_{\texttt{r1'}} \geq \texttt{char}, \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \text{`}\texttt{False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}}\,\&\,\alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r1'}}, \\
 & \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}} &
\end{array}
\right\}
$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\ , \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\ , \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} &
\end{array}
\right\}
$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \qquad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \texttt{`True}\,\alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{char}\}, & \alpha_{\texttt{r1'}} \geq \texttt{char}, \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \texttt{`False}\,\alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \alpha_{\texttt{r2'}} \geq \texttt{int}, \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}}\,\&\,\alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r1'}}, \\
 & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r2'}}, \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}} &
\end{array}
\right\}
$$

Perform Constraint Closure

$$
\left\{
\begin{array}{lll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\,\&\,\alpha_{\mathtt{x5}}, & & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \quad \alpha_{\mathtt{x7}} \geq \mathtt{char}, & & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} & &
\end{array}
\right\}
$$

Perform Constraint Closure

$$\left\{\begin{array}{llll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, & \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & & & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & & & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & & & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & & & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \mathtt{char}, & \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & & & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} & & &
\end{array}\right\}$$

Perform Constraint Closure

$$\left\{\begin{array}{llll}
\alpha_{\mathtt{b}} \geq \text{‘}\mathtt{True}\ \ldots, & \alpha_{\mathtt{b}} \geq \text{‘}\mathtt{False}\ \ldots, & & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & & & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{‘}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & & & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{‘}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & & & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & & & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \mathtt{char}, & \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & & & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}} & & &
\end{array}\right\}$$

Perform Constraint Closure

$$
\left\{
\begin{array}{ll}
\alpha_{\texttt{b}} \geq \texttt{`True}\ \ldots, \quad \alpha_{\texttt{b}} \geq \texttt{`False}\ \ldots, & \\
\alpha_{\texttt{x1}} \geq \texttt{int}, & \\
\alpha_{\texttt{x2}} \geq \texttt{int}, & \\
\alpha_{\texttt{x3}} \geq \texttt{+}\ \alpha_{\texttt{x1}}\ \alpha_{\texttt{x2}}, & \alpha_{\texttt{x3}} \geq \texttt{int}, \\
\alpha_{\texttt{x4}} \geq \{\alpha_{\texttt{p1}} \geq \texttt{()}\,, \alpha_{\texttt{p2}} \geq \texttt{`True}\, \alpha_{\texttt{p1}}\} \rightarrow \{\alpha_{\texttt{r1}} \geq \texttt{char}\}, & \alpha_{\texttt{r1'}} \geq \texttt{char}, \\
\alpha_{\texttt{x5}} \geq \{\alpha_{\texttt{p3}} \geq \texttt{()}\,, \alpha_{\texttt{p4}} \geq \texttt{`False}\, \alpha_{\texttt{p3}}\} \rightarrow \{\alpha_{\texttt{r2}} \geq \texttt{int}\}, & \alpha_{\texttt{r2'}} \geq \texttt{int}, \\
\alpha_{\texttt{x6}} \geq \alpha_{\texttt{x4}} \,\&\, \alpha_{\texttt{x5}}, & \\
\alpha_{\texttt{x7}} \geq \alpha_{\texttt{x6}}\ \alpha_{\texttt{b}}, \quad \alpha_{\texttt{x7}} \geq \texttt{char}, \quad \alpha_{\texttt{x7}} \geq \texttt{int}, & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r1'}}, \\
 & \alpha_{\texttt{x7}} \geq \alpha_{\texttt{r2'}}, \\
\alpha_{\texttt{x8}} \geq \texttt{+}\ \alpha_{\texttt{x3}}\ \alpha_{\texttt{x7}}, & \alpha_{\texttt{x8}} \geq \texttt{int}
\end{array}
\right\}
$$

Check Consistency

$$
\left\{
\begin{array}{llll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, & \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & & & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & & & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\multicolumn{3}{l}{\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\ , \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\, \alpha_{\mathtt{p1}}\} \rightarrow \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\},} & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\multicolumn{3}{l}{\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\ , \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\, \alpha_{\mathtt{p3}}\} \rightarrow \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\},} & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}}\, \&\, \alpha_{\mathtt{x5}}, & & & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, & \alpha_{\mathtt{x7}} \geq \mathtt{char}, & \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & & & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}, & & & \alpha_{\mathtt{x8}} \geq \mathtt{int}
\end{array}
\right\}
$$

Check Consistency

$$
\left\{
\begin{array}{ll}
\alpha_{\mathtt{b}} \geq \text{`}\mathtt{True}\ \ldots, \quad \alpha_{\mathtt{b}} \geq \text{`}\mathtt{False}\ \ldots, & \\
\alpha_{\mathtt{x1}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x2}} \geq \mathtt{int}, & \\
\alpha_{\mathtt{x3}} \geq \mathtt{+}\ \alpha_{\mathtt{x1}}\ \alpha_{\mathtt{x2}}, & \alpha_{\mathtt{x3}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x4}} \geq \{\alpha_{\mathtt{p1}} \geq \mathtt{()}\,, \alpha_{\mathtt{p2}} \geq \text{`}\mathtt{True}\,\alpha_{\mathtt{p1}}\} \to \{\alpha_{\mathtt{r1}} \geq \mathtt{char}\}, & \alpha_{\mathtt{r1'}} \geq \mathtt{char}, \\
\alpha_{\mathtt{x5}} \geq \{\alpha_{\mathtt{p3}} \geq \mathtt{()}\,, \alpha_{\mathtt{p4}} \geq \text{`}\mathtt{False}\,\alpha_{\mathtt{p3}}\} \to \{\alpha_{\mathtt{r2}} \geq \mathtt{int}\}, & \alpha_{\mathtt{r2'}} \geq \mathtt{int}, \\
\alpha_{\mathtt{x6}} \geq \alpha_{\mathtt{x4}} \,\&\, \alpha_{\mathtt{x5}}, & \\
\alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{x6}}\ \alpha_{\mathtt{b}}, \quad \alpha_{\mathtt{x7}} \geq \mathtt{char}, \quad \alpha_{\mathtt{x7}} \geq \mathtt{int}, & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r1'}}, \\
 & \alpha_{\mathtt{x7}} \geq \alpha_{\mathtt{r2'}}, \\
\alpha_{\mathtt{x8}} \geq \mathtt{+}\ \alpha_{\mathtt{x3}}\ \alpha_{\mathtt{x7}}, & \alpha_{\mathtt{x8}} \geq \mathtt{int}
\end{array}
\right\}
$$

✗

Typechecking by Example

First, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 5 else 1)
```

✓

Then, we will typecheck this program:

```
let b = ...
1 + 2 + (if b then 'z' else 1)
```

✗