Neural Type Prediction with Search-based Validation

Michael Pradel[1], **Georgios Gousios**[2], **Jason Liu**[3], **Satish Chandra**[3]

[1] University of Stuttgart, [2] TU Delft, [3] Facebook

FSE 2020

Types in Dynamic PLs

- **Dynamically typed languages:**
  Extremely popular
- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support
- **Gradual types to the rescue**

Types in Dynamic PLs

- **Dynamically typed languages:**
  Extremely popular
- **Lack of type annotations:**
  - Type errors
  - Hard-to-understand APIs
  - Poor IDE support
- **Gradual types to the rescue**

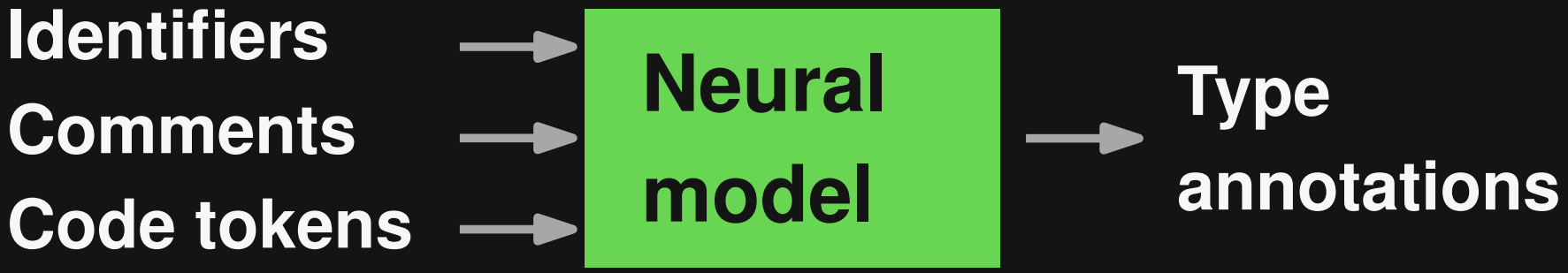
But: Annotating types is painful

How to Add Type Annotations?

- **Option 1: Static type inference**
  - Guarantees type correctness, but very limited
- **Option 2: Dynamic type inference**
  - Depends on inputs and misses types
- **Option 3: Probabilistic type prediction**
  - Models learned from existing type annotations

Probabilistic Type Prediction

E.g., neural model to predict types



Popular models:

- *Deep Learning Type Inference*, FSE'18
- *NL2Type: Inferring JavaScript Function Types from Natural Language Information*, ICSE'19

Challenges

- **Imprecision**
  - Some predictions are wrong
  - Developers must decide which suggestions to follow
- **Combinatorial explosion**
  - For each missing type: One or more suggestions
  - Exploring all combinations: Practically impossible

Example

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Example

```
def find_match(color):
    """
    Args:
      color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
      if color == candidate:
        return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
  """
  Args:
    color (s                    d return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Top-most predictions:
Type errors

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
    """
    Args:
      color (s                    d return
    """
    candidates = get_colors()
    for candidate in candidates:
      if color == candidate:
        return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

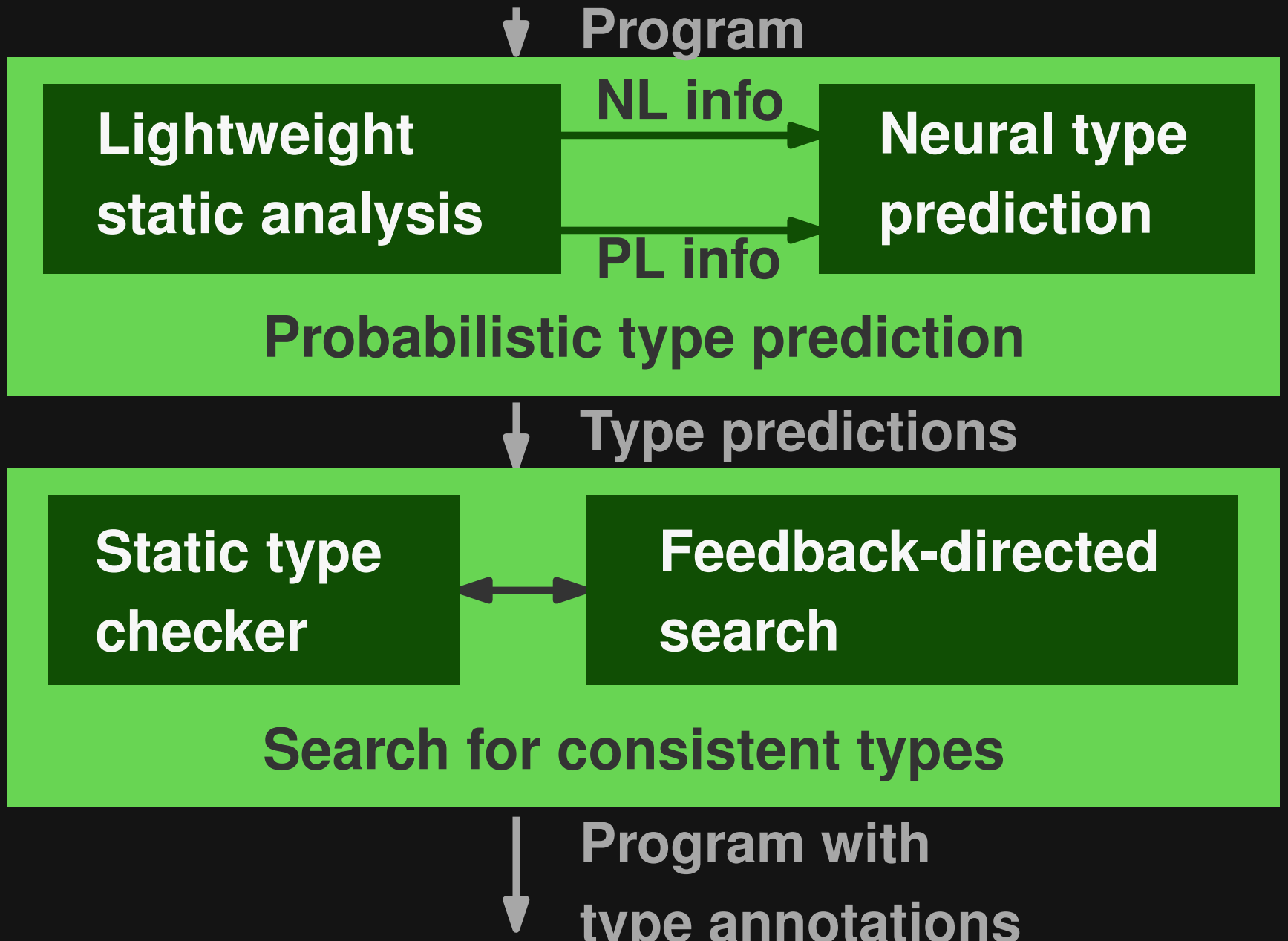
Correct predictions

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Overview of TypeWriter



Extracting NL and PL Info

- **NL information**
  - Names of functions and arguments
  - Function-level comments
- **PL information**
  - Occurrences of the to-be-typed code element
  - Types made available via imports

Extracting NL Information

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting NL Information

Identifiers associated with the to-be-typed program element

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting NL Information


```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Tokens around occurrences of the to-be-typed code element

Extracting PL Information

Tokens around occurrences of the to-be-typed code element

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Extracting PL Information

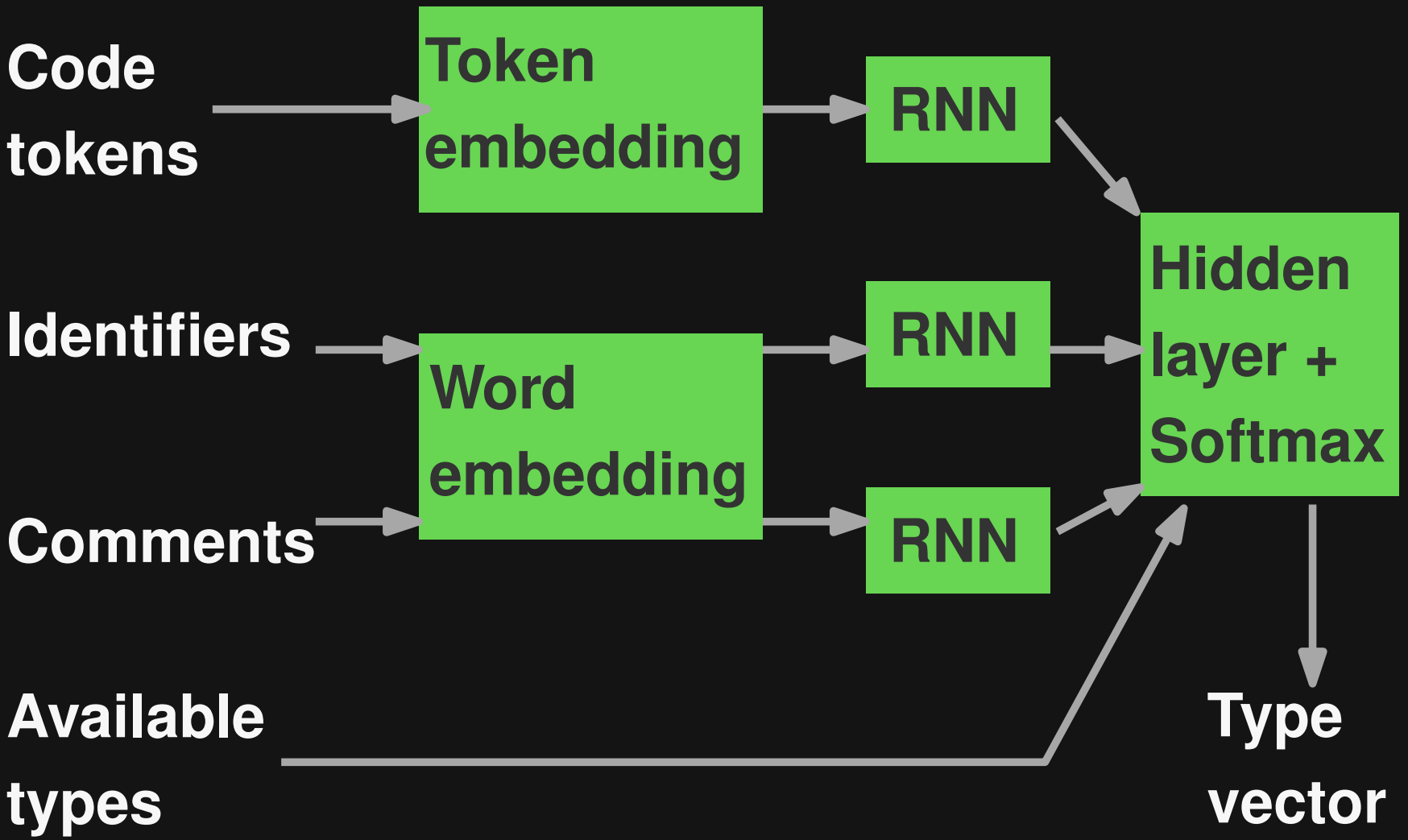
```
from ab import de
import x.y.z

def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Types made available via imports

Neural Type Prediction Model



Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript
- **Combinatorial search problem**
  - For type slots S and k predictions per slot: $(k+1)^{|S|}$ possible type assignments

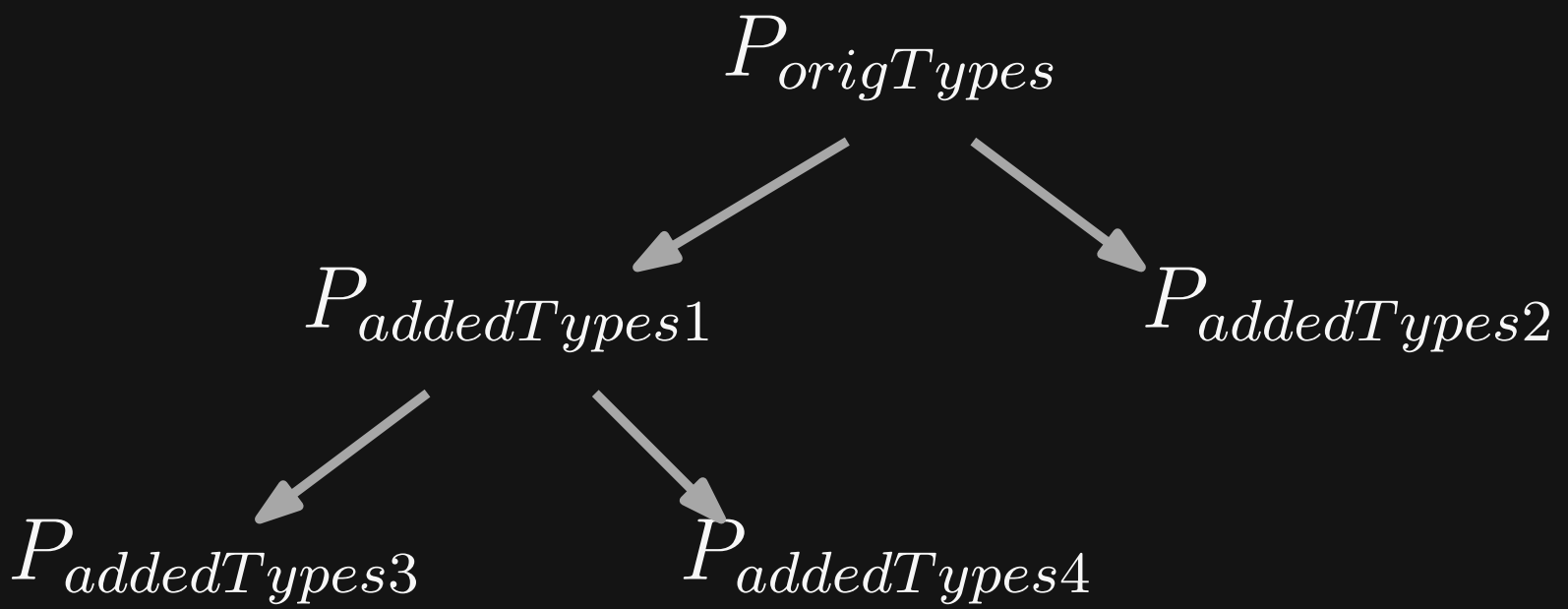
Searching for Consistent Types

- **Top-k predictions for each missing type**
  - Filter predictions using gradual type checker
  - E.g., pyre and mypy for Python, flow for JavaScript
- **Combinatorial search problem**
  - For type slots S and k predictions per slot: $(k+1)^{|S|}$ possible type assignments

Too large to explore exhaustively!

Exploring the Search Space

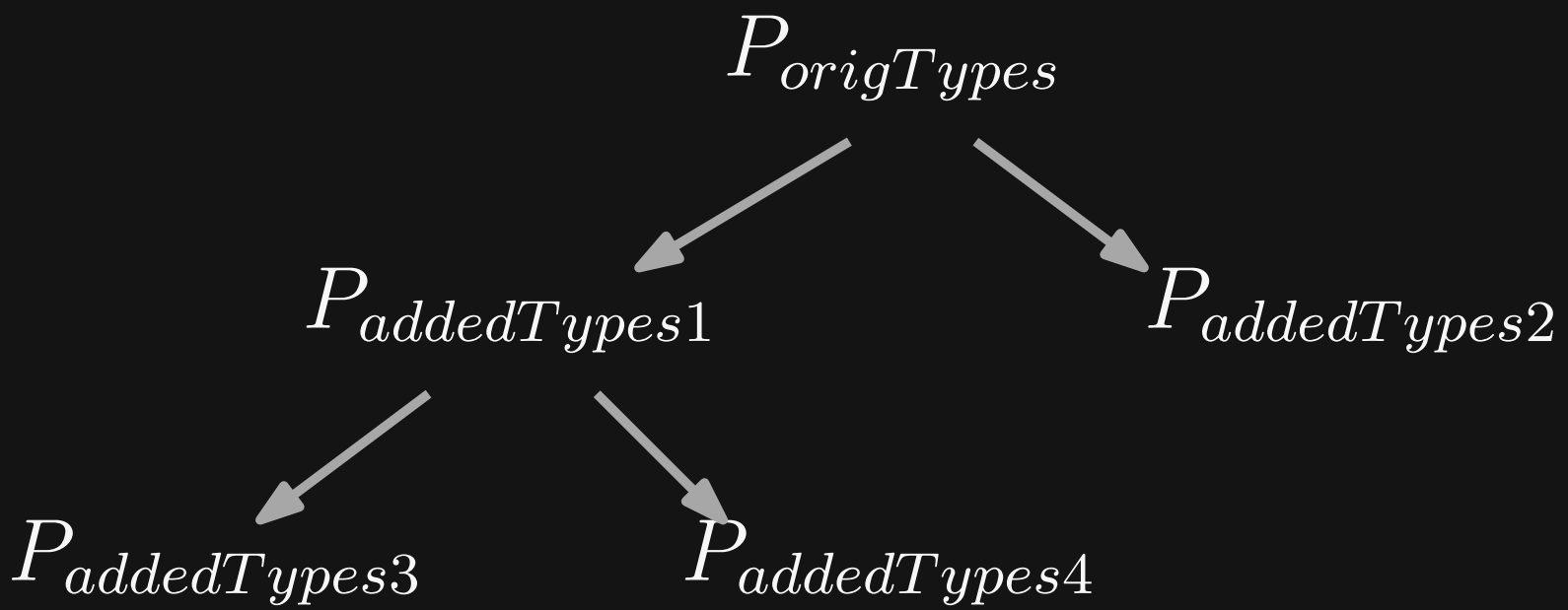
Tree of variants of program P



→ ... add, remove, or replace types

Exploring the Search Space

Tree of variants of program P



Which variants to explore first?

→ **... add, remove, or replace types**

Feedback Function

- Goal: **Minimize missing types** without introducing **type errors**
- **Feedback score** (lower is better):

$$v \cdot n_{missing} + w \cdot n_{errors}$$

Feedback Function

- Goal: Minimize missing types without introducing type errors

- Feedback score (lower is better):

$$v \cdot n_{missing} + w \cdot n_{errors}$$

Default: $v = 1$, $w = 2$,
i.e., higher weight for errors

Exploring the Search Space

- **Optimistic**: Add top-most predicted type everywhere and then remove types

- **Greedy** or **non-greedy**

If score decreases, keep the type

Backtrack to avoid local minima

Example

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Example

```
def find_match(color):
    """
    Args:
      color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
    """
    Args:
      color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
    """
    Args:
      color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:

1) int
2) str
3) bool

Predictions:

1) str
2) Optional[str]
3) None

Predictions:

1) List[str]
2) List[Any]
3) str

Example

```
def find_match(color):
    """
    Args:
      color (str): color to match on and return
    """
    candidates = get_colors()
    for candidate in candidates:
      if color == candidate:
        return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Evaluation: Setup

- **Code corpora**
  - Facebook's Python code
  - 5.8 millions lines of open-source code
- **Types**
  - Millions of argument and return types
  - 6-12% already annotated
  - Trivial types (e.g., type of `self`) ignored

Effectiveness of Neural Model

Approach	Top-1		
	Prec	Rec	F1
TypeWriter	65%	59%	62%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Neural Model

Approach	Top-1			Top-3			Top-5		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
TypeWriter	65%	59%	62%	80%	71%	75%	85%	75%	80%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Neural Model

Approach	Top-1			Top-3			Top-5		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
TypeWriter	65%	59%	62%	80%	71%	75%	85%	75%	80%
NL2Type	59%	55%	57%	73%	67%	70%	79%	71%	75%
Frequencies	12%	20%	15%	19%	35%	25%	22%	39%	28%

Open-source corpus; Combined prediction (arg. and return types)

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1		
	3		
	5		
Non-greedy search	1		
	3		
	5		

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1	215 (70%)	194 (63%)
	3	230 (75%)	196 (64%)
	5	231 (75%)	198 (65%)
Non-greedy search	1	216 (71%)	195 (64%)
	3	178 (58%)	148 (48%)
	5	164 (54%)	141 (46%)

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

Effectiveness of Search

Strategy	Top-k	Annotations	
		Type-correct	Ground truth match
Greedy search	1	215 (70%)	194 (63%)
	3	230 (75%)	196 (64%)
	5	231 (75%)	198 (65%)
Non-greedy search	1	216 (71%)	195 (64%)
	3	178 (58%)	148 (48%)
	5	164 (54%)	141 (46%)
Pyre Infer	—	106 (35%)	78 (25%)

Subsumes practically all types

Ground truth: 306 annotations in 47 fully annotated files

Exploring up to $7 \cdot |S|$ states

Limitations

- Type-correctness vs. soundness
- Limited type vocabulary
- Gradual type checking (and hence TypeWriter) is relatively slow

Conclusion

Neural type prediction with search-based validation

- Probabilistic type prediction based on NL and PL information
- Ensure type correctness of added types via gradual type checker
- TypeWriter tool in use at Facebook