Blame Assignment

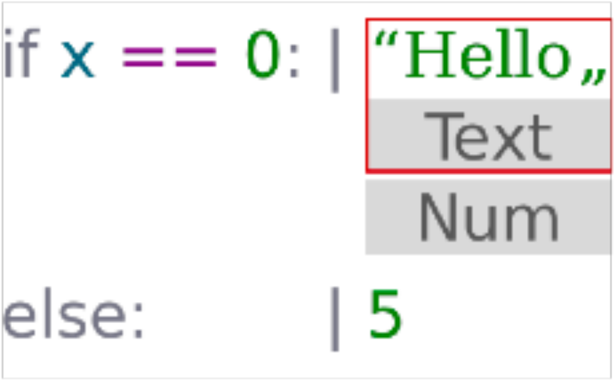
Traditionally compilers arbitrarily assign blame - often editing one piece of code results in an error somewhere seemingly unrelated.

“Hello” typed first

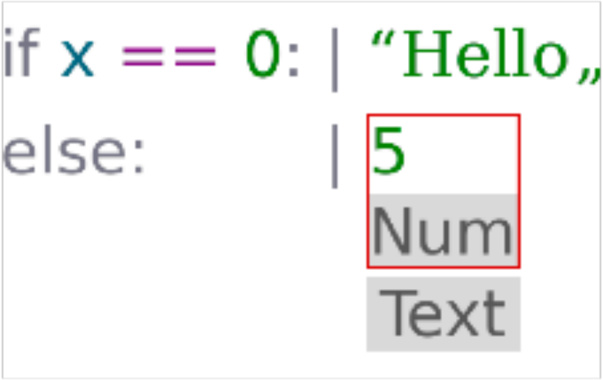
5

typed

first

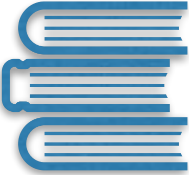


```
if x == 0: | “Hello„
           | Text
           | Num
else:      | 5
```



```
if x == 0: | “Hello„
else:      | 5
             Num
             Text
```

Lamdu uses the order in which code is written for
to assign type mismatches to newly written code.

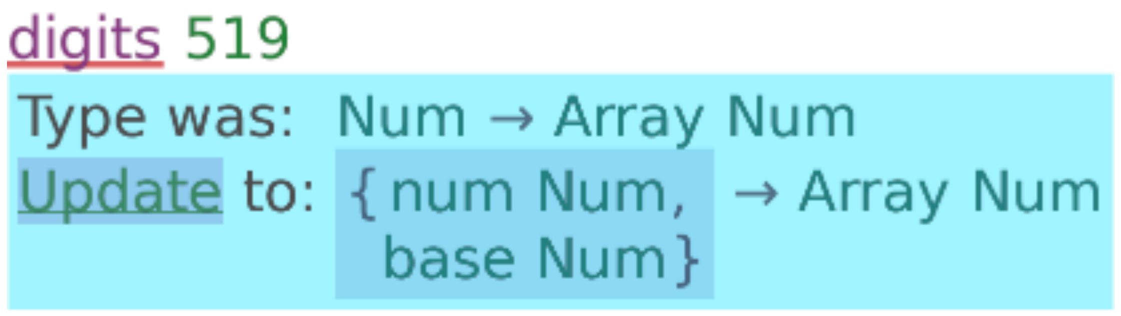


Blame Assignment Across Definitions

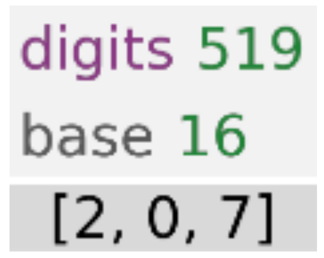
Within each definition, Lamdu stores the types of its dependencies. When they change, Lamdu tracks both the new type and the previously used type.

Until accepting the updated type, the old type is used for type inference, preserving coherency.

When updating, local type mismatches may be created.



```
digits 519
Type was:  Num → Array Num
Update to: {num Num,   → Array Num
            base Num}
```



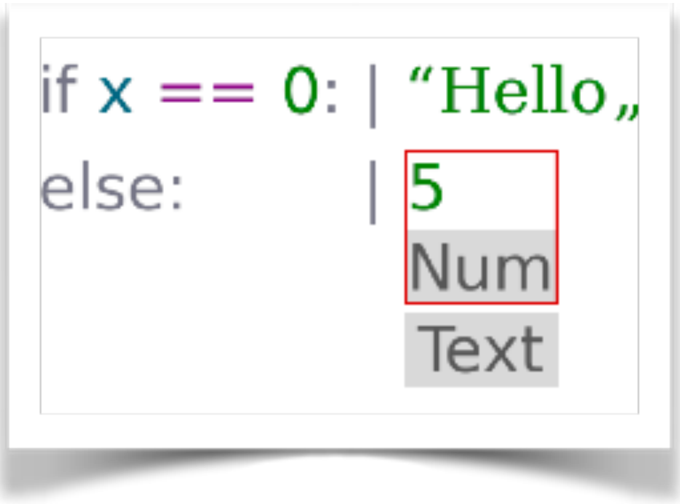
```
digits 519
base 16
[2, 0, 7]
```

Blame Assignment

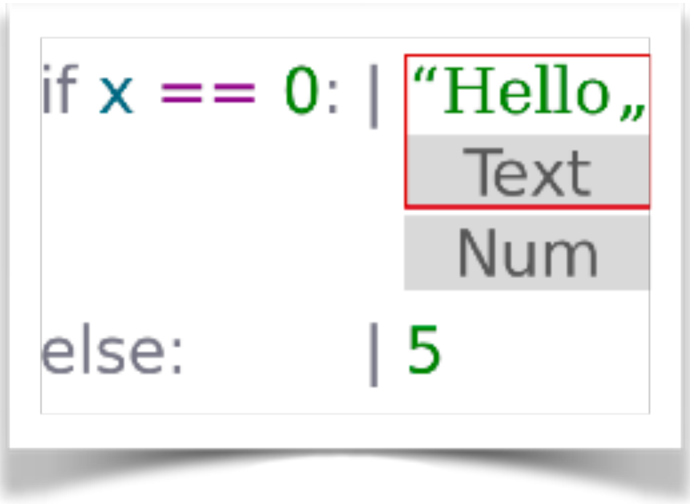
Traditionally compilers arbitrarily assign blame - often editing one piece of code results in an error somewhere seemingly unrelated.

Lamdu uses the order in which code is written for to assign type mismatches to newly written code.

"Hello" typed first



```
if x == 0: | "Hello„
else:      | 5
             Num
             Text
```



```
if x == 0: | "Hello„
             Text
             Num
else:      | 5
```

5 typed first