Python Biella Group

Serata 2 – Parser all'opera

Obiettivi

- Applicazioni del Language Engineering
- Abstract Syntax Tree: perché?
- Scrivere un interpreter

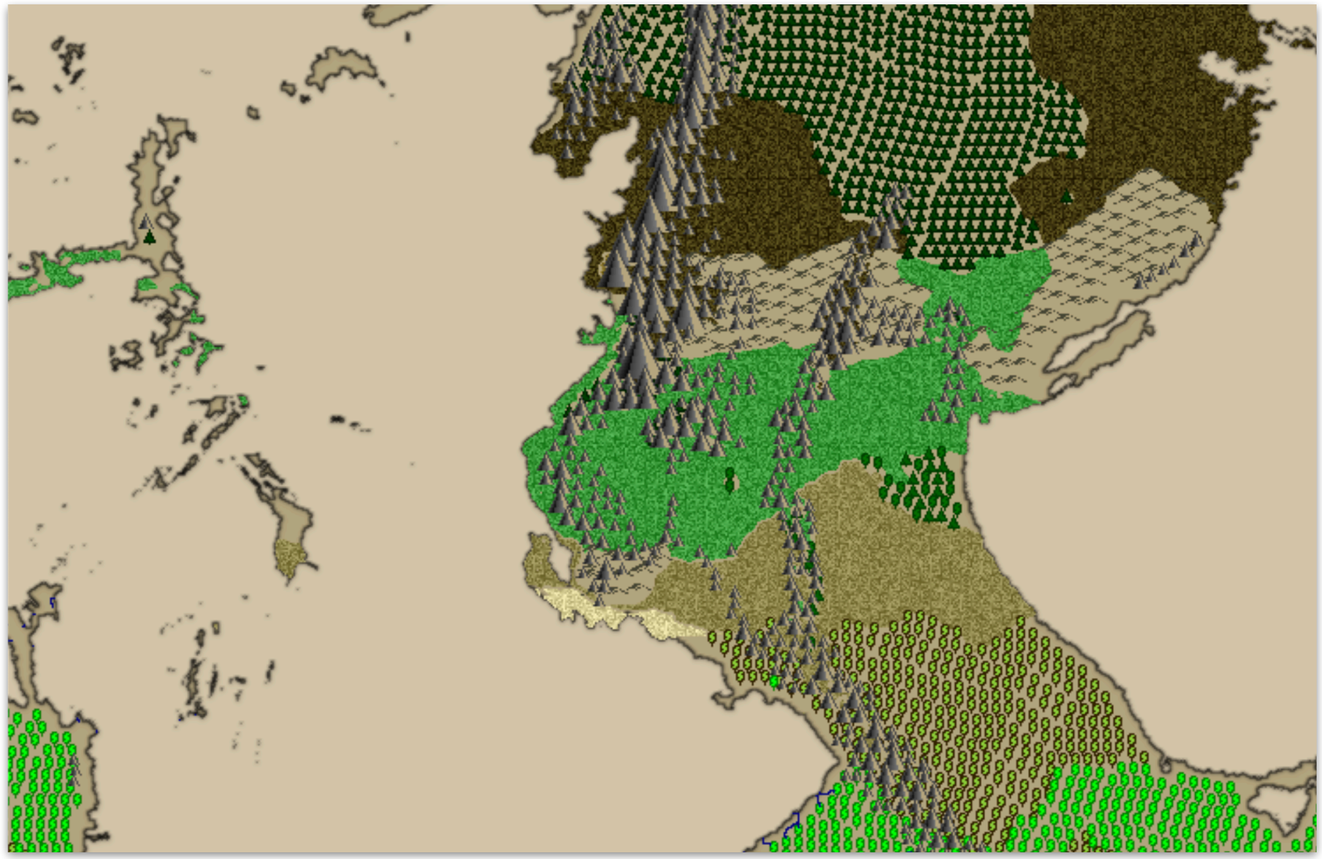
STRU
MEN
TA

Python Biella Group

Serata 2 – Parser all'opera

Lo speaker

- federico@strumenta.com
- Language Architect
- Non esperto di Python



https://github.com/Mindwerks/worldengine

STRU
MEN
TA

1. Applicazioni del Language Engineering

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  1. Se capisco il linguaggio posso “colorare” gli elementi del linguaggio
  2. Se capisco il linguaggio posso riportare gli errori e dare suggerimenti su come correggerli.
  3. Se capisco il linguaggio posso dare suggerimenti su che valori inserire quando viene lanciato l’autocompletamento
  4. Se capisco a quale funzione l’utente stia facendo riferimento posso mostrare la lista dei parametri attesi
  5. Se capisco il linguaggio posso collassare gli elementi

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  1. Se capisco il linguaggio posso “colorare” gli elementi del linguaggio

```
14
15  def branch_last_change(branch):
16      tz = pytz.timezone('Europe/Paris')
17      return datetime.fromtimestamp(branch.commit.authored_date, tz)
18
19
20  def branch_age(branch, now) -> int:
21      delta = now - branch_last_change(branch)
22      delta_days = delta.days
23      return delta_days
24
```

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  2. Se capisco il linguaggio posso riportare gli errori e dare suggerimenti su come correggerli

```
def branch_last_change(branch):
    tz = pytz.timezone('Europe/Paris')
    r                                  tamp(branch
```

Shadows name 'tz' from outer scope
Rename the element ⌥⇧⏎ More actions... ⌥⏎

```
tz: Any = pytz.timezone('Europe/Paris')
```

```
def branch_age(branch, now) -> int:
```

```
import pytz
from date                              imede
from data                              s
```

No module named 'pytz'
Install package pytz ⌥⇧⏎ More actions... ⌥⏎

No documentation found.

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  3. Se capisco il linguaggio posso dare suggerimenti su che valori inserire quando viene lanciato l'autocompletamento

```
7   tz = pytz.timezone('Europe/Paris')
8
9   def get_repo():
10      repo = Repo(Path(__file__).parent.parent.absolute())
11      assert not repo.bare
12
13      return
14      repo
15      if
16  de  True                    e(branch):...
19      Repo
20      and
21  de  assert                  , now) -> int:
22      class                   nch_last_change(branch)
23      def                     a.days
24      del
25      exec
        for
        Press ⏎ to insert, ⇥ to replace
```

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  4. Se capisco a quale funzione l'utente stia facendo riferimento posso mostrare la lista dei parametri attesi

```
26
27  def                                                    -> bool:
28      branch_last_change()
```

def branch_last_change(branch: {commit}) -> datetime

Applicazione 1 - Editor intelligenti

- Per dare supporto intelligente bisogna capire il linguaggio
  5. Se capisco il linguaggio posso collassare gli elementi

```
26
27  def is_special_branch(branch) -> bool:
28      names = ["dev", "master", "stable"]
29      prefixed_names = [f"origin/{n}" for n in names]
30      return branch.name in names or branch.name in prefixed_names
31
32
```

Applicazione 2 - DSL

Manifesto

Subject matter experts own the expertise at the core of software. But too often this rich expertise is not captured in a structured way and gets lost when translating it for software developers. With the rate of change increasing and product variability blooming, putting knowledge into software in this way is increasingly untenable. It causes delays, quality problems and frustration for everybody involved.

We advocate for putting SMEs directly in control of “their” part of the software and letting developers focus on software engineering.

https://subjectmatterfirst.org/

Applicazione 2 - DSL

```
decision tree DiarrheaStoolsDecisionTree(score: DiarrheaStoolsOverBaseline,
                                         patientHasAnySymptom: boolean, goToStartBrat: boolean)
patientHasAnySymptom
  ├─ DiarrheaReco1
  └─ score >= 7
     ├─ DiarrheaReco3
     └─ score in [4..6]
        ├─ DiarrheaReco2
        └─ goToStartBrat
           ├─ DiarrheaRecoSBrat
           └─ DiarrheaRecoCBrat
```

decision table BpScoreDecisionTable(sys: bpRange, dia: bpRange) =

		dia					
		<= 50	[51..90]	[91..95]	[96..100]	[101..109]	>= 110
sys	<= 90	1	1	3	4	5	6
	[91..140]	2	2	3	4	5	6
	[141..150]	3	3	3	4	5	6
	[151..160]	4	4	4	4	5	6
	[161..179]	5	5	5	5	5	6
	>= 180	6	6	6	6	6	6

https://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

Applicazione 2 - DSL

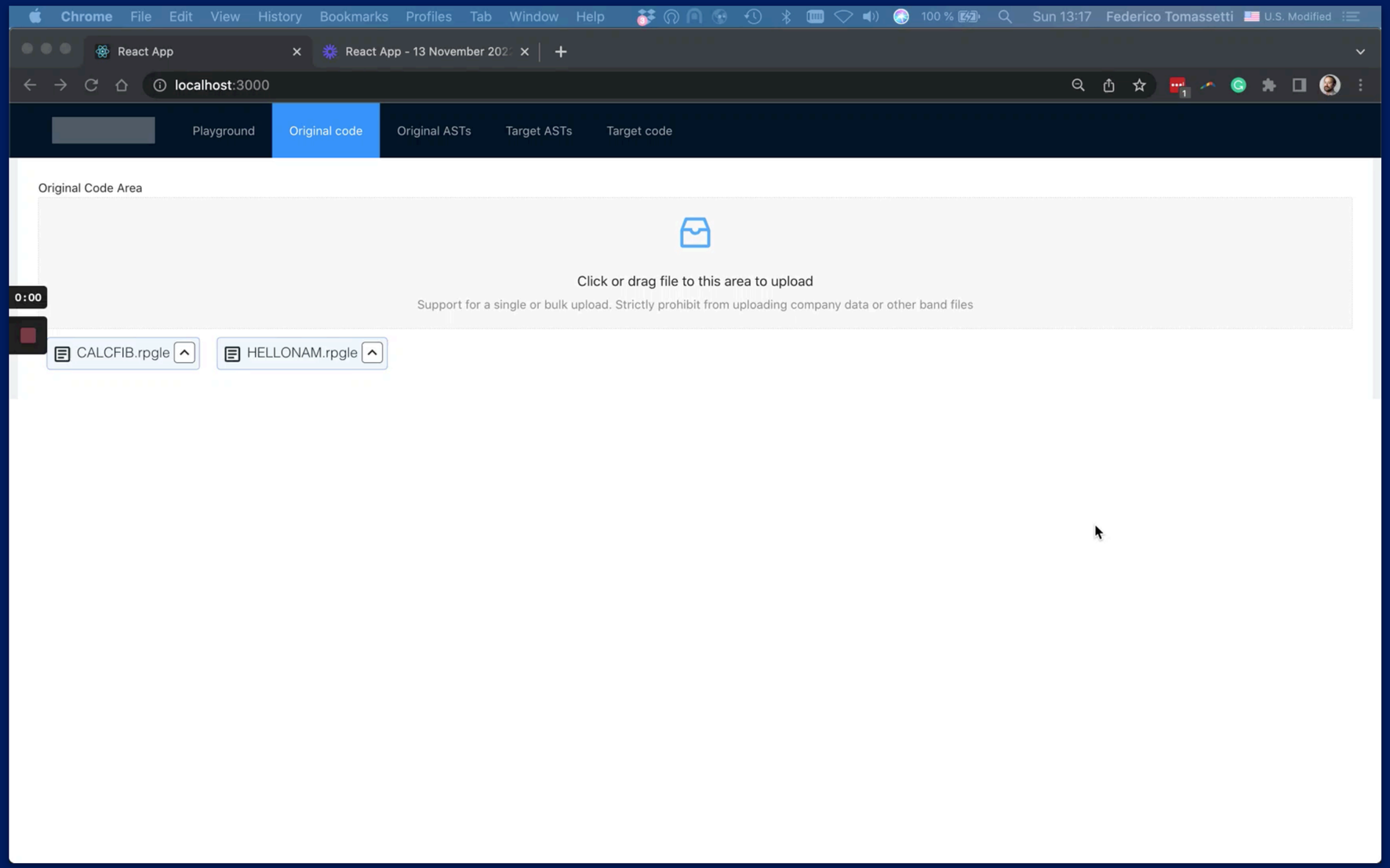
```
type temperature: number[36|42]{1}
type measuredTemp: number[35|43]{2}

Error: type number[32.55|39.99]{4} is not a subtype of number[36|42]{1}
val T_measured: measuredTemp = 42.22
val T_calibrated: temperature = T_measured * 0.93
```

```
scenario scenario_8
  global timeout:    1 hours
  time granularity:  60 seconds
given
  inputPainBaseline          = 1
  inputPainMedicineDuration = Six
when
  at 0 min: EventInPainMeasure answers to PainMeasureQuestionnaire {
                                  measure: 4
                                }
            EventInPainSymptoms1 answers to PainSymptoms1Questionnaire {
                                    interferingDailyActivities: false
                                    newSite                    : false
                                    interferingAbilityToWalk   : false
                                  }
then
  at 0 hours: assert parent sent message Recommendation(PainRecoSymptom1, Six)
  at 29 min:  assert parent in state     PainMeasure.Ask
  at 30 min:  assert parent in state     PainInitial.Ask
```

https://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

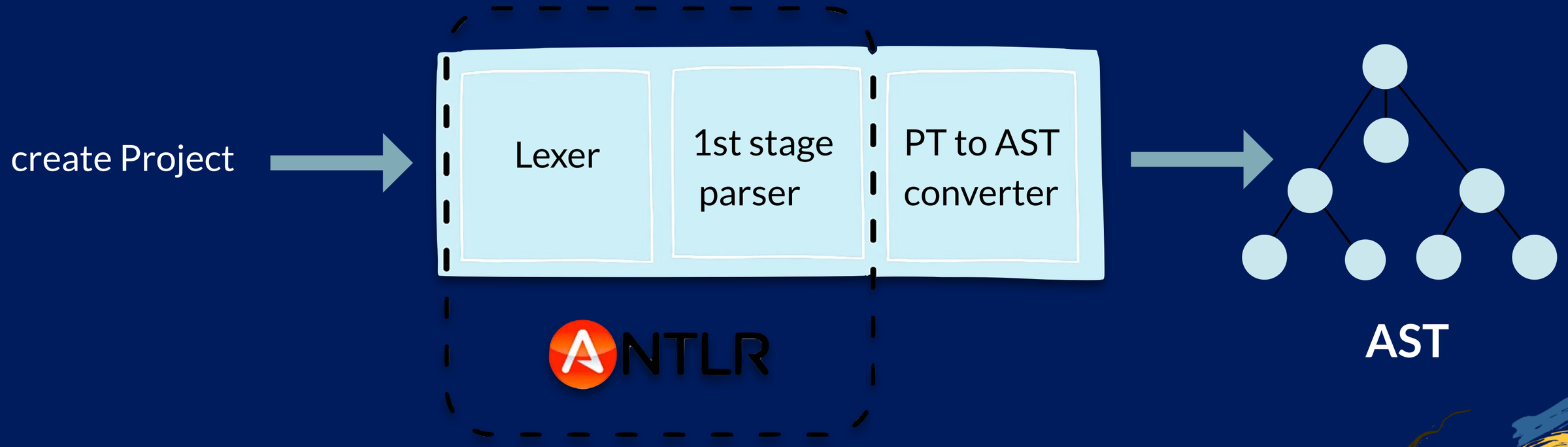
Applicazione 3 - Transpiler



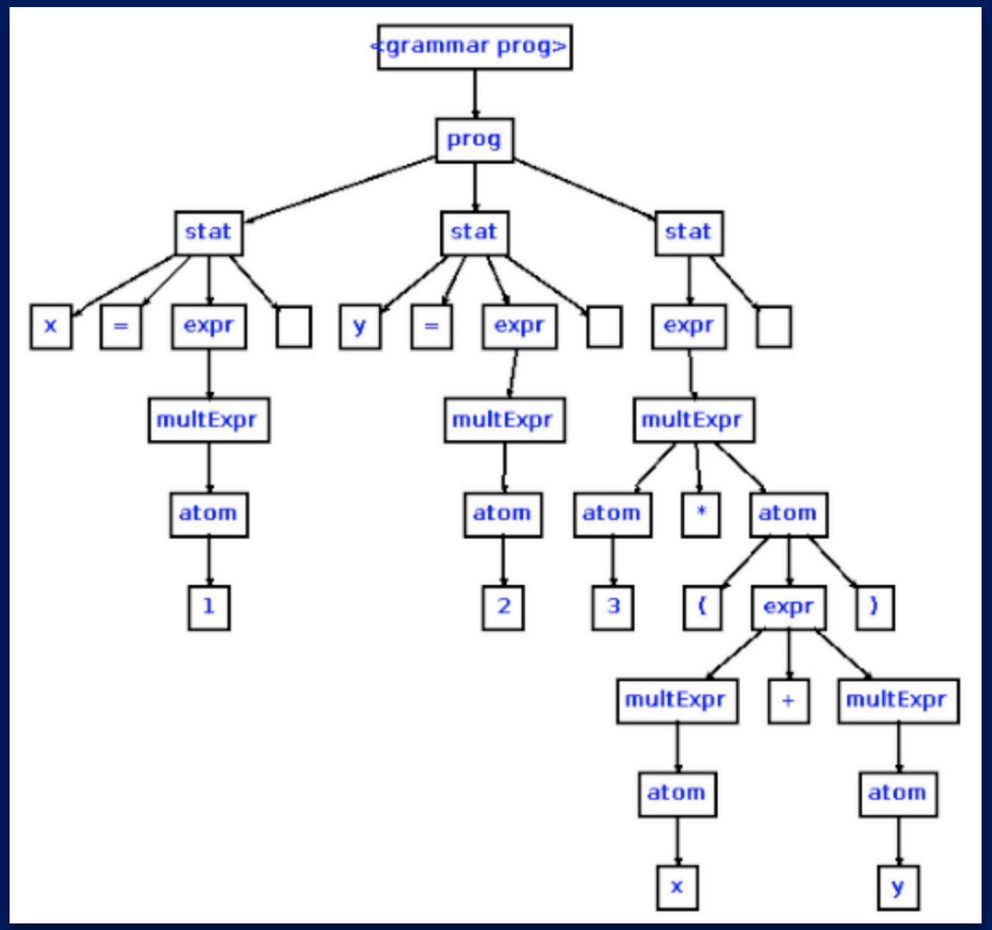
Demo:
Cosa vogliamo costruire

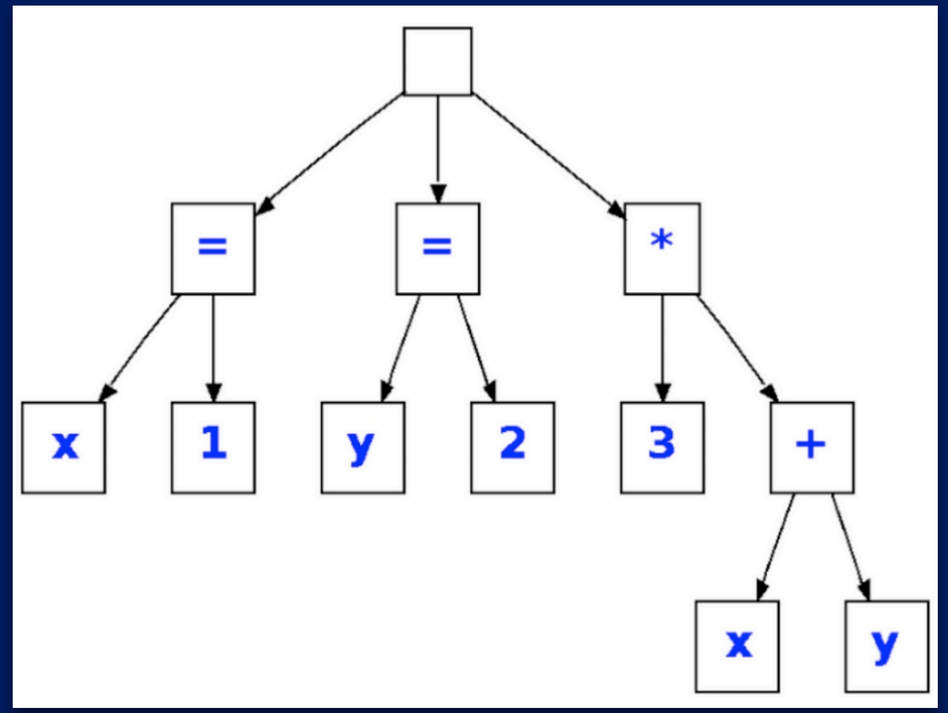
2. Abstract Syntax Tree: perché?

AST: Architettura



Perché l'AST? Semplificazione





Perché l'AST? Semplificazione

Parse Tree

```
R
  Statement
    Dspec
      T:DS_FIXED[D]
      Ds_name
        T:NAME[Msg§]
      T:EXTERNAL_DESCRIPTION[ ]
      T:DATA_STRUCTURE_TYPE[ ]
      T:DEF_TYPE_S[S ]
      T:FROM_POSITION[       ]
      T:TO_POSITION[     12]
      T:DATA_TYPE[ ]
      T:DECIMAL_POSITIONS[  ]
      T:RESERVED[ ]
      T:EOL[
]
```

Abstract Syntax Tree

```
DataDefinition {
    fields = [ ]
    initializationValue = null
    name = Msg§
    type = StringType(length=12)
 } // DataDefinition
```

```
D Msg§                    S                    12
```

Perché l'AST? API

StarLasu: Overview

StarLasu is a methodology to design and implement Language Engineering applications. The methodology is supported by a set of libraries that permit its application on different platforms.

The libraries are:

- Kolasu, for the implementation on the JVM (and in particular with Kotlin and Java)
- Tylasu, for the implementation on Node.JS and on the browser, using Typescript or Javascript
- Pylasu, for the implementation with Python

Examples of applications that could use StarLasu:

- Transpilers
- Parsers
- Code editors
- Code analyzers
- Linters
- Refactoring tools

https://starlasu.strumenta.com/

Demo:
L’AST nel nostro caso

Come l'AST? Definizione classi

```python
@dataclass
class Feature(Node):
    name: str = field(default=None)
    type: Type = field(default=None)
    many: bool = field(default=False)

    def __hash__(self) -> int:
        return self.name.__hash__()
```

Come l’AST? Mappatura parse-tree -> AST

```python
@extension_method(AntlrEntityParser.ModuleContext)
def to_ast(self: AntlrEntityParser.ModuleContext, issues: list[Issue]) -> Module:
    return Module(name=self.name.text, entities=[c.to_ast(issues) for c in self.entities])
```

Come l'AST? Test con l'AST

```python
    def test_pylasu_parse_module(self):
        code = '''module Insurance {
    entity Vehicle {
        licensePlate: string;
        year: integer;
        owner: Person;
    }

    entity Person {
        name: string;
        address: string;
    }
}'''
        result = EntitiesPylasuParser().parse(code)
        self.assertEqual([], result.issues)
        person = Entity(name='Person',
                        features=[
                            Feature(name='name', type=StringType()),
                            Feature(name='address', type=StringType()),
                        ])
        vehicle = Entity(name='Vehicle',
                         features=[
                             Feature(name='licensePlate', type=StringType()),
                             Feature(name='year', type=IntegerType()),
                             Feature(name='owner', type=EntityRefType(entity=ReferenceByName(name='Person'))),
                         ])
        self.assertEqual(Module(
            name='Insurance',
            entities=[
                vehicle,
                person
            ]
        ), result.root)
```

3. Scrivere un interprete

Come eseguire il codice?

- Interprete
- Compilatore
- Transpiler

Esempio

```
create Client as c
set name of c to 'ACME Inc.'
create Product as p1
set value of p1 to (1500 + 200) / 2
print concat 'Value of product #1 is: ' and value of p1
```

Demo:
L’interprete nel nostro caso

Interprete - Risoluzione dei simboli

```
def __resolve__(self, script: Script):
    for s in script.walk_descendants(restrict_to=GetInstanceExpression):
        s.entity.try_to_resolve(self.module.entities)
    for s in script.walk_descendants(restrict_to=CreateStatement):
        s.entity.try_to_resolve(self.module.entities)
    for e in script.walk_descendants(restrict_to=ReferenceExpression):
        e.what.try_to_resolve(script.walk_descendants(restrict_to=CreateStatement))
```

Interprete - Calcolo dei tipi

```
def __calc_type__(self, script: Script, node: Node) -> RType:
    if isinstance(node, ReferenceExpression):
        return REntityRefType(node.what.referred.entity.referred)
    elif isinstance(node, GetInstanceExpression):
        return REntityRefType(node.entity.referred)
    else:
        raise Exception("Unable to calculate type for %s" % (str(node)))
```

Interprete - Esecuzione

```
def evaluate_expression(self, expression, symbol_table) -> Any:...

def execute_statement(self, statement, symbol_table):...
```

Concludiamo

Il codice mostrato è su GitHub: https://github.com/PythonBiellaGroup/ANTLR
branch “serata2”

Il branch main conterrà il codice completo alla fine delle 3 serate

E se volessi imparare di più?

- Strumenta Community: https://strumenta.community/
- ANTLR Mega Tutorial: https://tomassetti.me/antlr-mega-tutorial/
- Video corso ANTLR: https://tomassetti.me/antlr-course-2-edition
- Pylasu: https://github.com/strumenta/pylasu
- Documentazione StarLasu: https://starlasu.strumenta.com/
- Webinar su parser e transpiler

Tutto il materiale è in inglese

Domande e risposte

Piazza Duomo, Biella