Graph- and behaviour-based machine learning models to understand program semantics

Seminar Current Topics in Compiler Construction (Hauptseminar)

Benno Fünfstück

July 13, 2020

Why ML for programs?

```
function fetchData(retries) {
  for (var i = 0; i < retries; i += 1) {
    print("attempt", i);

    // ...
  }
}
```

what is the type of the parameter `retries`?

Why ML for programs?

```
function fetchData(retries) {
  for (var i = 0; i < retries; i += 1) {
    print("attempt", i);

    // ...
  }
}
```

what is the type of the parameter `retries`?

The naturalness hypothesis

"Software is a form of human communication; software corpora have similar statistical properties to natural language corpora; and these properties can be exploited to build better software engineering tools." [Allamanis et al., 2018a]

Representing programs

Characters / Tokens

Abstract Syntax Tree

Program Graphs (data flow, control flow)

Program Behaviour (execution traces)

Representing programs

source code

Characters / Tokens

Abstract Syntax Tree

Program Graphs (data flow, control flow)

semantics

Program Behaviour (execution traces)

Representing programs

source code

semantics

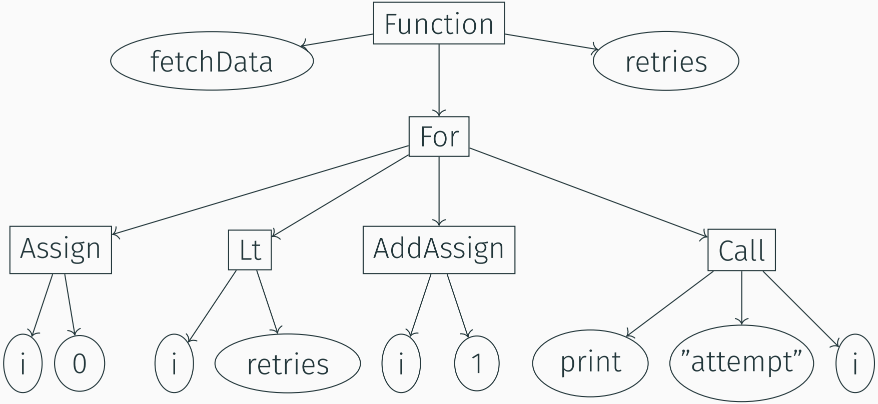
Characters / Tokens

Abstract Syntax Tree

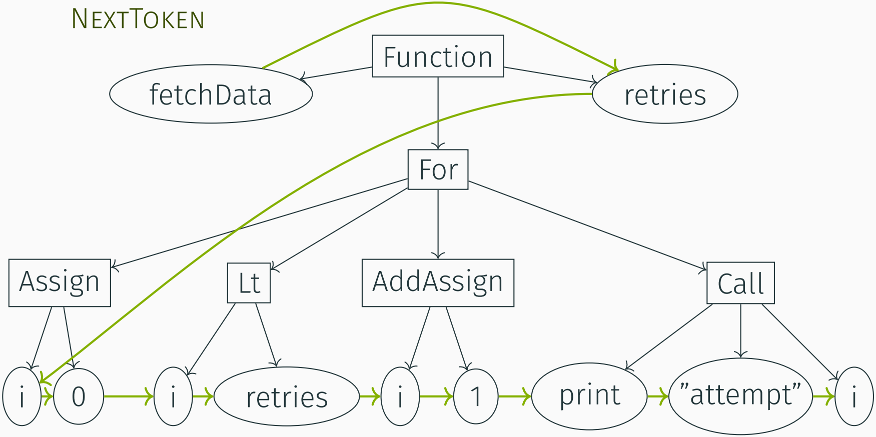
Program Graphs (data flow, control flow)

Program Behaviour (execution traces)

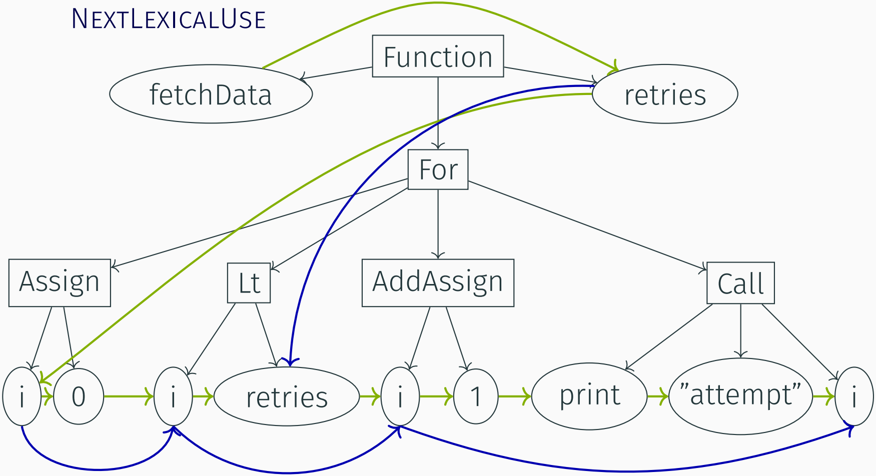
Programs as graphs [Allamanis et al., 2018b]



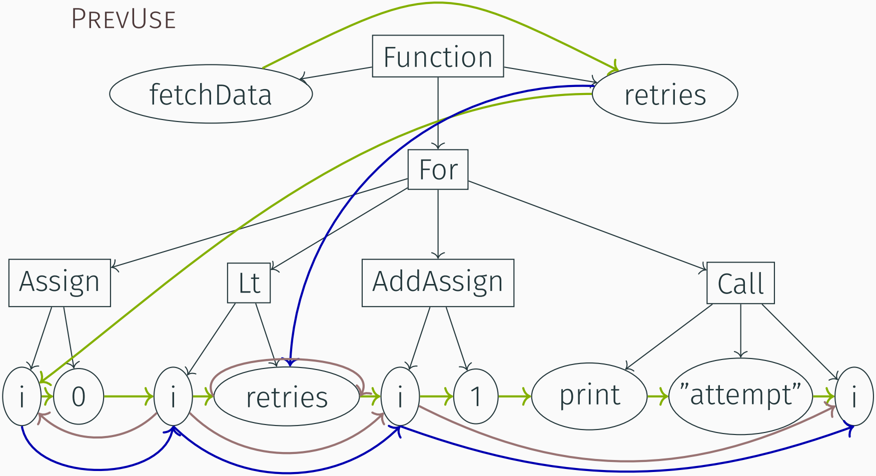
Programs as graphs [Allamanis et al., 2018b]



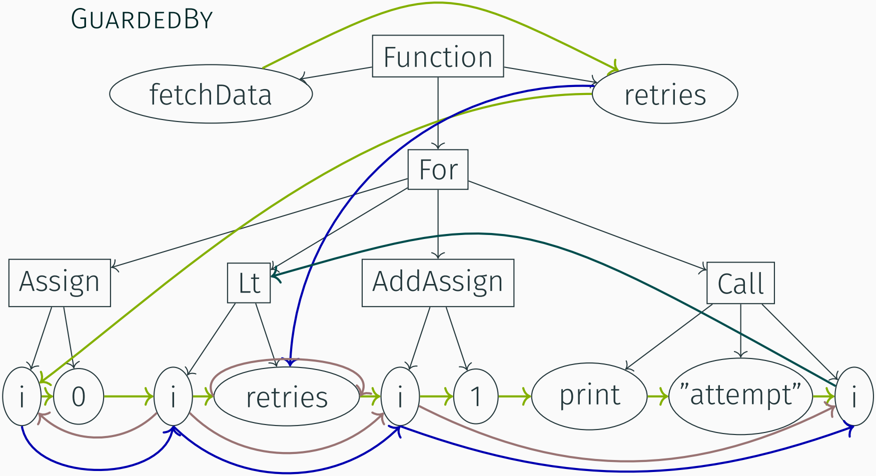
Programs as graphs [Allamanis et al., 2018b]



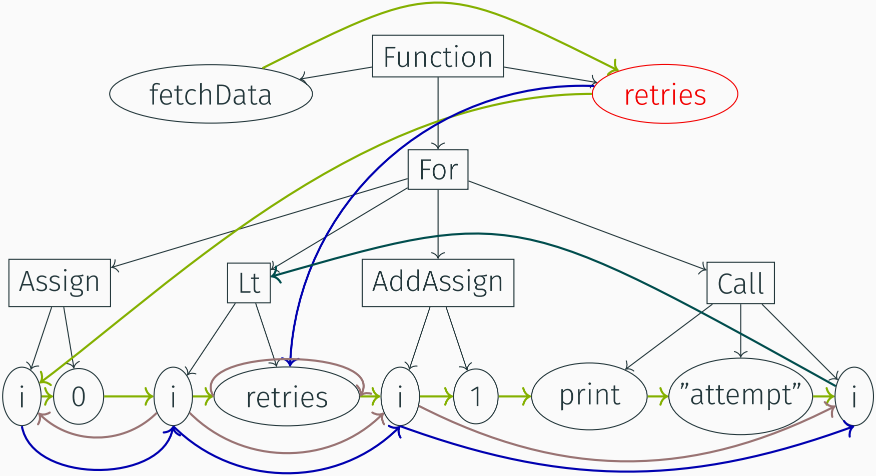
Programs as graphs [Allamanis et al., 2018b]



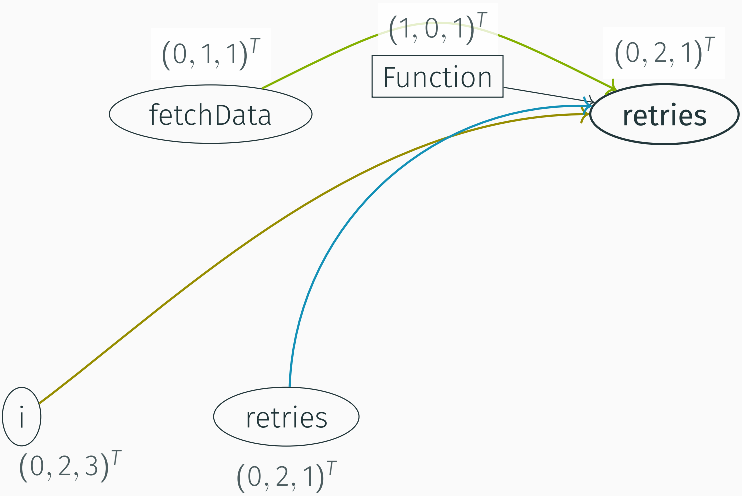
Programs as graphs [Allamanis et al., 2018b]

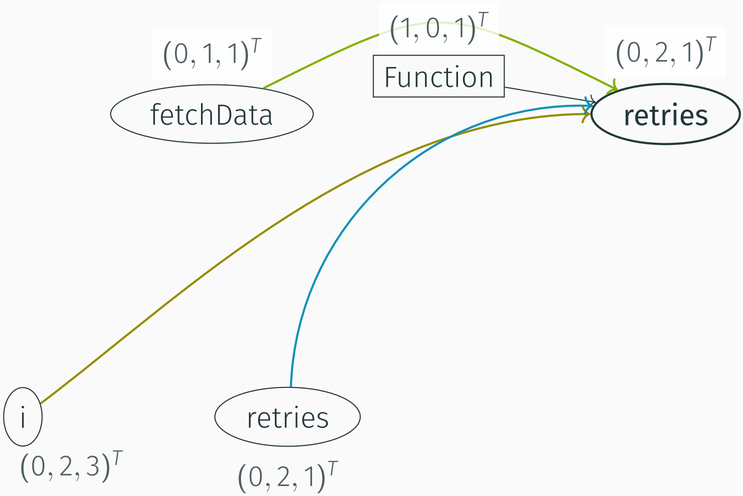


Programs as graphs [Allamanis et al., 2018b]



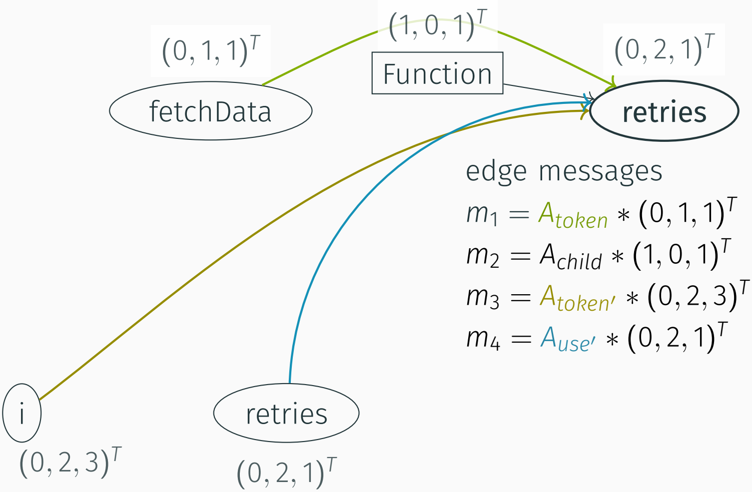
Message Passing Neural Network





$(0, 1, 1)^T$
$(1, 0, 1)^T$
$(0, 2, 1)^T$
Function
fetchData
retries
i
retries
$(0, 2, 3)^T$
$(0, 2, 1)^T$

Message Passing Neural Network



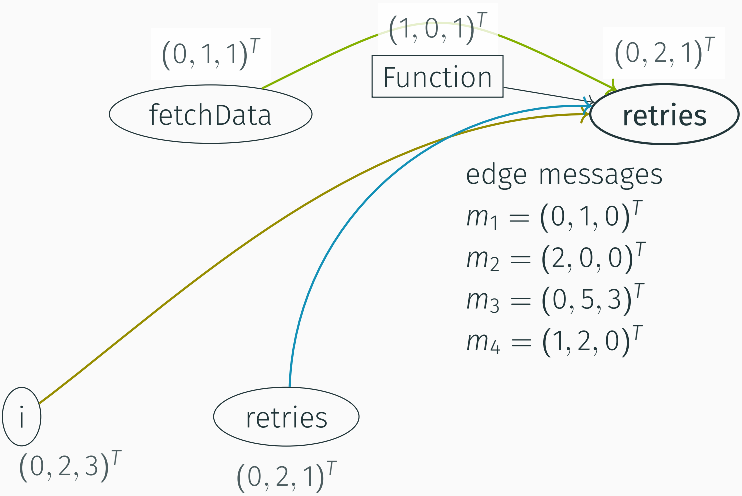
edge messages

$m_1 = A_{token} * (0,1,1)^T$

$m_2 = A_{child} * (1,0,1)^T$

$m_3 = A_{token'} * (0,2,3)^T$

$m_4 = A_{use'} * (0,2,1)^T$



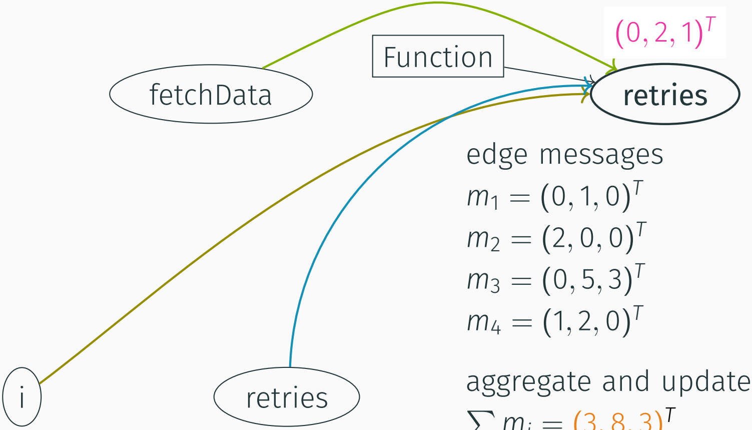
edge messages

$m_1 = (0, 1, 0)^T$

$m_2 = (2, 0, 0)^T$

$m_3 = (0, 5, 3)^T$

$m_4 = (1, 2, 0)^T$



edge messages

$m_1 = (0, 1, 0)^T$

$m_2 = (2, 0, 0)^T$

$m_3 = (0, 5, 3)^T$

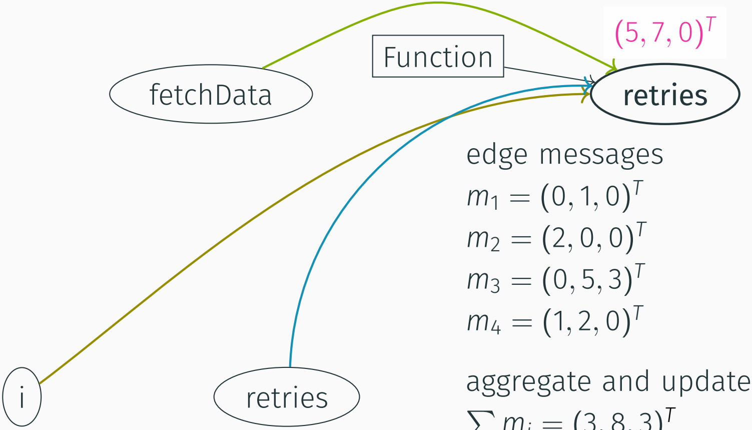
$m_4 = (1, 2, 0)^T$

aggregate and update

$\sum m_i = (3, 8, 3)^T$

$h_{t+1} = GRU((0, 2, 1)^T, (3, 8, 3)^T)$

Message Passing Neural Network



edge messages

$m_1 = (0,1,0)^T$

$m_2 = (2,0,0)^T$

$m_3 = (0,5,3)^T$

$m_4 = (1,2,0)^T$

aggregate and update

$\sum m_i = (3,8,3)^T$

$h_{t+1} = GRU((0,2,1)^T, (3,8,3)^T)$

Representing programs

source code

Characters / Tokens

Abstract Syntax Tree

Program Graphs (data flow, control flow)

semantics

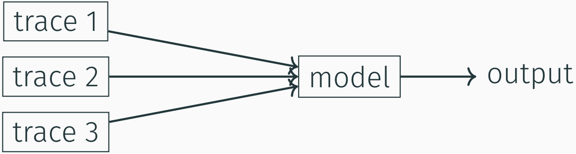
Program Behaviour (execution traces)

Programs as traces

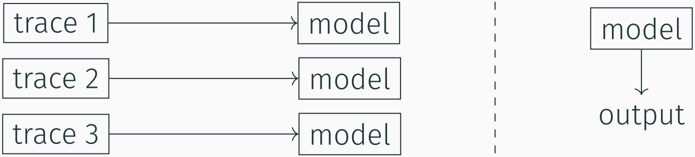
trace 1	trace 2	trace 3
…	…	…
retries=3 i=0	retries=5 i=0	retries=1 i=0
retries=3 i=1	retries=5 i=1	retries=1 i=1
retries=3 i=2	retries=5 i=2	…
retries=3 i=3	retries=5 i=3	
…	retries=5 i=4	
	retries=5 i=5	
	…	

Learning from traces

variant 1: learn program embedding



variant 2: learn model to “emulate” program (synthesis)



Evaluation

Evaluation: MethodNaming

```
boolean f(Set<String> set, String value) {
    for (String entry : set) {
        if (entry.equalsIgnoreCase(value)) {
            return true;
        }
    }
    return false;
}
```

prediction: contains ignore case

Evaluation: MethodNaming

java dataset (java-*)

code2seq [Alon et al., 2019] → SNS [Fernandes et al., 2019]

code2seq [Alon et al., 2019] → DyPro [Wang, 2019]

code2seq [Alon et al., 2019] → LiGer [Wang and Su, 2020]

DyPro [Wang, 2019] → LiGer [Wang and Su, 2020]

POJ-104 dataset

NCC [Ben-Nun et al., 2018] ⟶ ProGraML [Cummins et al., 2020]

Evaluation: Invariant inference

```
x = 1; y = 0;
while (y < 100000) {
  x = x + y;
  y = y + 1;
}
assert(x >= y);
```

Example invariant: $x > y$

Evaluation: Invariant inference

GGNN [Hellendoorn et al., 2019]
gated graph neural network for checking dynamically generated invariants

DyPro [Wang, 2019]
checking loop invariants with behaviour-based model

code2inv [Si et al., 2018]
synthesis using graph-neural network representation as memory

GCLN [Yao et al., 2020]
synthesis by fitting gated continuous logic networks to trace data

Conclusion and future research directions

- Graph-based models improve on simpler models
- Behaviour-based models better at semantics (as expected)
- But require execution traces

Research directions

datasets: standarized, to evaluate different models
architectures: combine static/dynamic, performance, pretrain
analysis: adversarial examples, different languages

References i

Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. (2018a).
A Survey of Machine Learning for Big Code and Naturalness.
arXiv:1709.06182 [cs].
arXiv: 1709.06182.

Allamanis, M., Brockschmidt, M., and Khademi, M. (2018b).
Learning to Represent Programs with Graphs.
ICLR.

Alon, U., Brody, S., Levy, O., and Yahav, E. (2019).
code2seq: Generating Sequences from Structured Representations of Code.
arXiv:1808.01400 [cs, stat].
arXiv: 1808.01400.

Ben-Nun, T., Jakobovits, A. S., and Hoefler, T. (2018).
Neural Code Comprehension: A Learnable Representation of Code Semantics.
In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 3585–3597. Curran Associates, Inc.

Cummins, C., Fisches, Z. V., Ben-Nun, T., Hoefler, T., and Leather, H. (2020).
ProGraML: Graph-based Deep Learning for Program Optimization and Analysis.
ArXiv.

Fernandes, P., Allamanis, M., and Brockschmidt, M. (2019).
Structured Neural Summarization.
arXiv:1811.01824 [cs, stat].
arXiv: 1811.01824.

Hellendoorn, V. J., Devanbu, P. T., Polozov, O., and Marron, M. (2019).
Are My Invariants Valid? A Learning Approach.
ArXiv.

Si, X., Dai, H., Raghothaman, M., Naik, M., and Song, L. (2018).
Learning Loop Invariants for Program Verification.
In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 7751–7762. Curran Associates, Inc.

References v

- Wang, K. (2019).
  Learning Scalable and Precise Representation of Program Semantics.
  ArXiv.
- Wang, K. and Su, Z. (2020).
  Blended, precise semantic program embeddings.
  In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 121–134, London, UK. Association for Computing Machinery.

Yao, J., Ryan, G., Wong, J., Jana, S., and Gu, R. (2020).
Learning Nonlinear Loop Invariants With Gated Continuous Logic Networks.
CoRR.
_eprint: 2003.07959v3.