Type-guided synthesis for dynamic languages

THOMAS LOGAN

ACM Reference Format:
Thomas Logan. 2022. Type-guided synthesis for dynamic languages. 1, 1 (October 2022), 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Dynamic programming languages can make writing programs quick and easy because they don't require specifying static bounds on behavior. Two of the most popular programming languages today, Javascript and Python, are dynamic programming languages. Javascript is the language of the web, while Python the most popular choice for data science and machine learning projects.

Although dynamic languages already offer ease and efficiency for writing programs, it may be possible to supplement this facility with autocompletion, or synthesis of programs from surrounding context. This article presents a theoretical system that synthesizes terms from context in a dynamic language. Synthesis of of programs for a dynamic language introduces a fundamental tension. While dynamic language programs benefit from a lack of static bounds, program synthesis must be a terminating procedure driven by static bounds representing the goals of synthesis.

Types have become the lingua franca of formal specification of static bounds. Others have shown how various forms of specification, including examples, abstract values, pure propositions, and modal propositions, can be encoded as types. Types have been used successfully for verifying programs, guiding program synthesis in ML-family languages, and guiding humans in dependently-typed interactive theorem proving.

TODO: what kind of types can tightly bound runtime behavior

Using types, it may be possible to synthesize programs for dynamic languages. To maintain the spirit of dynamic languages, type annotations must remain optional. If type annotations are provided, they can be propagated and decomposed, as is the case in theorem proving systems, local type checking, and synthesis of ML-family programs.

TODO: type propagation: when do we push down expected types and decompose

However, if type annotations are not available, then types must be inferred from context. Taking advantage of type inference techniques from ML-family languages is a good starting point, but it's not sufficient. ML-family type inference is sound, because an ML term is intrinsically associated with a static bound in accordance with ML's datatype mechanism.

TODO: type inference: when do we compose actual types and pop up

Terms in dynamic languages are not intrinsically limited to such static bounds, as that would be akin to specifying static bounds for all terms, which is antithetical to dynamic languages. Type inference for dynamic languages cannot be sound without violating the assumed definition of dynamic languages. Thus, it is necessary to devise a method of type

Author's address: Thomas Logan.

inference that is unsound, yet still provides sufficient information to guide efficient program synthesis. Additionally, despite being unsound, it should be able to prune/reject a significant portion of bad programs. It may reject good programs, but only if it can infer static bounds.

TODO: type widening: infer types from arguments

TODO: type narrowing: infer types from parameter types

2 OVERVIEW

3 TECHNICAL DETAILS

4 EVALUATION

5 RELATED WORK