CS 343S Project: Language Design and Implementation

1 Deliverables and Due Dates

Below are short descriptions and deadlines for each component of the term project. The following sections provide more details on the expected contents of the deliverables.

Component	**Deadline**
Brainstorming	Tuesday, April 16th, midnight
Proposal	Tuesday, April 30th, midnight
Demo	In-class May 20th and 22nd
Presentation	In-class June 3rd and 5th
Final Implementation	Thursday, June 6th, 2024, midnight

Table 1: Project deadlines

2 Summary

The deliverables are designed to guide your project direction and allow you to receive feedback at multiple points in time. That being said, you are not restricted to prior deliverables (i.e. you can write a proposal for an idea that you did not submit for brainstorming, and you can change the language sketch from your proposal while implementing the language). **There are no teams for final projects.** You are expected to implement your own project, and will receive feedback both from the staff and from a small group of your peers. Lecture periods during weeks 7-9 will be "Studio" time for you to work on your projects and ask for feedback outside of the concrete deliverables. You are welcome to implement your project in *any language you prefer*, you are not required to use the Python tools we will teach in the first few weeks of class.

3 Brainstorming

Your submission should include 3-5 short problem descriptions that you believe might be eloquently solved by a domain-specific language. Each problem description do not need to be longer than a paragraph (though you are welcome to write more), and should describe a **problem domain**, as well as at least one concrete example of a problem in that domain. We provide two sample brainstorms below:

3.1 Brainstorming Example 0

Fixed-point arithmetic [1] is commonly used in high-performance digital signal processing codes. Programmers using fixed-point arithmetic must painstakingly track the range of possible values and precision of each variable. This is in

start contrast to the more-commonly used floating-point arithmetic, where the number type itself automatically controls precision. A language for fixed-point arithmetic could allow the compiler to perform this task automatically, alleviating programmer cognitive burden. Consider a simple 8-bit multiply-add:

```
uint8_t a, b, c;
? d = a * b + c;
```

What is the correct output type for this operation? `a * b` *can overflow, so should be widened to 16 bits, and adding* `c` *requires another bit of precision. Therefore,* `d` *needs to be at least 17 bits. A correct program looks like:*

```
uint8_t a, b, c;
uint32_t d = (uint16_t)a * (uint16_t)b + (uint32_t)c;
```

A DSL for fixed-point arithmetic would allow the user to write just the mathematical expression, and the compiler would figure out types and when to insert widening. Alternatively, a DSL might help the programmer reason about approximate fixed-point computations. For instance, by deducing or allowing the programmer to specify operations which should intensionally lose some precision, and then bound the impact of that precision loss.

3.2 Brainstorming Example 1

One common game for children (and adults) is assembling LEGOs. In this, someone takes a set of small, simple pieces and follows either instructions or inspiration in order to assemble them into a larger, complex structure. Can a DSL be used to express LEGO assembly instructions? The DSL might be executed to produce a virtual representation of the final assembly, or to produce a rendering of visual instructions that a human could follow to build the assembly.

Consider the problem of describing this assembly. Perhaps it could be described by a program that formalizes the following pseudo-code:

```
pin p1, p2;
long l1, l2;
short s1, s2;
flat f1, f2;
insert p1 into hole 2 of l1;
insert p2 into hole -2 of l1;
insert p1 into hold 2 of l2 and p2 into hold -2 of l2;
check that l1 and l2 are adjacent, with shape 15(?)x2x1;
call this group b;
// ..continue..
```

In the case that a program produces a virtual assembly, that assembly could be (simplistically) rendered with pygame.

The basic nouns in this language appear to bricks (the things an assembled) and locations (used to specify what connects to what). The basic verbs are to connect bricks and to compute locations from bricks.

4 Proposal

Your proposal will involve choosing one specific problem domain, and providing a language sketch for your DSL. Your submission should be a 2-3 page report providing a sketch of your language, and multiple examples of programs in the language, highlighting more examples in the problem domain than in the brainstorming submission.

5 Demo

You will demonstrate some working components of your language implementation to a small group of your peers and the teaching staff. This is a live (in-class) presentation. Please prepare enough content (slides, live programming, etc) for 5 minutes, followed by 3 minutes of Q&A. This is a good opportunity to refine the sketch from your proposal, discuss any possible issues you have encountered, and get feedback on your design. You will be expected to pay attention to your peers' presentations, and provide feedback to them as well. The lecture following the Demos (May 29nd), we will break up into workshop groups to give hands-on language feedback.

5.1 Feedback Groups

We will place students in groups of 4–5 in order to provide consistent peer feedback. These feedback groups are responsible for providing useful feedback on using your language after the Demo, during the studio time in the following week of classes.

5.2 Writer's Workshop

During the feedback session (May 29nd), your peers will take turns trying to use your language. You will observe their attempts without speaking or providing assistance, and should just take notes on the difficulties your peers face. This format is designed to provide authentic feedback from real users. We expect each group to spend approximately 10 minutes per DSL.

6 Presentation

Your final presentation will be performed during one of the last two class lectures. This is expected to be a 5-10 minute presentation, followed by 3 minutes of Q&A. This should be more refined than your demo, and you should discuss what worked well *and what didn't*. We expect you to provide verbal feedback to at least one member of your feedback group during their Q&A.

7 Final Submission

Your final submission contains three parts:

1. Link to a repository containing your language implementation.

2. A language tutorial walking a user through writing programs in your language, in Markdown (as is standard in many open-source project repos).

3. A report (4–8 pages) detailing your language and your implementation of that language.

Your tutorial should not just involve writing the equivalent "Hello World" program (though it should include this, if such an example exists), but should also provide examples of *each* operator in your language, and a walkthrough of a sufficiently complicated program.

Your report should provide a thorough high-level description of the language and implementation, with figures detailing any necessary compiler details. You should describe which problems in your chosen domain are expressible, and which *are not*. Your report should clearly show how problems in your chosen domain map to language constructs in your DSL. Additionally, you should include a short related works section (if applicable), *and evaluation*, whether that is a user report, performance numbers, or otherwise.

References

[1] Wikipedia. Fixed-point arithmetic. `https://en.wikipedia.org/wiki/Fixed-point_arithmetic`, 2024. [Online; accessed 11-February-2024].