THE HUSKY PROGRAMMING LANGUAGE

TECHNICAL REPORT

Xiyu Zhai

ABSTRACT

The Husky programming language is new programming language designed for next-generation AI and software.

Keywords First keyword · Second keyword · More

1 Introduction

This is introduction.

Rust Bugden and Alahmar [2022].

2 Super Computation Graph

Super computation graph in the simplest form. Let G be a directed graph with vertices V. We associate each vertex $v \in V$ with a set S_v called the **value set**.

Let V^{sources} be all the source vertices and let $S^{\text{source}} := \prod_{v \in V^{\text{sources}}} S_v$ be the cartesian product of all the value sets associated with source vertices.

Let V_v^{incoming} be all the incoming vertices of some non-source vertex $v \in V$, and let $S_v^{\text{incoming}} := \prod_{v \in V_v^{\text{incoming}}} S_v$ be the product of all the value sets associated with incoming vertices of v.

Given assigments of generators g_v for each non-source $v \in V \setminus V^{\text{source}}$ a function of domains and codomains to be specified later, we aim to construct $f_v : S^{\text{source}} \to S_v$ for each $v \in V$, as follows,

- First, we specify that for each source vertex $v \in V^{\text{source}}$, f_v is the projection from S^{source} to S_v.
- Then, we specify that for each non-source vertex $v \in V \setminus V^{\text{source}}$,

$$f_v := g_v(f_v^{\text{incoming}}) \circ f_v^{\text{incoming}} \tag{1}$$

  where

$$f_v^{\text{incoming}} := \prod_{v' \in V_v^{\text{incoming}}} f_{v'}, \tag{2}$$

  which can be identified as a function from S^{source} to $\prod_{v' \in V_v^{\text{incoming}}} S_{v'}$.

The signature of g_v is thus determined to be $g_v : \prod_{v' \in V_v^{\text{incoming}}} S_{v'}^{S^{\text{source}}} \to S_v^{S_v^{\text{incoming}}}$.

The key takeaway is that we don't specify f_v by simply composing those $f_{v'}$ with v' incoming vertices of v with a fixed function, we specify by generating a function based on incoming vertices and then compose. By setting g_v to be a constant function over the first argument, we recover the ordinary computation graph.

The following C code implements.

```
int x = 0;

int y() { return x + 1; }

int y_with_x(int new_x) {
    int old_x = new_x;
    x = new_x;
    int y_value = y();
    x = old_x;
    return y_value;
}

int main() {
    printf("y equals %d\n", y());
    printf("y with x = 2 equals %d\n", y_with_x(2));
    printf("y equals %d\n", y());
}
```

References

William Bugden and Ayman Alahmar. Rust: The programming language for safety and performance, 2022.