University of Bergen

A Web Based Command Line Interface for Specifying Multiway Dataflow Constraint Systems

Bo Victor Isak Aanes

June 19, 2023

INF319 Project

Contents

1 Introduction

Graphical user interfaces (GUIs) are prevalent in most of today's technology that involves user interaction. However, developing and maintaining GUIs can still be difficult. Much of this is because as a GUI involves a larger set of state, the business logic neede in order to keep track of this state quickly becomes complex. This problem grows even bigger when we introduce dependencies between elements of the GUI. Multiway dataflow constraint systems (MDCS) is a programming model designed to let a developer specify dependencies between elements of the GUI using constraints [5]. This is usually done declaratively and MDCS will handle all business logic related to these constraints.

2 Background

2.1 Multiway Dataflow Constraint Systems

A multiway dataflow constraint system consists of a set of variables and a set of constraints. A variable holds a value which can be modified, and a constraint consists of a set of methods which contain expressions that can modify a variable.

Libraries that implement MDCSs usually also handles enforcing of constraints, such that whenever a value gets updated by a user the constraint system automatically satisfies all its constraints. The main part of *solving* a constraint system is achieved through what is called *planning* [7]. When a variable is updated by a user, the constraints involving this variable may no longer be satisfied, that is, that the relations specified in the methods no longer hold. As such, we have to select the appropriate methods to invoke in order to enforce the given constraints, such that the constraint system is satisfied. This is what the planner does.

When a user updates a variable, they will expect that the constraint system does not overwrite that variable instantly. Because of this we do not want the planner to select a method writing to this variable, but rather a method that reads from it. I.e., we want to select the methods from the constraints that result in the least surprising outcome to the user. This is what is known as a *hierarchical* planning algorithm [7].

The hierarchical planner assigns a strength to each variable, which gets updated when a user modifies a variable. The variable with the highest strength is the one that was updated most recently, and the variable with the lowest strength was the one updated least recently. By assigning a strength to the variables, the planner can make selections on which variables to preserve by selecting the appropriate methods.

When the appropriate methods are selected, they are invoked in the topological order of the dataflow graph. This is done to ensure that all

variables are updated in the correct order, such that no variable is updated before its dependencies are updated.

2.2 HotDrink

HotDrink is a JavaScript library for specifying multiway dataflow constraint systems [4]. It provides both an API and a DSL for this purpose. Variables can be declared along with constraints between these, which in turn can be bound to specific HTML-elements. The constraints themselves contain *methods* which can hold expressions or JavaScript functions that return values which are written to specified variables. In Listing 1, we can see how we can use HotDrink's API and DSL to define a constraint system to perform currency conversion between EUR and NOK.

Listing 1: Currency conversion in HotDrink.

```
1  const constraintSystem = new ConstraintSystem();
2
3  const component = component`
4      var exchangeRate = 10;
5      var eur = 0;
6      var nok = 0;
7
8      constraint {
9          eurToNok(eur, exchangeRate -> nok) = eur *
               ↪ exchangeRate;
10         nokToEur(nok, exchangeRate -> eur) = nok /
               ↪ exchangeRate;
11     }
12 `;
13
14 constraintSystem.addComponent(component);
15 constraintSystem.update();
```

We first define a constraint system using the `ConstraintSystem` class. We then define a component using the `component` template literal, which contains specifications for a currency conversion using HotDrink's DSL. Here we define variables and a constraint containing two methods: one for converting EUR to NOK, and one for converting NOK to EUR. After specifying our component, we add this component to the constraint system and call `constraintSystem.update()` to enforce the constraint.

The variables defined in the constraint system in Listing 1 can be bound to elements in the document object model (DOM), e.g., two input fields. When the user changes one of these input fields, the other one's value will be updated to the correct value determined by the constraint system.

2.3 Structure Manipulation

WarmDrink is a domain specific language (DSL) for defining relation between components in constraint systems [8]. This allows for defining what we call *intercalating constraints* which can manipulate values between elements in structures such as lists or trees. An example of this could be a

multi-city flight itinerary [8]. Let us say the first flight is at 15th of June, then the next flight can be no earlier than 15th of June as well. This relation can be introduced as an intercalating constraint between all flights in flight itinerary. The date of each flight must be equal to or greater than the preceding flight's date. If one of the flight's date gets updated, then so must all the succeeding flights as well.

2.4 React

React is a JavaScript library for building user interfaces [3]. It is based on the concept of components, which are reusable pieces of code that can be composed together to build complex user interfaces. React components are written in JavaScript, and can be written using either a class-based or a functional approach. In Listing 2, we can see an example of a React component written using the functional approach. Notice how the `return`-function contains HTML-like syntax. This is known as JSX (or JavaScript XML), which is an extension of JavaScript's syntax that allows for using HTML directly in you JavaScript files. React then transpiles the code into regular JavaScript, which can be run in a web browser.

Listing 2: A React component written using the functional approach.

```
function MyComponent(props) {
    return (
        <div>
            <h1>{props.title}</h1>
            <p>{props.text}</p>
        </div>
    );
}
```

React components can be rendered to the DOM using the `ReactDOM.render` method. This method takes a React component and a DOM element as arguments, and renders the component to the DOM element. In Listing 3, we can see how we can render the component from Listing 2 to the DOM. Notice how we pass *props* from the parent component down to the child component within the JSX.

Listing 3: Rendering a React component to the DOM.

```
ReactDOM.render(
    <MyComponent title="Hello, world!" text="This is my first
        ↪ React component." />,
    document.getElementById('root')
);
```

2.5 Web Application Programming Interfaces

An application programming interface (API) is a set of rules and protocols for building software applications. It defines how different software components can interact with each other. This can for example be between a

frontend web application and a backend database. This is known as a Web API, or an HTTP API.

A web server can provide *endpoints* for a client to send requests to. These endpoints are URLs that the client can send HTTP requests to, and the server will respond with a HTTP response. The client can send different types of requests, such as GET, POST, PUT, and DELETE. These requests are used to retrieve, create, update, and delete data from the server, respectively. The server can then respond with different types of responses, such as 200 OK, 201 Created, 400 Bad Request, and 404 Not Found. These responses are used to indicate whether the request was successful or not, and can also contain data.

2.6 Haskell

Haskell is a purely functional programming language first released in 1990 [6]. It is a statically typed general purpose language which uses lazy evaluation and type inference. Lazy evaluation is a technique where the compiler does not evaluate expressions until they are needed. And type inference is a feature in the compiler which can infer the types of expressions without them having to be explicitly annotated in the actual code..

Haskell also provide a powerful type system, which allows for the creation of custom types and type classes. Type classes are similar to interfaces in object-oriented programming languages, and can be used to define functions that can be used on different types. For example, the `Show` type class defines a function `show` which can be used to convert a value to a string. While the `Eq` type class is used for determining equality between values.

Data types in Haskell can be considered algebraic data types. They can be either *product types* or *sum types*. Product types are types that contain multiple values, such as tuples and records. Sum types are types that can be one of multiple types, such as the `Maybe` type. The `Maybe` type can either be `Just a` or `Nothing`, where `a` is a type variable. This type is used to represent values that may or may not be present, such as the result of a database query.

3 Frontend

The frontend part of our CLI tool is written in TypeScript [2] with React as our frontend library of choice.The main component `App` is the root component of our frontend and embodies everyhting the user sees and interacts with. The user is presented with a split screen consisting of a two main components: the view of the constraint system, and the terminal, see Figure 1.

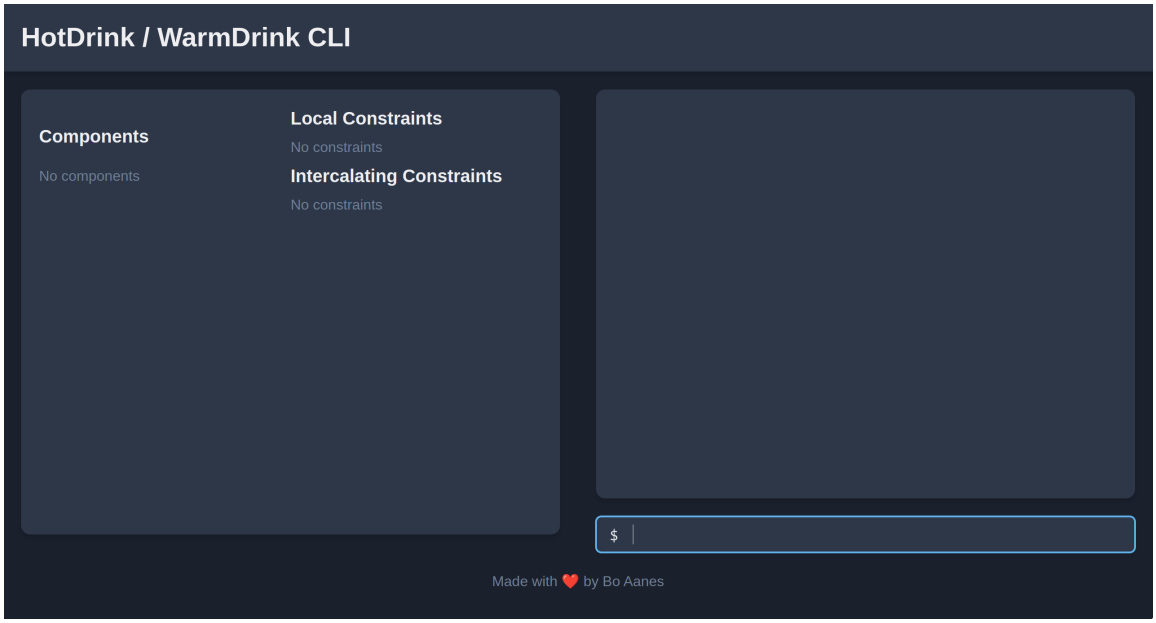

Figure 1: The view of the constraint system and the terminal.

In addition to the view of the program, the `App`-component also manages the state of the constraint system. We have defined types for components, constraints, methods, variables and component lists as can be seen in Listing 4.

To manage state in React we use a *hook* [1] called `useState`. `useState` is a function that takes an optional default value state and returns an array consisting of two elements. The first being the current state, and the second begin a function to update the state.

Listing 4: Types for constraint systems.

```
export interface Method {
  methodName: string;
  inputs: string[];
  expressions: Array<[string, string | undefined]>;
}

export interface Constraint {
  constraintName: string;
  methods: Method[];
}

export interface Variable {
  varID: string;
  varValue: number | boolean;
}

export interface IComponent {
  compID: number;
  variables: Variable[];
  strength: string[];
}

export interface ComponentList {
  components: IComponent[];
}
```

Listing 5: Using `useState` to manage state.

```
  const [components, setComponents] =
      ↪ useState<IComponent[]>([]);
  const [constraints, setConstraints] =
      ↪ useState<Constraint[]>([]);
  const [intercalatingConstraints,
      ↪ setIntercalatingConstraints] = useState<
    Constraint[]
  >([]);
  const [highlighted, setHighlighted] = useState<string[]>([]);
  const [enforceFromIndex, setEnforceFromIndex] =
      ↪ useState<number>(0);
```

References

[1] Introducing Hooks. https://legacy.reactjs.org/docs/hooks-intro.html. Accessed: 2023-06-19.

[2] TypeScript: JavaScript With Syntax For Types. https://www.typescriptlang.org/, 2023. Accessed: 19.06.2023.

[3] Alex Banks and Eve Porcello. *Learning React: modern patterns for developing React apps*. O'Reilly Media, 2020.

[4] John Freeman, Jaakko Järvi, and Gabriel Foust. HotDrink: A Library for Web User Interfaces. *SIGPLAN Not.*, 48(3):80–83, sep 2012.

[5] Magne Haveraaen and Jaakko Järvi. Semantics of multiway dataflow constraint systems. *Journal of Logical and Algebraic Methods in Programming*, 121:100634, 2021.

[6] Paul Hudak and Joseph H Fasel. A gentle introduction to Haskell. *ACM Sigplan Notices*, 27(5):1–52, 1992.

[7] Jaakko Järvi, Gabriel Foust, and Magne Haveraaen. Specializing Planners for Hierarchical Multi-Way Dataflow Constraint Systems. *SIGPLAN Not.*, 50(3):1–10, sep 2014.

[8] Knut Anders Stokke, Mikhail Barash, and Jaakko Järvi. A domain-specific language for structure manipulation in constraint system-based GUIs. *Journal of Computer Languages*, 74:101175, 2023.