Finite-State Transducers in OCamlFLAT/OFLAT

João Santos

NOVA University Lisbon

July 2025

Overview

- Context, motivation and objectives
- Background on FSTs
- The OCamlFLAT/OFLAT system
- Implementation plan
- Related tools

Context and Objectives

Context

- **Theory of Computation** is a foundational area in computer science.
- Formal Languages and Automata Theory (FLAT) is key to modeling computation and designing compilers.
- Educational tools like **OCamlFLAT/OFLAT** help visualize and simulate formal models.
- The system supports many models, but **lacks support for Finite-State Transducers (FSTs)**.

Objectives

- Extend OCamlFLAT/OFLAT with full support for FSTs.
- Implement core operations: translation, recognition, determinization, minimization.
- Design interactive FST support in the web interface (OFLAT).
- Ensure consistency with existing models and support exercises.

Motivation

- Formal Languages and Automata Theory are fundamental in CS education.
- Students struggle with abstract concepts.
- Visual and interactive tools help to understand these concepts.
- FSTs are underrepresented in pedagogical tools.
- OCamlFLAT/OFLAT lacks FST support.

What are Finite-State Transducers?

- FSTs are understood as an extension of finite automata by producing outputs during computation.
- They are defined over an input alphabet Σ and output alphabet Γ.
- **Two common cases**:
  - **Mealy machines:** output is produced on transitions.
  - **Moore machines:** output is associated with states.
- **General FSTs**:
  - Allow nondeterminism and ϵ-transitions.
  - Represented with a transition *relation* instead of a function.
- **Common operations:** union, concatenation, Kleene star, composition, projection.

What are Finite-State Transducers?

Definition:

- Formally, an FST is defined as a 6-tuple $(Q, \Sigma, \Gamma, I, F, \delta)$, where:
  - Q is a finite set, the set of **states**;
  - Σ is a finite set, called the **input alphabet**;
  - Γ is a finite set, called the **output alphabet**;
  - $I \subseteq Q$ is the set of **initial states**;
  - $F \subseteq Q$ is the set of **final states**;
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$ is the **transition relation**, where ε represents the empty string.
- Translates input strings into output strings.

OCamlFLAT/OFLAT System Overview

- **OCamlFLAT Library:**
  - Logic for formal models
  - Modular and extensible architecture based on OCaml.
- **OFLAT Web Application:**
  - Pedagogical and interactive browser interface.
  - Allows creation, editing and simulation of models.
  - Pedagogical exercises for training and self evaluation
  - Real-time visual feedback for word recognition and transducing.
  - The implementation follows the Model-View-Controller (MVC) pattern.
- **Pedagogical Goals:**
  - Help students visualize abstract computation steps.
  - Enable experimentation and learning through interaction.

Technologies Behind OFLAT

- **js_of_ocaml:**
  - Interoperability solution.
  - Compiles OCaml bytecode to JavaScript.
  - Supports bindings to native JS libraries (e.g., DOM, Cytoscape.js).
- **Cytoscape.js for Visualization:**
  - Used to draw graphs representing automata and trees.
  - Allows dynamic interaction: add/edit states, transitions, layout.
- **Web Foundations:**
  - HTML for structure, JavaScript for interactivity, DOM for manipulation.
  - OFLAT dynamically updates based on user actions.

Related Tools

- **JFLAP**
  - Comprehensive desktop application for FLAT.
  - Supports Mealy and Moore machines, but lacks general FST support.
  - Includes conversion tools, extensive model support, and simulations.
  - Lacks web-based accessibility.
- **FAdo**
  - Python-based command-line tool for formal languages and automata.
  - Supports transducers.
  - Mostly suited for scripted or batch experiments.

Related Tools

- **Automaton Simulator**
  - Simple Java-based simulator for DFA, NFA and PDAs.
  - No support for FSTs.
- **Flap.js**
  - Web-based tool for FA, PDA and RE.
  - Support input testing, but lacks features like animations.
  - Lack of FST support
  - Interface is unusual but intuitive.

Critical Discussion

- OCaml: Functional, strong typing and pattern matching.
- Visual tools improve understanding.
- OFLAT bridges theory and practice.

Work Plan Summary

Phase	Duration
OCamlFLAT FST logic	8 weeks
OFLAT integration	6 weeks
Pedagogical exercises	1 week
Evaluation	1 week
Writing	8 weeks (overlapping)

Expected Contributions

- New FST module in OCamlFLAT.
- Visual simulation in OFLAT.
- New exercises for teaching.