Building Moses Training Pipelines with Arrows

Jie Jiang
David Kolovratnik
Ian Johnson

Arrows and Pipelines

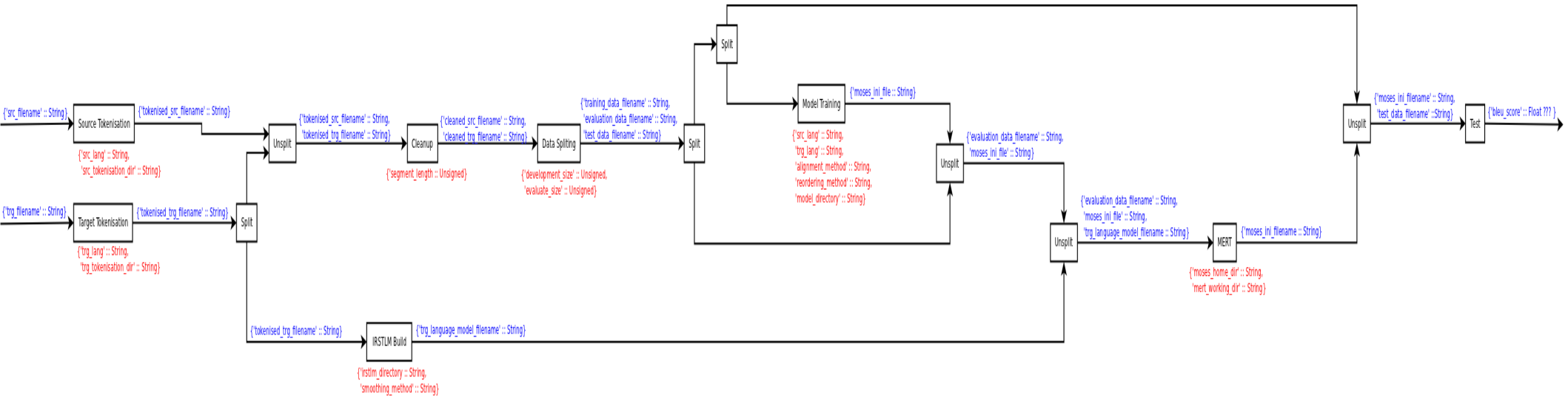
- Arrows are abstractions of computation
- Present a common interface to the world
- They compose
- An arrow represents a pipeline component
  - Composable components
- Benefits:
  - Collaboration: sharing pipelines or components.
  - Development: pipeline components can be added, updated, removed with minimal effort.
  - Modular: high cohesion, low coupling

Aims

- Assessing the Python based Arrows library
  - Arrows are implemented fully in Haskell
  - We only implement what we need in Python
- Can we build a Moses training pipeline?
- If we do build a training pipeline does it work?
- Does the library provide an “easy” abstraction?
- Is it flexible enough to build other pipelines?
  - From scratch
  - Adding to existing pipelines

Progress so far

- Defined a training pipeline
  - As simple as we can get it
  - Yet it'll do something useful
  - Defined the data and state



More Progress

- Implementing training components
  - Based on the existing Moses scripts
  - Knowledge of how to “talk to” scripts is in component
  - Where possible scripts are replaced by functions
  - “Wiring” of components is user defined.
- Cleanup component
  - A simple python function
  - Replaces a script
- Tokeniser
  - Uses the tokeniser script, i.e. subprocess

Even more progress

- A pipeline manager script has been started
  - Defines a configuration object
  - Defines pipeline topology description objects
  - Build pipeline components dynamically
  - Wires the components together
  - Executes the pipeline

Last bit of progress

- Pypeline library
  - Bugs fixed
  - Enhancements added to accommodate use cases
  - More unit testing
- On Github
  - https://github.com/ianj-als/pypeline