Programming Language Support for Natural Language Interaction

Alex Renda
Cornell University

Harrison Goldstein
Cornell University

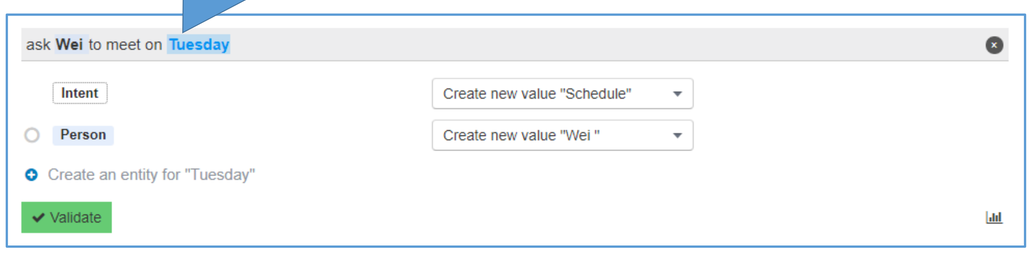
Sarah Bird
Facebook

Chris Quirk
Microsoft Research

Adrian Sampson
Cornell University

A **DSL** for configuring Natural Language Understanding models that ensures **consistency** and **type safety**...

Configuring Wit.ai generally requires manual highlighting of entities.




```
free-text Person;
free-text Time;
keywords Day = "Sunday" | "Monday" | ... | "Saturday";
alias Date = { day: Day, time?: Time };
trait Intent =
  | <Schedule> { who: Person, when: Date }
  | <Move> { from: Date, to: Date }
  | <List> {};
```

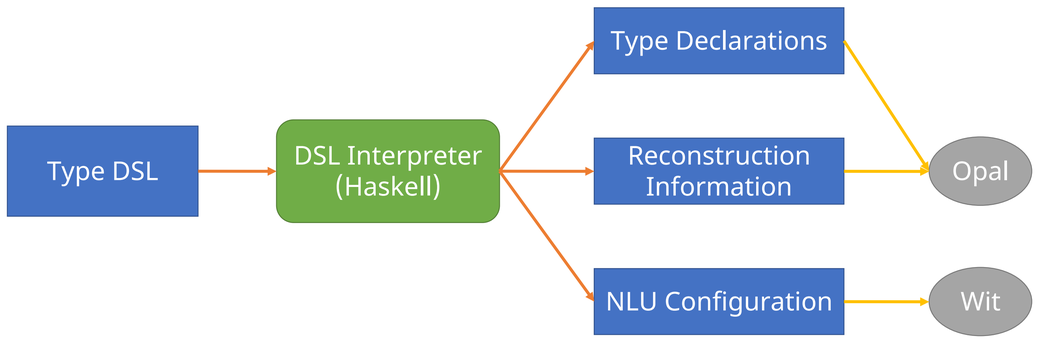
Our type DSL provides a way to configure Wit alongside application types, avoiding the graphical interface.

"ask Wei to meet on Tuesday"

```
{
  Intent: ["Schedule"],
  Person: ["Wei"],
  Day: ["Tuesday"]
}
```

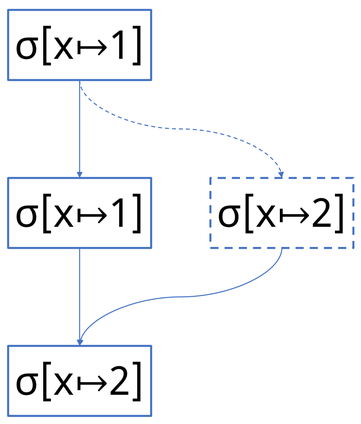
```
{
  tag: "Schedule",
  data: {
    who: "Wei",
    when: { day: "Tuesday" }
  }
}
```

Once the application has been configured, we provide a mechanism for building the application type from Wit's response.



... and a new language construct to express **ambiguity** by exploring **hypothetical actions**.

```
x := 1;
world := hyp {
  x := 2;
  // Local effects are visible.
  assert x == 2;
}
// The world has not yet been committed.
assert x == 1;
commit world;
// The world's changes have been merged.
assert x == 2;
```



"remind me to email Jean at 8:00"

```
remind(emailPerson("Jean"), 8)
remind(emailPerson("Gene"), 8)
remind(emailPerson("Jean"), 20)
remind(emailPerson("Gene"), 20)
```

```
interps := parse(utterance);
worlds := search (likelihood, name, time) in interps {
  person := contacts.get(name);
  calendar.schedule_meeting(person, time);
  fitness := calendar.fitness() * likelihood;
}
commit worlds.max(fitness);
```

ω is a map of hypothetical stores

$$\frac{\langle c, \sigma_{\text{orig}}, \varnothing, \mu\rangle \Downarrow \langle \sigma_{\text{hyp}}, \omega_{\text{hyp}}\rangle}{\langle u := \texttt{hyp } \{\ c\ \}, \sigma_{\text{orig}}, \omega, \mu\rangle \Downarrow \langle \sigma_{\text{orig}}, \omega[u \mapsto \sigma_{\text{hyp}}]\rangle}\ \textsc{Hyp}$$

μ specifies how to merge two stores

$$\frac{\omega[u] = \sigma_{\text{hyp}} \qquad \begin{array}{c}\forall v \notin \sigma_{\text{hyp}}.v \notin \sigma_{\text{merge}} \\ \forall v \in \sigma_{\text{hyp}}, \sigma_{\text{merge}}[v] = \mu(\sigma_{\text{curr}}[v], \sigma_{\text{hyp}}[v])\end{array}}{\langle \texttt{commit } u, \sigma_{\text{curr}}, \omega, \mu\rangle \Downarrow \langle \sigma_{\text{merge}}; \sigma_{\text{curr}}, \omega\rangle}\ \textsc{Commit}$$

https://capra.cs.cornell.edu/research/opal