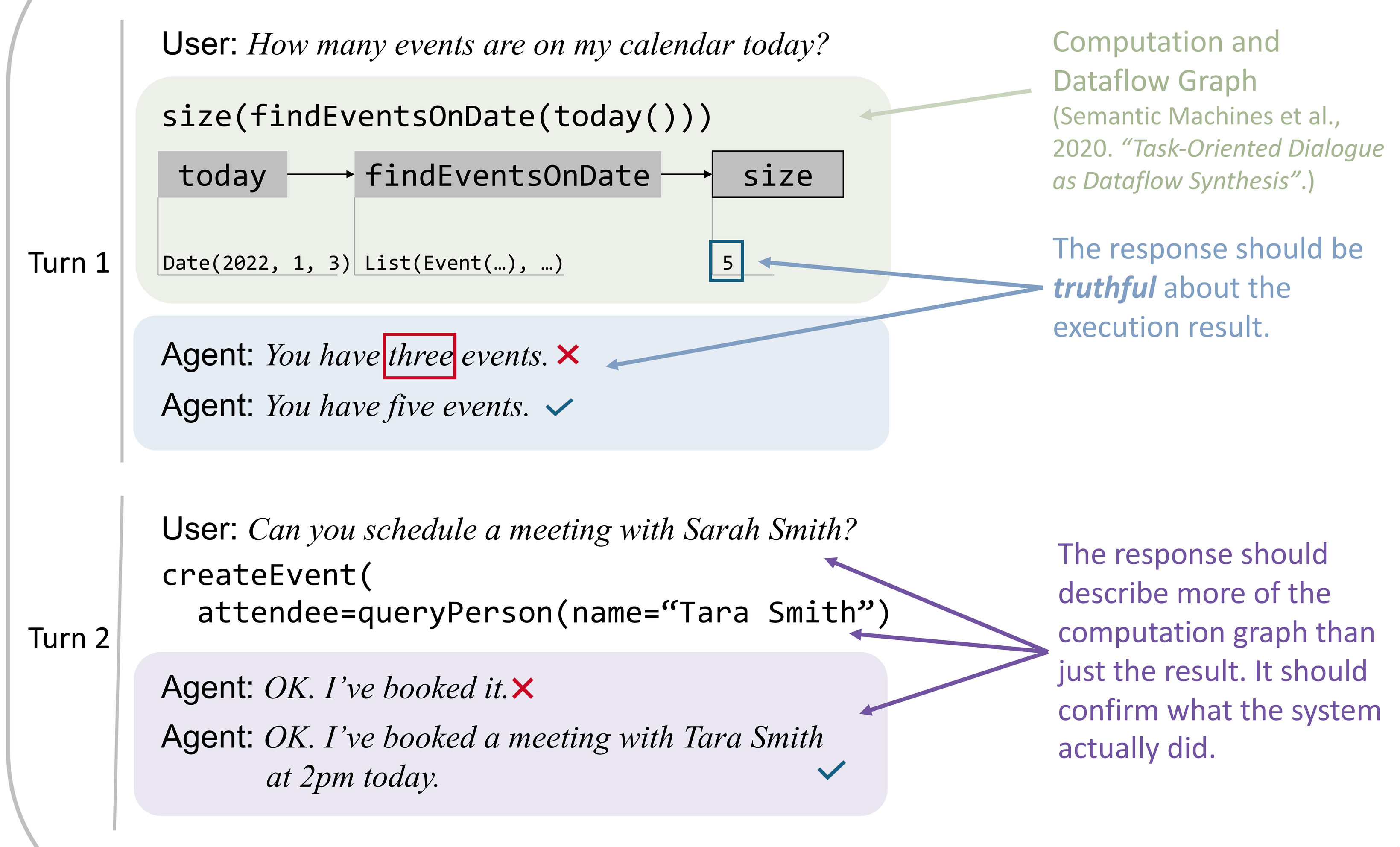
The Whole Truth and Nothing But the Truth: Faithful and Controllable Dialogue Response Generation with Dataflow Transduction and Constrained Decoding

Microsoft semanticmachines

Hao Fang*, Anusha Balakrishnan*, Harsh Jhamtani*, John Bufe, Jean Crawford, Jayant Krishnamurthy, Adam Pauls, Jason Eisner, Jacob Andreas, Dan Klein



Two Predominant Paradigms for Dialogue Response Generation

Neural Language Modeling
- Produce fluent, coherent, and diverse responses.
- Can leverage pre-trained large language models (e.g., GPT-3, ChatGPT).
- Issues:
  - Suffer from hallucination.
  - Struggle in maintaining faithfulness.
  - Produce unsafe responses.
  - Difficult to control.

Rule-Based Generation
- Easy to control (by modifying rules).
- Safe for production (can only produce responses allowed by hand-written rules).
- Issues:
  - Hard to maintain for complex domains.
  - Require extensive domain knowledge, including both low-level details like grammar and high-level properties like truthfulness.

Our Framework: A Hybrid Approach for Response Generation

Dataflow Transducer — (Dataflow Transduction) Transduce a dataflow graph into a context-free grammar.

\+

Language Model — (Constrained Decoding) Generate responses constrained by the context-free grammar.

The context-free grammar (CFG) defines the space of all responses allowed for the given computation (dataflow graph).

Responses are *truthful* but not always *grammatical* or *natural*.
- ✓ *I found 1 event on Thursday. It's "Show and Tell".*
- ✗ *I found 1 events on Thursday.*
- ✗ *The "Show and Tell" meeting on Thursday starts on Thursday.*

Hybrid generation has a long history in NLP, dating back to Knight & Hatzivassiloglou (1995) and Langkilde & Knight (1998).

Dataflow Transducer$(\mathcal{T}, \Sigma, \mathcal{R}, t_{start})$

Nonterminal Types ($t \in \mathcal{T}$)
- S: the start nonterminal t_{start}
- PP, NP, …: syntactic categories
- EVENT, …: semantic categories
- LEX: lexicalization

Terminals ($w \in \Sigma$)
Word types (*"I"*, *"found"*, …)

Transduction Rules ($r \in \mathcal{R}$)
Applied to a dataflow node v to create a QCFG production $(t, v) \to \beta_1\beta_2 \dots \beta_N$

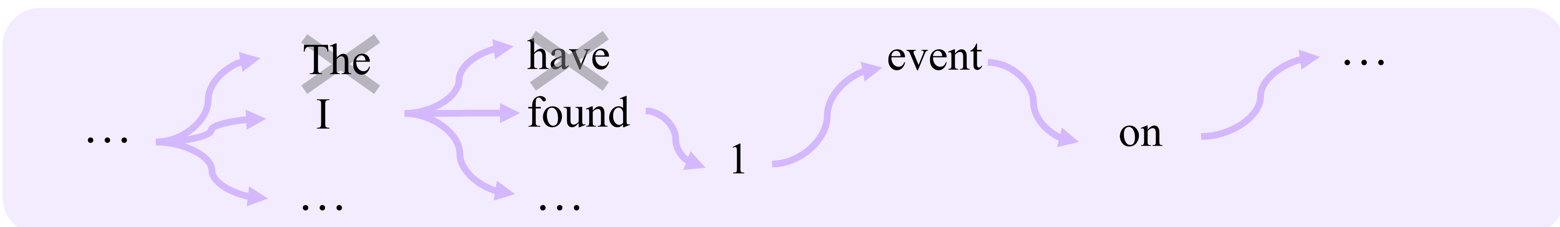
*: QCFG is a special type of CFG (more details in our paper).

Dataflow Transduction Procedure

- Each dataflow node is converted to one or more specialized nonterminals, which expand to natural language descriptions of *that node*.
  - Descriptions are nested: they can recurse to descriptions of neighboring nodes.
  - Neighboring nodes may be added on demand.
  - See a full example at the bottom of the poster.
- There may be multiple transduction rules for each QCFG nonterminal and the QCFG may admit combinatorially many derivation trees.
- Each derivation tree derives a truthful response. But they vary in their information content, presentation order, linguistic style, and choice of terminals.
- In this paper, we use a neural LM with constrained decoding to select a fluent and appropriate response from all these truthful responses.

Constrained Decoding

- Generate response candidates from a neural LM (pre-trained and preferably fine-tuned), constrained by the QCFG.



Shin et al., 2020. *"Constrained Language Models Yield Few-Shot Semantic Parsers"*.

- Can be efficiently performed via an incremental context-free parsing algorithm (Earley, 1970) using the parsing state of the prefix.

Dataflow Transduction Rules

The rule head is a nonterminal type $t \in \mathcal{T}$.
The rule body checks application conditions and extracts argument via structural pattern matching.
Introduce new nodes to the dataflow graph and bind them with variables in the response template.

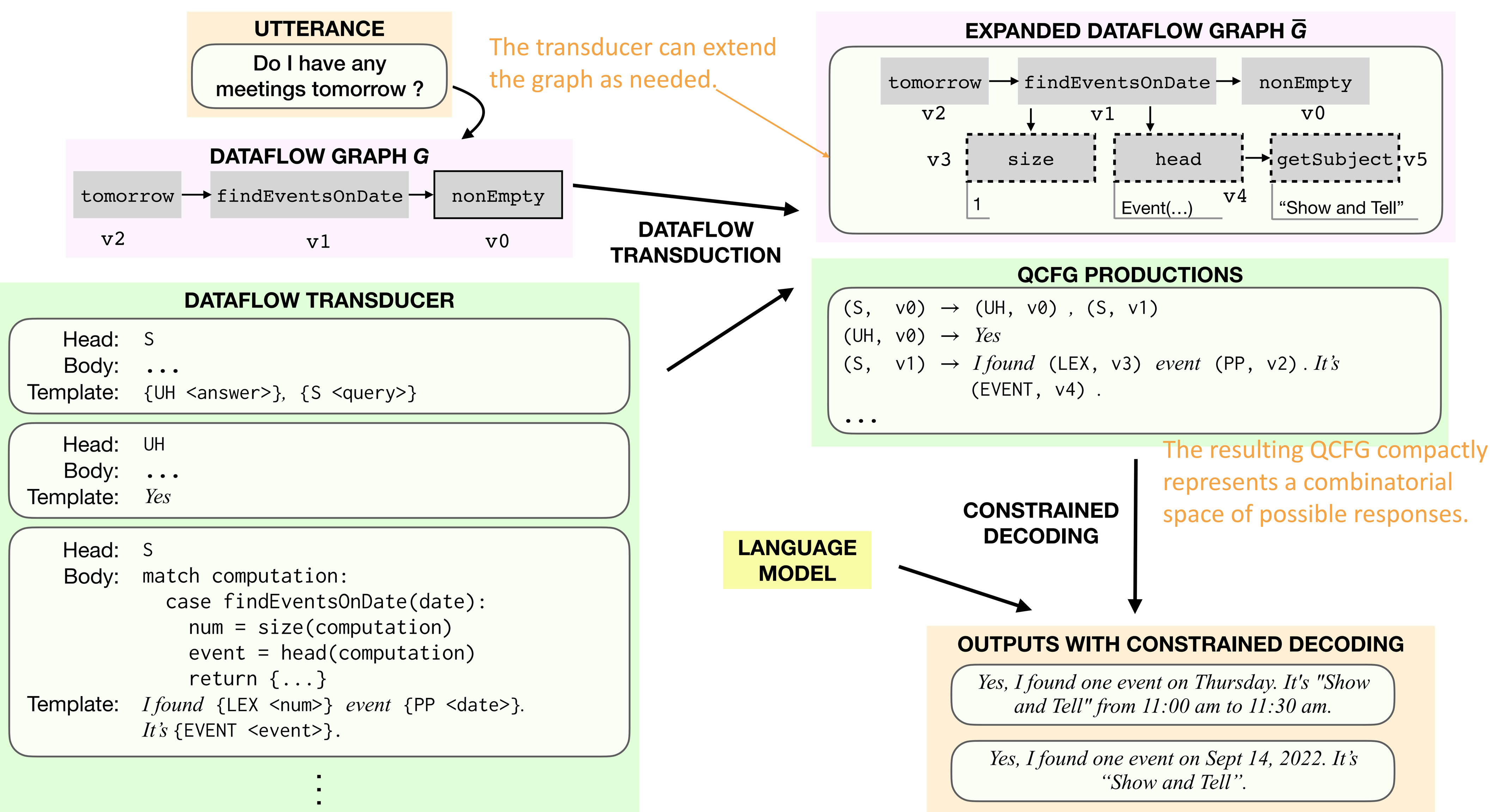
Head: S
Body:
```
match computation:
  case findEventsOnDate(date):
    num = size(computation)
    event = head(computation)
    return {"num": num, "event": event, "date": date}
```
Response Template:
I found {LEX <num>} *event* {PP <date>}. *It's* {EVENT <event>}.

Indicate the computation is described as a prepositional phrase.
Check whether the computation is a call of the given function and extract the argument month.
The response template defers describing the month to appropriate NP rules.

Head: PP
Body:
```
match computation:
  case FullMonthofPreviousMonth(month):
    return {"month": month}
```
Response Template:
last {NP <month>}

Check the value of the computation rather than its structure.
The response template has no nonterminal, so the body returns an empty binding.

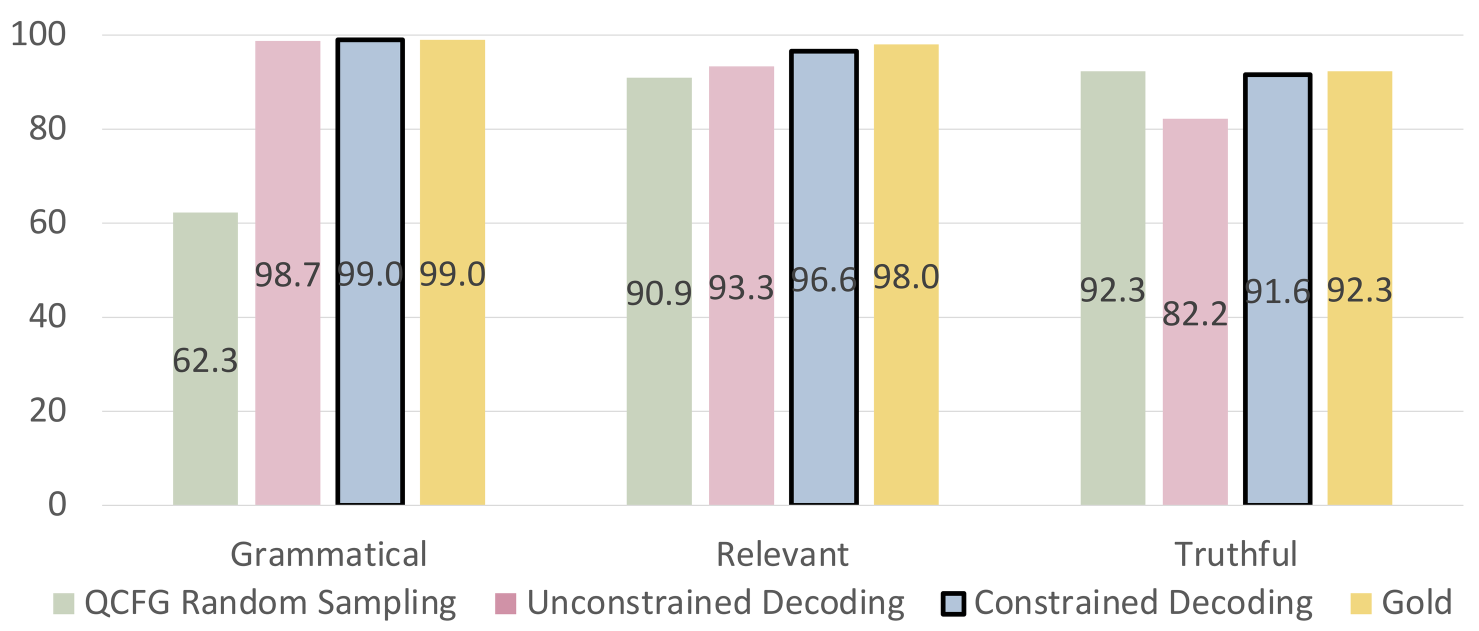
Head: NP
Body:
```
if computation.__value__ == Month.March:
  return {}
```
Response Template:
March



Data and Human Evaluation

SMCalFlow2Text
- A subset of SMCalFlow examples involving calendar event queries.
- 8938 training examples, 1041 validation examples, with meta information for executing the dataflow programs.
- 187 transduction rules (written by some of us in a matter of hours) sufficient to cover all gold agent responses.

Human Evaluation
- Grammaticality (*"has the virtual assistant made any grammar errors?"*)
- Relevance (*"has the virtual assistant misunderstood the user's request?"*)
- Truthfulness (*"has the virtual assistant provided any incorrect information as judged using the database and timestamp?"*)



- Random sampling from the QCFG can produce ungrammatical but relevant and truthful responses.
- Unconstrained decoding with dataflow transduction produce grammatical and relevant responses, but it scores low on truthfulness.
- Constrained decoding produces grammatical, relevant, and truthful responses (very close to gold references).

Conclusion

- A hybrid approach for building dialogue response generation systems.
  - Developers can write transduction rules to *truthfully* describe computations.
  - Surface realization decisions are deferred to a flexible language model.
- The proposed approach outperforms unconstrained conditional language modeling in both automatic and human evaluations, especially on *truthfulness*.
- Several expert hours spent on authoring rules hold almost equivalent value to a large volume of training data.
- Code and data: https://github.com/microsoft/dataflow2text