Converting RDF Data into MRS for generation

Liz Conrad — DELPH-IN 2023

Generating Referring Expressions

- GOAL: Generate a variety of referring expressions for different entities
  - indefinite expressions
  - definite expressions
  - pronouns
  - etc.

Graph Based Generation

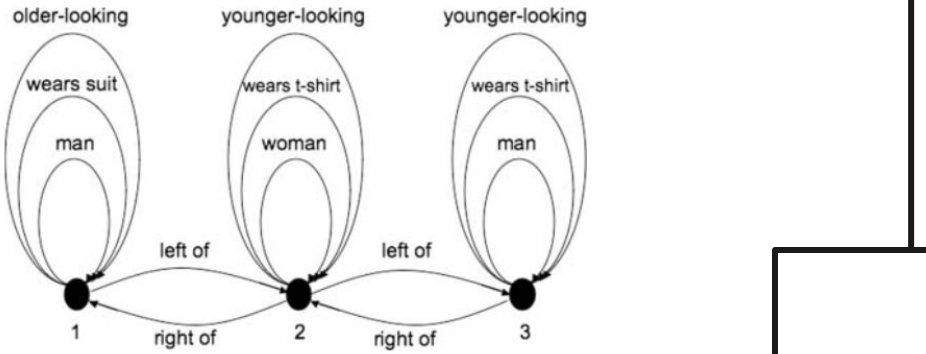


Figure 4
Representation of our example scene in Figure 1 as a labeled directed

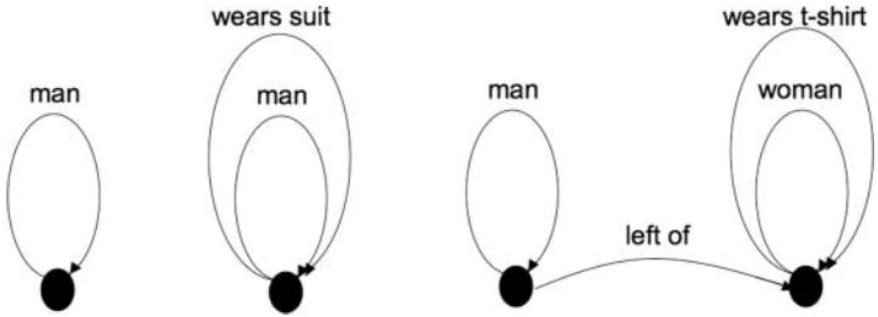
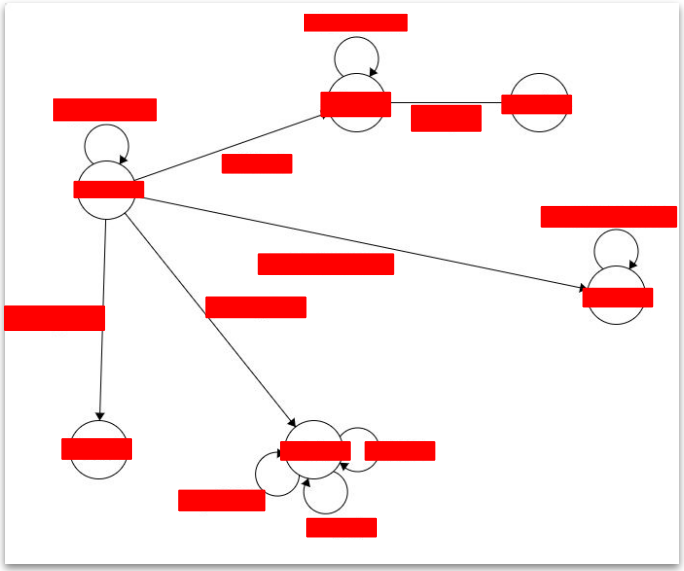


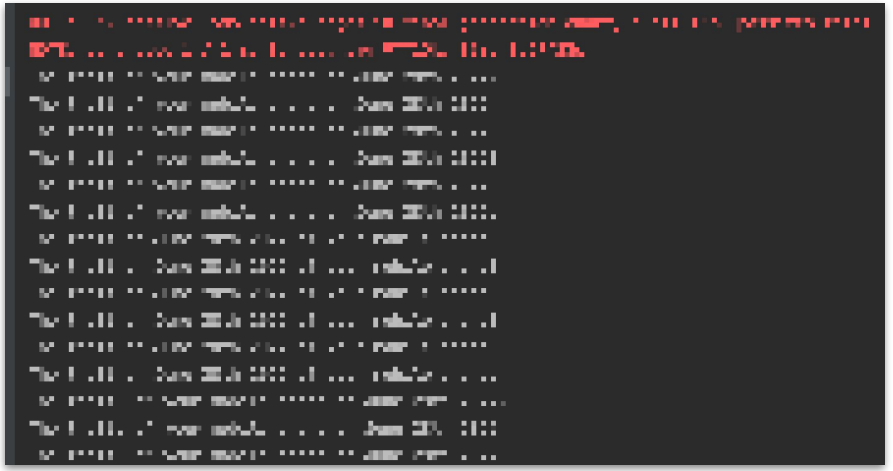
Figure 5
Some referring graphs for target d_1.

Krahmer and Deemter (2012)

General Idea



Graph representation of some entity



Resulting referring expressions, generated by the ERG from MRS

General Pipeline

1. Obtain a graph representation of an entity
2. Convert the graph to MRS using three levels of rules
    a. LOW LEVEL — 'nitty-gritty' rules to combine MRS fragments (e.g. via intersective combination)
    b. MID LEVEL — rules for particular syntactic/semantic phenomena that use the low level rules
    c. HIGH LEVEL — rules for the domain of interest that use the mid level rules
3. Generate English text from the MRS using the ERG

Low Level Rules

An Algebra for Semantic Construction in Constraint-based Grammars

Ann Copestake
Computer Laboratory
University of Cambridge
New Museums Site
Pembroke St, Cambridge, UK
aac@cl.cam.ac.uk

Alex Lascarides
Division of Informatics
University of Edinburgh
2 Buccleuch Place
Edinburgh, Scotland, UK
alex@cogsci.ed.ac.uk

Dan Flickinger
CSLI, Stanford University *and*
YY Software
Ventura Hall, 220 Panama St
Stanford, CA 94305, USA
danf@csli.stanford.edu

Abstract

We develop a framework for formalizing semantic construction within grammars expressed in typed feature structure logics, including HPSG. The approach provides an alternative to the lambda calculus; it maintains much of the desirable flexibility of unification-based approaches to composition, while constraining the allowable operations in order to capture basic generalizations and improve maintainability.

4. All signs have an index functioning somewhat like a λ-variable.

A similar approach has been used in a large number of implemented grammars (see Shieber (1986) for a fairly early example). It is in many ways easier to work with than λ-calculus based approaches (which we discuss further below) and has the great advantage of allowing generalizations about the syntax-semantics interface to be easily expressed. But there are problems. The operations are only specified in terms of the TFS logic: the interpretation relies on an intuitive cor-

Low Level Rules

```
def intersective(hole_ssement, plug_ssement, arg_name, lbl_identity, head):
    """
    TOP = LBL/TOP of head
    INDEX = INDEX of head
    RELS = sum of both RELS lists
    HCONS = sum of both HCONS lists
    ICONS = sum of both ICONS lists
    VARIABLES = combine both VARIABLBES dicts
    HOLES = HOLES list - ARG hole (which gets plugged)
    EQs = EQs + hole.TOP=plug.TOP + hole.ARG.variable=plug.INDEX

    :param hole_ssement: SSEMENT object with the hole being filled
    :param plug_ssement: SSEMENT object plugging the hole
    :param arg_name: ARG that is the hole
    :param lbl_identity: whether the hole and plug should have their labels identified
    :param head: which SSEMENT object is the semantic head. Either member of the Fragment enumeration (Fragment.hole or Fragment.plug)
    :return: the new combined SSEMENT
    """
```

Low Level Rules

```
def scopal_quantifier(scoping_ssement, scoped_ssement):
    """
    Scopal rules for quantifiers (plug both the ARG0 and RSTR holes)

    TOP = new LBL
    INDEX = INDEX of scoped
    RELS = sum of both RELS lists
    HCONS = sum of both HCONS lists + scoping.holes.RSTR=scoped.TOP
    ICONS = sum of both ICONS lists
    VARIABLES = combine both VARIABLBES dicts
    HOLES = HOLES list - ARG0 hole of scoping (which gets plugged) - RSTR hole of scoping (plugged via qeq)
    EQs = EQs + hole.TOP=plug.TOP + scoping.ARG0.variable=scoped.INDEX

    :param scoping_ssement: SSEMENT object doing the scoping
    :param scoped_ssement: SSEMENT object being scoped
    :return: New combined SSEMENT object
    """
```

Mid Level Rules

```
# PARTS OF SPEECH
Liz Conrad
def noun(noun_pred, constraints=None):
    return ep(noun_pred, constraints)


Liz Conrad
def verb(verb_pred, constraints=None):
    return ep(verb_pred, constraints)


Liz Conrad
def pronoun(constraints=None):
    pron_ep = ep("pron", constraints)
    pron_q = ep("pronoun_q")

    pron_mrs = quant_phr(pron_q, pron_ep)
    return pron_mrs


Liz Conrad
def adjective(adj_pred):
    return ep(adj_pred)


Liz Conrad
def preposition(prep_pred):
    return ep(prep_pred)


Liz Conrad
def quantifier(quant_pred):
    return ep(quant_pred)
```

Mid Level Rules

Liz Conrad

```
def quant_phr(quant_ssement, noun_ssement):
    # perform scopal combination with the quantifier and the noun
    return scopal_quantifier(quant_ssement, noun_ssement)
```

Liz Conrad

```
def adj_phr(adj_ssement, noun_ssement):
    # perform intersective combination with the ADJ and NOUN
    # plug ADJ.ARG1 with NOUN
    # identify labels
    # assign NOUN (plug) as the HEAD
    return intersective(adj_ssement, noun_ssement, "ARG1", True, Fragment.PLUG)
```

Mid Level Rules

Liz Conrad

```
def poss_phr(possessor_ssement, possessee_ssement):
    # get poss EP
    poss = ep("poss", None, {"ARG0": "e", "ARG1": "u", "ARG2": "u"})
    # plug ARG1 hole with noun, identify labels, assign the noun (plug) as the head
    poss_arg1_plugged = intersective(poss, possessee_ssement, "ARG1", True, Fragment.PLUG)
    # plug ARG2 hole with pron_ssement, don't identify labels, assign the SSEMENT with the hole as the head
    # (i.e. the one with the noun at this point)
    poss_arg2_plugged = intersective(poss_arg1_plugged, possessor_ssement, "ARG2", False, Fragment.HOLE)

    # def_explicit_q with poss
    def_q = quantifier("def_explicit_q")
    final_poss = quant_phr(def_q, poss_arg2_plugged)

    return final_poss
```

$$
\begin{bmatrix}
\text{TOP} & h0 \\
\text{INDEX} & e2 \\
\text{RELS} & \left\langle
\begin{bmatrix} \textit{unknown}\langle 0{:}11\rangle \\ \text{LBL} & h1 \\ \text{ARG0} & e2 \\ \text{ARG} & x4 \end{bmatrix},
\begin{bmatrix} \textit{def\_explicit\_q}\langle 0{:}4\rangle \\ \text{LBL} & h5 \\ \text{ARG0} & x4 \\ \text{RSTR} & h6 \\ \text{BODY} & h7 \end{bmatrix},
\begin{bmatrix} \textit{poss}\langle 0{:}4\rangle \\ \text{LBL} & h8 \\ \text{ARG0} & e9 \\ \text{ARG2} & x10 \\ \text{ARG1} & x4 \end{bmatrix},
\begin{bmatrix} \textit{pronoun\_q}\langle 0{:}4\rangle \\ \text{LBL} & h11 \\ \text{ARG0} & x10 \\ \text{RSTR} & h12 \\ \text{BODY} & h13 \end{bmatrix},
\begin{bmatrix} \textit{pron}\langle 0{:}4\rangle \\ \text{LBL} & h14 \\ \text{ARG0} & x10 \end{bmatrix},
\begin{bmatrix} \textit{\_cookie\_n\_1}\langle 5{:}11\rangle \\ \text{LBL} & h8 \\ \text{ARG0} & x4 \end{bmatrix}
\right\rangle \\
\text{HCONS} & \left\langle
\begin{bmatrix} \textit{qeq} \\ \text{HARG} & h12 \\ \text{LARG} & h14 \end{bmatrix},
\begin{bmatrix} \textit{qeq} \\ \text{HARG} & h6 \\ \text{LARG} & h8 \end{bmatrix},
\begin{bmatrix} \textit{qeq} \\ \text{HARG} & h0 \\ \text{LARG} & h1 \end{bmatrix}
\right\rangle
\end{bmatrix}
$$

Mid Level Rules

Liz Conrad

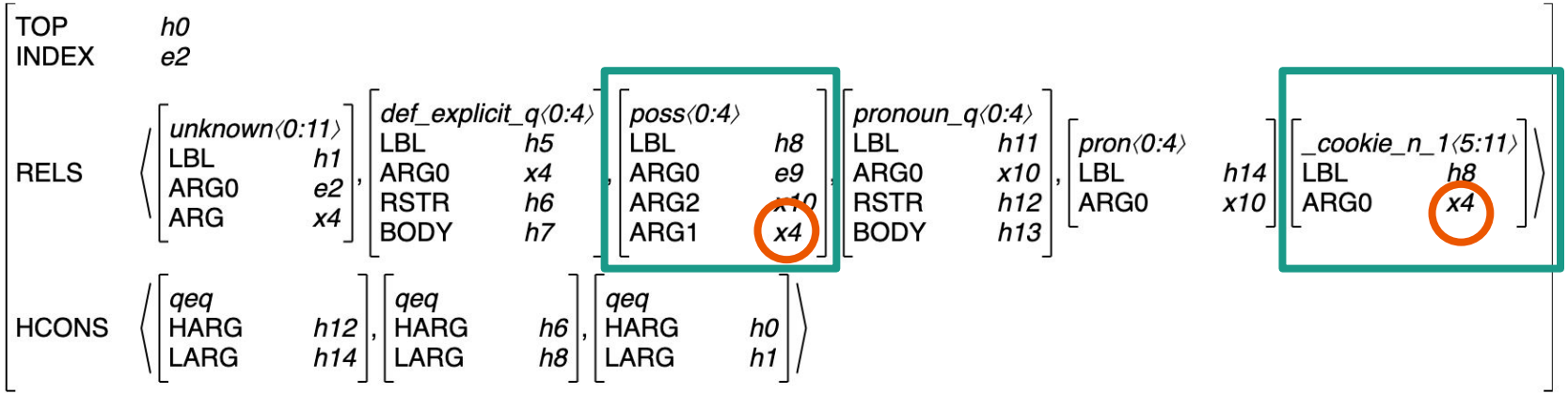
```
def poss_phr(possessor_ssement, possessee_ssement):
    # get poss EP
    poss = ep("poss", None, {"ARG0": "e", "ARG1": "u", "ARG2": "u"})
    # plug ARG1 hole with noun, identify labels, assign the noun (plug) as the head
    poss_arg1_plugged = intersective(poss, possessee_ssement, "ARG1", True, Fragment.PLUG)
    # plug ARG2 hole with pron_ssement, don't identify labels, assign the SSEMENT with the hole as the head
    # (i.e. the one with the noun at this point)
    poss_arg2_plugged = intersective(poss_arg1_plugged, possessor_ssement, "ARG2", False, Fragment.HOLE)

    # def_explicit_q with poss
    def_q = quantifier("def_explicit_q")
    final_poss = quant_phr(def_q, poss_arg2_plugged)

    return final_poss
```

$$
\begin{bmatrix}
\text{TOP} & h0 \\
\text{INDEX} & e2 \\
\text{RELS} & \left\langle
\begin{bmatrix} \textit{unknown}\langle 0{:}11\rangle & \\ \text{LBL} & h1 \\ \text{ARG0} & e2 \\ \text{ARG} & x4 \end{bmatrix},
\begin{bmatrix} \textit{def\_explicit\_q}\langle 0{:}4\rangle & \\ \text{LBL} & h5 \\ \text{ARG0} & x4 \\ \text{RSTR} & h6 \\ \text{BODY} & h7 \end{bmatrix},
\begin{bmatrix} \textit{poss}\langle 0{:}4\rangle & \\ \text{LBL} & h8 \\ \text{ARG0} & e9 \\ \text{ARG2} & x10 \\ \text{ARG1} & x4 \end{bmatrix},
\begin{bmatrix} \textit{pronoun\_q}\langle 0{:}4\rangle & \\ \text{LBL} & h11 \\ \text{ARG0} & x10 \\ \text{RSTR} & h12 \\ \text{BODY} & h13 \end{bmatrix},
\begin{bmatrix} \textit{pron}\langle 0{:}4\rangle & \\ \text{LBL} & h14 \\ \text{ARG0} & x10 \end{bmatrix},
\begin{bmatrix} \textit{\_cookie\_n\_1}\langle 5{:}11\rangle & \\ \text{LBL} & h8 \\ \text{ARG0} & x4 \end{bmatrix}
\right\rangle \\
\text{HCONS} & \left\langle
\begin{bmatrix} \textit{qeq} & \\ \text{HARG} & h12 \\ \text{LARG} & h14 \end{bmatrix},
\begin{bmatrix} \textit{qeq} & \\ \text{HARG} & h6 \\ \text{LARG} & h8 \end{bmatrix},
\begin{bmatrix} \textit{qeq} & \\ \text{HARG} & h0 \\ \text{LARG} & h1 \end{bmatrix}
\right\rangle
\end{bmatrix}
$$

Mid Level Rules

Liz Conrad

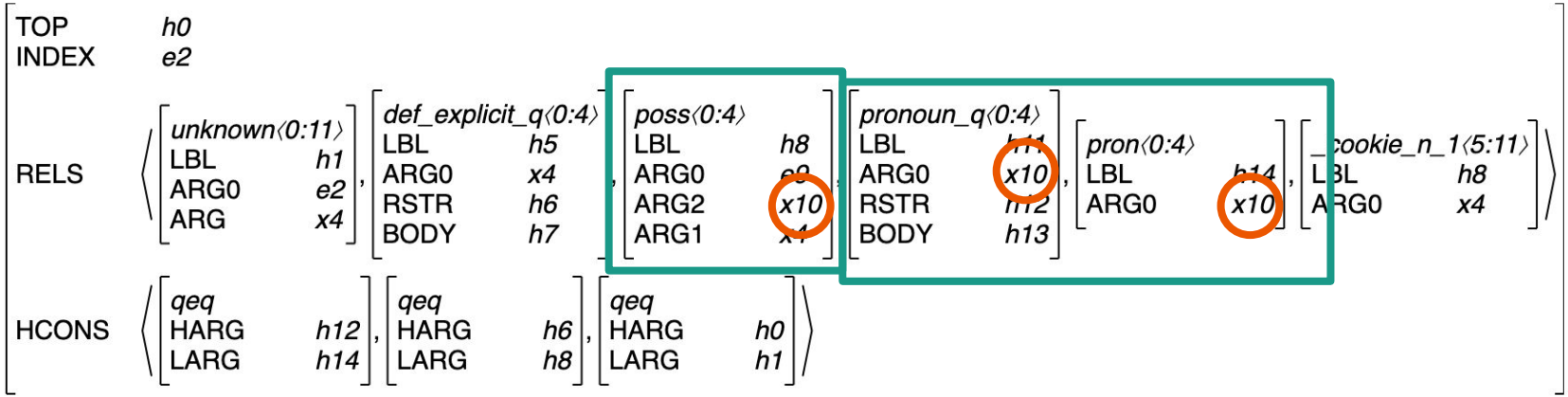
```
def poss_phr(possessor_ssement, possessee_ssement):
    # get poss EP
    poss = ep("poss", None, {"ARG0": "e", "ARG1": "u", "ARG2": "u"})
    # plug ARG1 hole with noun, identify labels, assign the noun (plug) as the head
    poss_arg1_plugged = intersective(poss, possessee_ssement, "ARG1", True, Fragment.PLUG)
    # plug ARG2 hole with pron_ssement, don't identify labels, assign the SSEMENT with the hole as the head
    # (i.e. the one with the noun at this point)
    poss_arg2_plugged = intersective(poss_arg1_plugged, possessor_ssement, "ARG2", False, Fragment.HOLE)

    # def_explicit_q with poss
    def_q = quantifier("def_explicit_q")
    final_poss = quant_phr(def_q, poss_arg2_plugged)

    return final_poss
```



Mid Level Rules

Liz Conrad

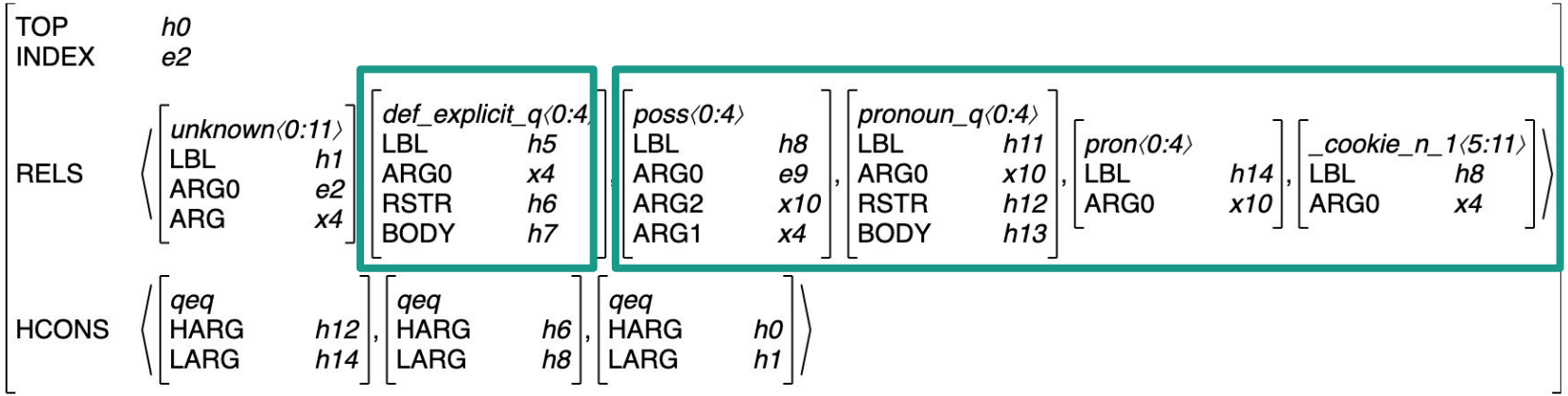
```
def poss_phr(possessor_ssement, possessee_ssement):
    # get poss EP
    poss = ep("poss", None, {"ARG0": "e", "ARG1": "u", "ARG2": "u"})
    # plug ARG1 hole with noun, identify labels, assign the noun (plug) as the head
    poss_arg1_plugged = intersective(poss, possessee_ssement, "ARG1", True, Fragment.PLUG)
    # plug ARG2 hole with pron_ssement, don't identify labels, assign the SSEMENT with the hole as the head
    # (i.e. the one with the noun at this point)
    poss_arg2_plugged = intersective(poss_arg1_plugged, possessor_ssement, "ARG2", False, Fragment.HOLE)

    # def_explicit_q with poss
    def_q = quantifier("def_explicit_q")
    final_poss = quant_phr(def_q, poss_arg2_plugged)

    return final_poss
```



Mid Level Rules

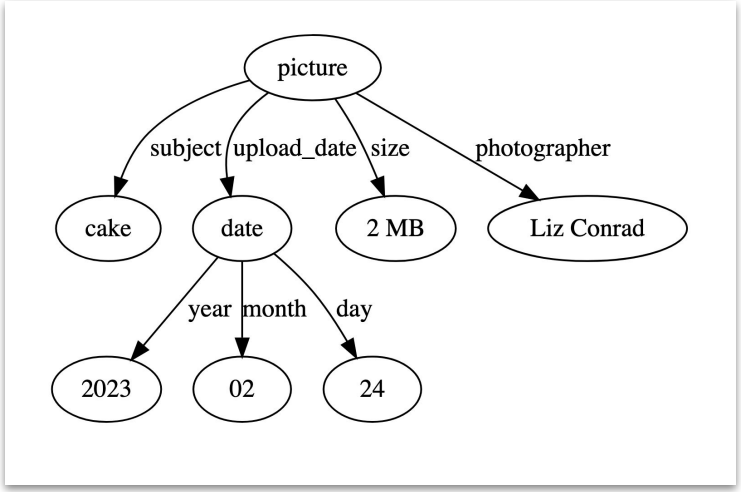
Liz Conrad

```
def poss_phr(possessor_ssement, possessee_ssement):
    # get poss EP
    poss = ep("poss", None, {"ARG0": "e", "ARG1": "u", "ARG2": "u"})
    # plug ARG1 hole with noun, identify labels, assign the noun (plug) as the head
    poss_arg1_plugged = intersective(poss, possessee_ssement, "ARG1", True, Fragment.PLUG)
    # plug ARG2 hole with pron_ssement, don't identify labels, assign the SSEMENT with the hole as the head
    # (i.e. the one with the noun at this point)
    poss_arg2_plugged = intersective(poss_arg1_plugged, possessor_ssement, "ARG2", False, Fragment.HOLE)

    # def_explicit_q with poss
    def_q = quantifier("def_explicit_q")
    final_poss = quant_phr(def_q, poss_arg2_plugged)

    return final_poss
```



High Level Rules

```
# DOMAIN RULES
# NODE RULES
Liz Conrad
def event_node(event_pred_label):
    return noun(event_pred_label, {'NUM': 'sg'})


Liz Conrad
def object_node(object_pred_label):
    return noun(object_pred_label)
```




```
# RELATIONAL RULES
Liz Conrad
def object_edge(event_ssement, object_ssement):
    of = ep("_of_p")
    return prepositional_phr(of, object_ssement, event_ssement)
```

Switching to WikiData

- Existing structured data
- Can obtain partial RDF dumps
- Choosing three related domains
  - Video games
  - Video game developers
  - Video game genres

Entity Example

The Sims 4 (Q12579896)

2014 video game
Sims 4 | TS4

edit

In more languages

Statements

instance of	video game 1 reference	edit + add value
part of	The Sims 4 + Star Wars: Journey to Batuu Bundle 0 references	edit + add reference + add value
logo image	 Logo of The Sims 4.svg 980 × 373; 14 KB 0 references	edit + add reference + add value

Property Example

instance of (P31)

that class of which this subject is a particular example and member; different from P279 (subclass of); for example: K2 is an instance of mountain; volcano is a subclass of mountain (and an instance of volcanic landform)

is a | is an | unique individual of | unitary element of class | rdf:type | type | ∈ | example of

▾ In more languages

Configure

Language	Label	Description	Also known as
English	instance of	that class of which this subject is a particular example and member; different from P279 (subclass of); for example: K2 is an instance of mountain; volcano is a subclass of mountain (and an instance of volcanic landform)	is a is an unique individual of unitary element of class rdf:type type ∈ example of

Aspirationally...

- By focusing on a subset of properties from each domain, will hopefully be able to generate referring expressions such as…
  - *The Sims 4*
  - *A life simulation game*
  - *A life simulation game developed by Maxis*
  - *A life simulation game released in 2014 by Maxis*
  - *The Sims 4, a life simulation game released in 2014 by Maxis*
  - …

QUESTIONS & DISCUSSION

1. What would be good terminology to use for the mid-level rules in particular?
2. What existing projects are there that involve building MRS fragments outside of a grammar?
3. What other domains/use cases can you imagine for this type of system?
4. How would you imagine evaluating this system?