A Uniform Meaning Representation for NLP Systems:

Lecture 5: Knowledge Grounding and Logical Inference with UMR

Martha Palmer and James Pustejovsky

Joint work with Jens Van Gysel, Meagan Vigus, Jin Zhao, Nianwen Xue, Jayeol Chun, Kenneth Lai, Sara Moeller, Jiarui Yao, Tim O'Gorman, Andrew Cowell, William Croft, Chu-Ren Huang, Jan Hajič, James Martin, Stephan Oepen, Rosa Vallejos, Jingxuan Tu, Kyeongmin Rim, Bingyang Ye, Susan Brown

ESSLLI 2023 Summer School

Ljubljana, Slovenia

August 7-12, 2023

1/42

Course Outline

- Monday: Formal Foundations of UMR and Extensions beyond AMR
- Tuesday: UMR Mechanisms for Quantification and Discourse Anaphora
- Wednesday: Annotation in UMR for Multiple Languages and Parsing UMRs
- Thursday: Extensions of UMR for Multimodal Communication and Situated Grounding
- Friday: UMR for Knowledge Grounding and Logical Inference

Mapping from VerbNet-GL to GLAMR

- VerbNet (VN) (Schuler, 2005; Brown et al., 2018,2022) as the primary lexical resource for identifying the canonical GL-event structure of each predicate

Subevent Structure in Generative Lexicon

Dynamic Event Structure

- Events are built up from multiple (stacked) layers of primitive constraints on the individual participants.

Dynamic Event Structure

- Events are built up from multiple (stacked) layers of primitive constraints on the individual participants.
- There may be many changes taking place within one atomic event, when viewed at the subatomic level.

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, *s*.

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:
  1. They can be ordered, $\alpha; \beta$ (α is followed by β);

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:
  1. They can be ordered, $\alpha; \beta$ (α is followed by β);
  2. They can be iterated, a^* (apply a zero or more times);

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, *s*.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:
  1. They can be ordered, $\alpha; \beta$ (α is followed by β);
  2. They can be iterated, a^* (apply *a* zero or more times);
  3. They can be disjoined, $\alpha \cup \beta$ (apply either α or β);

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:
  1. They can be ordered, $\alpha; \beta$ (α is followed by β);
  2. They can be iterated, a^* (apply a zero or more times);
  3. They can be disjoined, $\alpha \cup \beta$ (apply either α or β);
  4. They can be turned into formulas
     $[\alpha]\phi$ (after every execution of α, ϕ is true);
     $\langle\alpha\rangle\phi$ (there is an execution of α, such that ϕ is true);

Dynamic Interval Temporal Logic

(Pustejovsky and Moszkowicz, 2011)

- Formulas: ϕ propositions. Evaluated in a state, s.
- Programs: α, functions from states to states, $s \times s$. Evaluated over a pair of states, (s, s').
- Temporal Operators: $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$, $\phi\,\mathcal{U}\psi$.
- Program composition:
  1. They can be ordered, $\alpha; \beta$ (α is followed by β);
  2. They can be iterated, a^* (apply a zero or more times);
  3. They can be disjoined, $\alpha \cup \beta$ (apply either α or β);
  4. They can be turned into formulas
     $[\alpha]\phi$ (after every execution of α, ϕ is true);
     $\langle\alpha\rangle\phi$ (there is an execution of α, such that ϕ is true);
  5. Formulas can become programs, $\phi?$ (test to see if ϕ is true, and proceed if so).

Simple First-order Transition

$x := y$ (ν-transition)
"x assumes the value given to y in the next state."
$\langle \mathcal{M}, (i, i+1), (u, u[x/u(y)]) \rangle \models x := y$
iff $\langle \mathcal{M}, i, u \rangle \models s_1 \wedge \langle \mathcal{M}, i+1, u[x/u(y)] \rangle \models x = y$

Simple First-order Transition

$x := y$ (ν-transition)
"x assumes the value given to y in the next state."
$\langle \mathcal{M}, (i, i+1), (u, u[x/u(y)]) \rangle \models x := y$
iff $\langle \mathcal{M}, i, u \rangle \models s_1 \wedge \langle \mathcal{M}, i+1, u[x/u(y)] \rangle \models x = y$

$$\begin{tikzcd}
 & e^{[i,i+1]} \arrow[dl, dash] \arrow[dr, dash] & \\
e_1^{i} \arrow[rr, "x := y"] \arrow[d, dash] & & e_2^{i+1} \arrow[d, dash] \\
\mathcal{A}(z) = x & & \mathcal{A}(z) = y
\end{tikzcd}$$

Processes

With a ν-transition defined, a *process* can be viewed as simply an iteration of basic variable assignments and re-assignments:

Processes

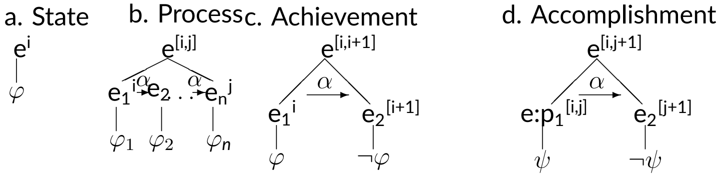
With a ν-transition defined, a *process* can be viewed as simply an iteration of basic variable assignments and re-assignments:

$$e$$

$$e_1 \xrightarrow{\nu} e_2 \ \ldots \xrightarrow{\nu} \ e_n$$

Event Structure with a dimension of scalar change

- Dynamic Event Structure (DES): ES enriched to track dynamically object attributes modified in the course of the event (Pustejovsky and Moszkowicz 2011).
- All the events are represented as a sequence of states related by functions (programs) which go from state to state.



Mapping from VerbNet-GL to GLAMR

- Each VN class is associated with multiple frames on the possible compositions of the verb sense (left)
- Each Frame shows its syntax and GL-event structure (right)

NP V NP.patient
NP V NP PP.instrument
NP.patient V
NP V ADV-Middle
NP.instrument V NP
NP V NP PP.result
NP V NP PP.result PP.instrument
NP V PP.result
NP V PP.result PP.instrument

EXAMPLE:

Jennifer baked the potatoes in the oven.

SHOW DEPENDENCY PARSE TREE

SYNTAX:

Agent VERB Patient { in on with } Instrument

SEMANTICS:

¬ COOKED(e1 , Patient , V_Final_State)
DO(e2 , Agent)
UTILIZE(e2 , Agent , Instrument)
APPLY_HEAT(ë3 , Instrument , Patient)
CAUSE(ë3 , e4)
COOKED(e4 , Patient , V_Final_State)

FORCE DYNAMICS:

None

Mapping from VerbNet-GL to GLAMR

- Each VN class is also associated with a group of lemmas

Member Verb Lemmas:

BAKE	BARBECUE	BARBEQUE	BLANCH	BOIL	BRAISE	BROIL	BROWN
CARAMELIZE	CHAR	CHARBROIL	CHARCOAL-BROIL	CODDLE	COOK	CRISP	
DEEP-FRY	FRENCH-FRY	FRY	GRILL	HARDBOIL	HEAT	MICROWAVE	
OVEN-FRY	OVEN-POACH	OVERBAKE	OVERCOOK	OVERHEAT	PAN-BROIL		
PAN-FRY	PARBOIL	PARCH	PERCOLATE	PERK	PICKLE	PLANK	POACH
POT-ROAST	REHEAT	RISSOLE	ROAST	SAUTE	SCALD	SCALLOP	SEAR
SHIRR	SIMMER	SOFTBOIL	STEAM	STEAM-BAKE	STEEP	STEW	
STIR-FRY	TOAST	WARM_UP					

Mapping from VerbNet-GL to GLAMR

- Given the GL-event structure from VN, for the predicate node `pred-01`, and the subevent `se(E1, ROLE1, ROLE2)`, we propose the general form for the GLAMR:
- ```
  (p / pred-01
       :event-structure (se / subevents
            :E1 (sub1 / subevent
                 :ROLE1 [value1]
                 :ROLE2 [value2]))
  ```
    - subevent index: `:Ex` (E1, E2, ...)
    - subevent names: `subevent`
    - subevent roles: `ROLE1`, `ROLE2`
  ```

Subevent Sequence in GLAMR

- GL-event structure of a predicate contains multiple subevent
- GLAMR graph only includes subevents that contain as least one `Patient` or `Theme` role (for tracking the change of the object)

    Example: Pour them into the bowl.

```
HAS_LOCATION(e1, Theme, Initial_Loc.)
HAS_LOCATION(e4, Theme, Destination)

(p / pour-01
  :ARG0 (y / you)
  :ARG1 (t / them)
  :ARG3 (b / bowl)
  :event-structure (se / subevents
      :E1 (se1 / has_location
        :THEME t
        :INITIAL_LOC N/A)
      :E4 (se4 / has_location
        :THEME t
        :DESTINATION b)
  :mode imperative)
```

Action Subevent in GLAMR

- Given the nature of the procedural texts in the data, GLAMR incorporates the `:ACTION` subevent into the predicate
- The subevent represents the action that has been performed on the objects during the event time

Example: Wash the cranberries.

```
(w / wash-01
  :ARG0 (y / you)
  :ARG1 (c / cranberries)
  :event-structure (se / subevents
      :E0 (se0 / do
        :ACTION (w1 / wash))
      [... ...]
  :mode imperative)
```

Negation in GLAMR

- GL-event structure use logical connective "¬" to represent the negation of the subevent state (`¬COOK` == "uncooked" or "not cooked")
- GLAMR uses the attribute `:polarity` to represent the negation

  Example: Wash the cranberries.

```
¬HAS_STATE(e1, Patient, V_Final_State)
HAS_STATE(e3, Patient, V_Final_State)

(w / wash-01
  :ARG0 (y / you)
  :ARG1 (c / cranberries)
  :event-structure (se / subevents
      :E1 (se1 / has_state
        :polarity -
        :PATIENT c
        :V_FINAL_STATE (w1 / washed))
      :E3 (se3 / has_state
        :PATIENT s4c
        :V_FINAL_STATE w1)
      [... ...]
  :mode imperative)
```

Simultaneous Subevents in GLAMR

- GL-event structure also contains subevents that happen simultaneously denoted by the same subevent index (e.g., two `E0`).
- Subevents with the same index in GLAMR are stacked with the `:op` roles

  Example: Roll the cranberries in the sugar.

```
MOTION(e2 ,Theme ,Trajectory)
¬HAS_LOCATION(e2 ,Theme ,Initial_Loc.)
HAS_LOCATION(e3, Theme, Destination)

(r / roll-01
  :ARG0 (y / you)
  :ARG1 (c / cranberries)
  :ARG2 (s / sugar)
  :event-structure (se / subevents
      :E2 (se2 / and
        :op1 (se21 / motion
          :THEME c
          :TRAJECTORY N/A)
        :op2 (se22 / has_location
          :polarity -
          :THEME c
          :INITIAL_LOC N/A))
      [... ...]
  :mode imperative)
```

Data Preparation

- Randomly sampled 15 recipes from the R2VQ corpus (Tu et al., 2022a).
- Use AMR parser to parse each recipe sentence into a PENMAN graph, which is then dually annotated and validated.
- Disagreements between annotators are adjudicated, and a finalized gold standard AMR annotation is created.

# of	**Count**
Documents	15
Sentences / PENMAN Graphs	137
Predicates	197

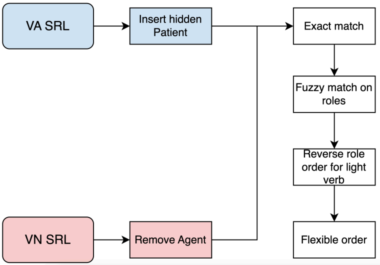
Table: Count of PENMAN graphs and predicates in the 15 recipes from our data.

Sense and Role Mapping

- The VerbAtlas verb sense (Di Fabio et al., 2019) from the SRL annotation is mapped to the PropBank (PB) sense using provided mapping files.
- Use mapping files in Semlink (Stowe et al., 2021) to connect PB verb senses to corresponding VerbNet classes (VN) and their subevent structures.
- VerbNet-GL (Brown et al., 2018, 2022) is used as the resource for the subevent structure in GLAMR.

Subevent Structure Identification

- Each VerbNet class can have multiple different frames.
- Leverage the SRLs in both VerbNet and R2VQ corpus to extract the correct VerbNet frame and its unique subevent structure.
- Built a pipeline with heuristics since the SRLs are not always perfect match.

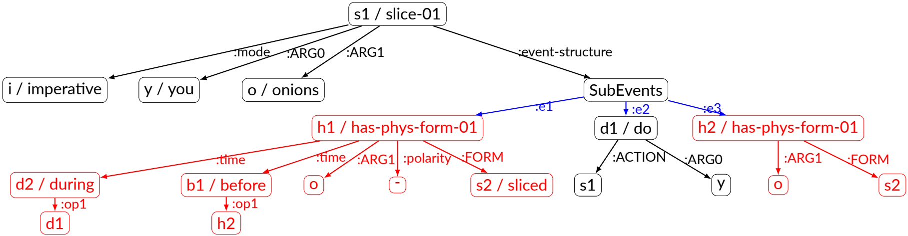


GLAMR Generation

- Generate final GLAMR graphs by integrating the fulfilled predicate subevent structure into the gold-standard AMR graphs using token alignment information.
- Subevent names often involve changes of locations and event states, and the most frequent subevent roles involve objects undergoing changes, locations, and tools used.
- GL-event structure is effective for enriching AMR with fine-grained subevent information in the cooking domain.

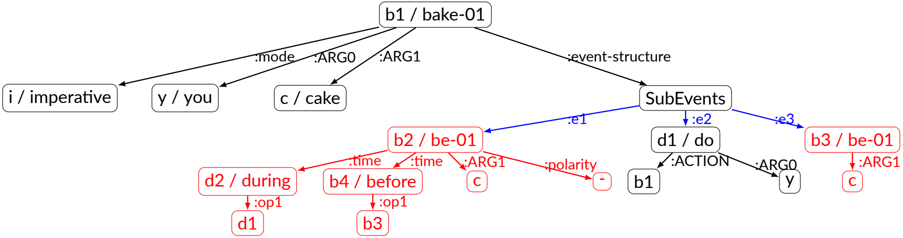
Subevent Names	**Count**	**Subevent Roles**	**Count**
has_location	102	Patient	262
motion	49	Theme	223
¬has_location	49	Initial_Loc.	94
cooked	30	V_Final_State	74
¬cooked	30	Trajectory	50
apply_heat	30	Destination	50
together	20	Instrument	36
has_state	19	V_State	30

GLAMR - Transformation



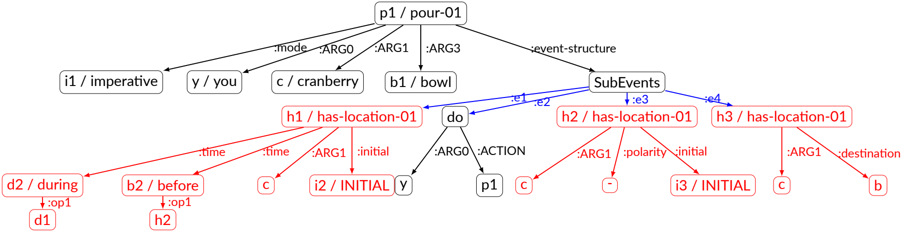
Slice the onions.

GLAMR - Creation



Bake the cake.

GLAMR - Change of Location

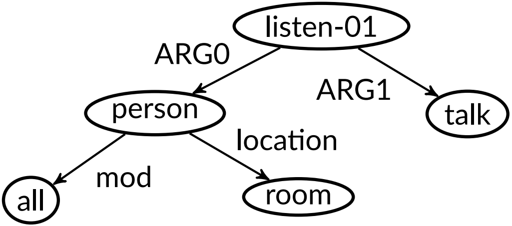


Pour the cranberries into a large bowl.

A Continuation Semantics for AMR

- AMR comes with a “logical form”

Everyone in the room listened to a talk.




```
(l / listen-01
      :ARG0 (p / person
              :mod (a / all)
              :location (r / room))
      :ARG1 (t / talk))
```

instance(l, listen-01)$\wedge$instance(p, person)$\wedge$instance(a, all)$\wedge$instance(r, room)$\wedge$ instance(t, talk) $\wedge$ ARG0(l, p) $\wedge$ mod(p, a) $\wedge$ location(p, r) $\wedge$ ARG1(l, t)

A Continuation Semantics for AMR

- However, this logical form cannot be used to draw inferences
    - Quantifiers and negation treated as ordinary modifiers
        - Infer “a dog barked” from “a dog didn’t bark”
    - No representation of quantifier scope

A Continuation Semantics for AMR

- Idea: use continuations to translate AMR to first-order logic
- The continuation hypothesis: meanings of expressions are functions of their contexts (Barker, 2002; Barker and Shan, 2014)
    - Continuations: set of contexts that, when combined with an expression, result in a true sentence

Continuations

The continuation hypothesis: meanings of expressions are functions of their contexts (Barker, 2002; Barker and Shan, 2014)

Continuations

The continuation hypothesis: meanings of expressions are functions of their contexts (Barker, 2002; Barker and Shan, 2014)

- Generalized quantifiers: set of properties true of an entity

Continuations

The continuation hypothesis: meanings of expressions are functions of their contexts (Barker, 2002; Barker and Shan, 2014)

- Generalized quantifiers: set of properties true of an entity
- Continuations: set of contexts that, when combined with an expression, result in a true sentence

Continuations

What is the meaning of

```
(b / bark-01
   :ARG0 (d / dog)) ?
```

Continuations

What is the meaning of
```
(b / bark-01
   :ARG0 (d / dog))
```
?

- Suppose we know the meaning of `(d / dog)`; what is its continuation?

Continuations

What is the meaning of
```
(b / bark-01
   :ARG0 (d / dog)) ?
```

- Suppose we know the meaning of `(d / dog)`; what is its continuation?
  - We know it's the `ARG0` of something, so let's say $\lambda n.\text{ARG0}(m, n)$

Continuations

What is the meaning of

```
(b / bark-01
   :ARG0 (d / dog)) ?
```

- Suppose we know the meaning of `(d / dog)`; what is its continuation?
  - We know it's the `ARG0` of something, so let's say $\lambda n.\mathrm{ARG0}(m, n)$
- What about the continuation of `(b / bark-01)`?

Continuations

What is the meaning of
```
(b / bark-01
   :ARG0 (d / dog))
```
?

- Suppose we know the meaning of `(d / dog)`; what is its continuation?
  - We know it's the `ARG0` of something, so let's say $\lambda n.\mathrm{ARG0}(m, n)$
- What about the continuation of `(b / bark-01)`?
  - Two parts: the dog $[\![$`(d / dog)`$]\!](\lambda n.\mathrm{ARG0}(m, n))$, and ϕ, the rest of the sentence (if any)

Continuations

What is the meaning of
```
(b / bark-01
   :ARG0 (d / dog))
```
?

$$\lambda\phi.[\![\texttt{(b / bark-01)}]\!](\lambda m.[\![\texttt{(d / dog)}]\!](\lambda n.\mathrm{ARG0}(m,n)) \wedge \phi(m))$$

Continuations

What is the meaning of

```
(b / bark-01
   :ARG0 (d / dog)) ?
```

$$\lambda\phi.[\![\texttt{(b / bark-01)}]\!](\lambda m.[\![\texttt{(d / dog)}]\!](\lambda n.\mathrm{ARG0}(m,n)) \wedge \phi(m))$$

- In AMR, all variables are assumed to be existentially quantified

Continuations

What is the meaning of `(b / bark-01 :ARG0 (d / dog))` ?

$$\lambda\phi.[\![\texttt{(b / bark-01)}]\!](\lambda m.[\![\texttt{(d / dog)}]\!](\lambda n.\mathrm{ARG0}(m,n)) \wedge \phi(m))$$

- In AMR, all variables are assumed to be existentially quantified
  - $[\![\texttt{(b / bark-01)}]\!] = \lambda\psi.\exists b.\mathrm{bark\text{-}01}(b) \wedge \psi(b)$ ("a bark")
  - $[\![\texttt{(d / dog)}]\!] = \lambda\chi.\exists d.\mathrm{dog}(d) \wedge \chi(d)$ ("a dog")

Continuations

What is the meaning of `(b / bark-01 :ARG0 (d / dog))` ?

$$\lambda\phi.[\![\texttt{(b / bark-01)}]\!](\lambda m.[\![\texttt{(d / dog)}]\!](\lambda n.\mathrm{ARG0}(m,n)) \wedge \phi(m))$$

- In AMR, all variables are assumed to be existentially quantified
  - $[\![\texttt{(b / bark-01)}]\!] = \lambda\psi.\exists b.\text{bark-01}(b) \wedge \psi(b)$ ("a bark")
  - $[\![\texttt{(d / dog)}]\!] = \lambda\chi.\exists d.\text{dog}(d) \wedge \chi(d)$ ("a dog")

Using $\lambda x.\top$ (the trivial continuation) for ϕ:

Continuations

What is the meaning of
```
(b / bark-01
   :ARG0 (d / dog))
```
?

$\lambda\phi.[\![\texttt{(b / bark-01)}]\!](\lambda m.[\![\texttt{(d / dog)}]\!](\lambda n.\mathrm{ARG0}(m,n)) \wedge \phi(m))$

- In AMR, all variables are assumed to be existentially quantified
  - $[\![\texttt{(b / bark-01)}]\!] = \lambda\psi.\exists b.\mathrm{bark\text{-}01}(b) \wedge \psi(b)$ ("a bark")
  - $[\![\texttt{(d / dog)}]\!] = \lambda\chi.\exists d.\mathrm{dog}(d) \wedge \chi(d)$ ("a dog")

Using $\lambda x.\top$ (the trivial continuation) for ϕ:

$\exists b.\mathrm{bark\text{-}01}(b) \wedge \exists d.\mathrm{dog}(d) \wedge \mathrm{ARG0}(b,d)$

Continuations and Scope

To reverse the scope, reverse the order of application:

Continuations and Scope

To reverse the scope, reverse the order of application:

$\lambda\phi.[\![\texttt{(d \textbackslash{} dog)}]\!](\lambda n.[\![\texttt{(b / bark-01)}]\!](\lambda m.\mathrm{ARG0}(m, n) \wedge \phi(m)))$

Continuations and Scope

To reverse the scope, reverse the order of application:

$\lambda\phi.[\![\texttt{(d \textbackslash\ dog)}]\!](\lambda n.[\![\texttt{(b / bark-01)}]\!](\lambda m.\mathsf{ARG0}(m, n) \wedge \phi(m)))$

$\rightsquigarrow \exists d.\mathsf{dog}(d) \wedge \exists b.\mathsf{bark\text{-}01}(b) \wedge \mathsf{ARG0}(b, d)$

Continuations and Scope

To reverse the scope, reverse the order of application:

$\lambda\phi.[\![\texttt{(d \textbackslash\ dog)}]\!](\lambda n.[\![\texttt{(b / bark-01)}]\!](\lambda m.\mathrm{ARG0}(m,n) \wedge \phi(m)))$

$\rightsquigarrow \exists d.\mathrm{dog}(d) \wedge \exists b.\mathrm{bark\text{-}01}(b) \wedge \mathrm{ARG0}(b,d)$

- Methods of marking scope:
  - "\" for projection/wide scope (Bos, 2016)
  - Scope node (Pustejovsky et al., 2019)
  - ...

Detailed Derivation

$$[\![c]\!] = \lambda\phi.\phi(c)$$
$$[\![x]\!] = \lambda\phi.\phi(x)$$
$$[\![(x|P :R_1(y\backslash Q\ldots) :R_iA_i)]\!] = \lambda\phi.[\![(y\backslash Q\ldots)]\!](\lambda n.[\![(x|P :R_iA_i)]\!](\lambda m.R_1(m,n)\wedge\phi(m)))$$
$$[\![(x|P :R_1A_1 :R_iA_i)]\!] = \lambda\phi.[\![(x|P :R_iA_i)]\!](\lambda m.[\![A_1]\!](\lambda n.R_1(m,n))\wedge\phi(m))$$
$$[\![(x|P)]\!] = \lambda\phi.\exists x.P(x)\wedge\phi(x)$$

$$\begin{aligned}
&[\![\text{a dog scratched itself}]\!]\\
&= \lambda\phi.[\![\text{a dog}]\!](\lambda n.[\![\text{scratched itself}]\!](\lambda m.\text{ARG0}(m,n)\wedge\phi(m)))\\
&= \lambda\phi.[\![\text{a dog}]\!](\lambda n.(\lambda\psi.\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\psi(s))(\lambda m.\text{ARG0}(m,n)\wedge\phi(m)))\\
&= \lambda\phi.[\![\text{a dog}]\!](\lambda n.\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge(\lambda m.\text{ARG0}(m,n)\wedge\phi(m))(s))\\
&= \lambda\phi.[\![\text{a dog}]\!](\lambda n.\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\text{ARG0}(s,n)\wedge\phi(s))\\
&= \lambda\phi.(\lambda\psi.\exists d.\text{dog}(d)\wedge\psi(d))(\lambda n.\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\text{ARG0}(s,n)\wedge\phi(s))\\
&= \lambda\phi.\exists d.\text{dog}(d)\wedge(\lambda n.\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\text{ARG0}(s,n)\wedge\phi(s))(d)\\
&= \lambda\phi.\exists d.\text{dog}(d)\wedge\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\text{ARG0}(s,d)\wedge\phi(s)\\
&\rightsquigarrow \exists d.\text{dog}(d)\wedge\exists s.\text{scratch-01}(s)\wedge\text{ARG1}(s,d)\wedge\text{ARG0}(s,d)
\end{aligned}$$

Universal Quantification

- In AMR, all variables are assumed to be existentially quantified
    - $[\![\texttt{(d / dog)}]\!] = \lambda\phi.\exists d.\mathrm{dog}(d) \wedge \phi(d)$

Universal Quantification

- In AMR, all variables are assumed to be existentially quantified
  - $[\![\texttt{(d / dog)}]\!] = \lambda\phi.\exists d.\mathrm{dog}(d) \wedge \phi(d)$
- Note that $\forall x.\phi(x) \rightarrow \psi(x) \equiv \neg\exists x.\phi(x) \wedge \neg\psi(x)$
  - To get a universal effect, negate the expression, negate the continuation

Universal Quantification

- In AMR, all variables are assumed to be existentially quantified
  - $[\![\texttt{(d / dog)}]\!] = \lambda\phi.\exists d.\mathrm{dog}(d) \wedge \phi(d)$
- Note that $\forall x.\phi(x) \rightarrow \psi(x) \equiv \neg\exists x.\phi(x) \wedge \neg\psi(x)$
  - To get a universal effect, negate the expression, negate the continuation

$$[\![\texttt{(d / dog :quant (e / every))}]\!] = \lambda\phi.\neg[\![\texttt{(d / dog)}]\!](\lambda m.\neg\phi(m))$$

Universal Quantification

- Depending on when the universal quantification rule is applied, one can control what is interpreted in the restriction vs. the nuclear scope

Universal Quantification

- Depending on when the universal quantification rule is applied, one can control what is interpreted in the restriction vs. the nuclear scope
    - Adjectives

Universal Quantification

- Depending on when the universal quantification rule is applied, one can control what is interpreted in the restriction vs. the nuclear scope
  - Adjectives
    - ```
      (s / scratch-01
          :ARG0 (d \ dog
                  :quant (e / every)
                  :mod (b / brown))
          :ARG1 d)
      ```
    - “Every brown dog scratched itself” vs. “Every dog is brown and scratched itself”
      ```

Universal Quantification

- Depending on when the universal quantification rule is applied, one can control what is interpreted in the restriction vs. the nuclear scope
  - Adjectives
    - ```
      (s / scratch-01
          :ARG0 (d \ dog
                    :quant (e / every)
                    :mod (b / brown))
          :ARG1 d)
      ```
    - “Every brown dog scratched itself” vs. “Every dog is brown and scratched itself”
  - Relative clauses
      ```

AMR for Donkey Sentences

Every farmer who owns a donkey loves it.

AMR for Donkey Sentences

Every farmer who owns a donkey loves it.

```
(l / love-01
      :ARG0 (f \ farmer
                    :quant (e / every)
                    :ARG0-of (o \ own-01
                                  :ARG1 (d \ donkey)))
      :ARG1 d)
```

AMR for Donkey Sentences

Every farmer who owns a donkey loves it.

```
(l / love-01
     :ARG0 (f \ farmer
                  :quant (e / every)
                  :ARG0-of (o \ own-01
                                 :ARG1 (d \ donkey)))
     :ARG1 d)
```

$[\![\text{every farmer who owns a donkey loves it}]\!]$

$= \lambda\phi.[\![\text{every farmer who owns a donkey}]\!](\lambda n.[\![\text{loves it}]\!](\lambda m.\text{ARG0}(m, n) \wedge \phi(m)))$

$\ldots$

$\rightsquigarrow \forall f.\forall d.\forall o.(\text{farmer}(f) \wedge \text{donkey}(d) \wedge \text{own-01}(o) \wedge \text{ARG0-of}(f, o) \wedge \text{ARG1}(o, d))$

$\quad\quad \to \exists l.\text{love-01}(l) \wedge \text{ARG1}(l, d) \wedge \text{ARG0}(l, f)$

AMR for Donkey Sentences

Every farmer who owns a donkey loves it.

```
(l / love-01
      :ARG0 (f \ farmer
                    :quant (e / every)
                    :ARG0-of (o \ own-01
                                  :ARG1 (d \ donkey)))
      :ARG1 d)
```

$[\![\text{every farmer who owns a donkey loves it}]\!]$

$= \lambda\phi.[\![\text{every farmer who owns a donkey}]\!](\lambda n.[\![\text{loves it}]\!](\lambda m.\text{ARG0}(m, n) \wedge \phi(m)))$

$\ldots$

$\rightsquigarrow \forall f.\forall d.\forall o.(\text{farmer}(f) \wedge \text{donkey}(d) \wedge \text{own-01}(o) \wedge \text{ARG0-of}(f, o) \wedge \text{ARG1}(o, d))$

$\quad\quad \rightarrow \exists l.\text{love-01}(l) \wedge \text{ARG1}(l, d) \wedge \text{ARG0}(l, f)$

See paper for more details!

Negation

- Two types of negation

Negation

- Two types of negation
  - Wide-scope (attaches to quantifier)
    - $\neg\forall d.\text{dog}(d) \rightarrow \exists m.\text{meow-01}(m) \wedge \text{ARG0}(m, d)$
      "Not every dog meowed."

Negation

- Two types of negation
  - Wide-scope (attaches to quantifier)
    - $\neg\forall d.\text{dog}(d) \rightarrow \exists m.\text{meow-01}(m) \wedge \text{ARG0}(m, d)$
      "Not every dog meowed."
  - Narrow-scope (attaches to predicate)
    - $\forall d.\neg\text{dog}(d) \rightarrow \exists m.\text{meow-01}(m) \wedge \text{ARG0}(m, d)$
      "Every non-dog meowed."

Negation

- Adopting (as one option) the backslash (wide-scope)/forward slash (narrow-scope) syntax:

Negation

- Adopting (as one option) the backslash (wide-scope)/forward slash (narrow-scope) syntax:
  - Wide-scope ($\neg\forall d.\text{dog}(d) \rightarrow \ldots$)
    - $[\![\texttt{(d / dog :polarity (n \textbackslash\ -))}]\!] = \lambda\phi.\neg[\![\texttt{(d / dog)}]\!](\phi)$

Negation

- Adopting (as one option) the backslash (wide-scope)/forward slash (narrow-scope) syntax:
  - Wide-scope ($\neg\forall d.\mathrm{dog}(d) \to \ldots$)
    - $[\![$`(d / dog :polarity (n \ -))`$]\!] = \lambda\phi.\neg[\![$`(d / dog)`$]\!](\phi)$
  - Narrow-scope ($\forall d.\neg\mathrm{dog}(d) \to \ldots$)
    - $[\![$`(d / dog :polarity (n / -))`$]\!] = \lambda\phi.[\![$`(d / ¬dog)`$]\!](\phi)$

Negation

- Adopting (as one option) the backslash (wide-scope)/forward slash (narrow-scope) syntax:
  - Wide-scope ($\neg\forall d.\text{dog}(d) \rightarrow \ldots$)
    - $[\![$`(d / dog :polarity (n \ -))`$]\!] = \lambda\phi.\neg[\![$`(d / dog)`$]\!](\phi)$
  - Narrow-scope ($\forall d.\neg\text{dog}(d) \rightarrow \ldots$)
    - $[\![$`(d / dog :polarity (n / -))`$]\!] = \lambda\phi.[\![$`(d /` $\neg$`dog)`$]\!](\phi)$

Again, see paper for more details!

Discussion

- Maintain the predicative core

Discussion

- Maintain the predicative core
- Avoids the pitfalls of underspecification

Discussion

- Maintain the predicative core
- Avoids the pitfalls of underspecification
- Allows us to utilize existing AMR corpora

Discussion

- Order of application is crucial

Discussion

- Order of application is crucial
- Will this increase annotators' cognitive load?

Discussion

- Order of application is crucial
- Will this increase annotators' cognitive load?
  - Specify default interpretations where possible (perhaps automatically/behind the scenes)

Discussion

- Order of application is crucial
- Will this increase annotators' cognitive load?
  - Specify default interpretations where possible (perhaps automatically/behind the scenes)
    - Arguments of an event should be projective in general
    - Negation should take wide scope over event quantifiers by default

Future Work

- Extending the translation to other semantic phenomena (e.g., Uniform Meaning Representation)
    - Tense
    - Aspect
    - Modality
    - ...

Future Work

- Extending the translation to other semantic phenomena (e.g., Uniform Meaning Representation)
  - Tense
  - Aspect
  - Modality
  - ...
- Dynamic semantics
  - "Discourse moves as continuations"

UMR summary

- UMR is a rooted directed node-labeled and edge-labeled document-level graph.
- UMR is a document-level meaning representation that builds on sentence-level meaning representations
- UMR aims to achieve semantic stability across syntactic variations and support logical inference
- UMR is a cross-lingual meaning representation that separates language-general aspects of meaning from those that are language-specific
- We are testing UMR English, Chinese, Arabic, Arapaho, Kukama, Sanapana, Navajo

References

Pustejovsky, J. and Moszkowicz, J. L. (2011). The qualitative spatial dynamics of motion in language.
Spatial Cognition & Computation, 11(1):15–44.

Acknowledgements

This work is supported in part by a grant from the IIS Division of National Science Foundation (Awards No. 1763926, 1764048, 1764091) entitled Building a Uniform Meaning Representation for Natural Language Processing awarded to Nianwen Xue, James Pustejovsky, Martha Palmer and William Croft. All views expressed in this paper are those of the authors and do not necessarily represent the view of the National Science Foundation.