Abstract Meaning Representation (AMR) 1.2 Specification

May 14, 2014

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, Nathan Schneider

Table of Contents *generated with DocToc*

Part I. Introduction

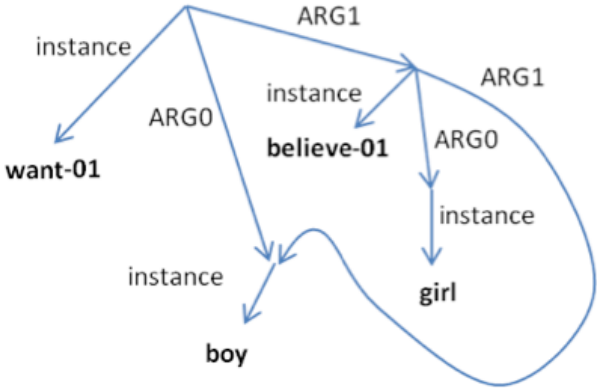
AMR captures “who is doing what to whom” in a sentence. Each sentence is represented as a **rooted, directed, acyclic graph** with labels on edges (relations) and leaves (concepts).

Like a parse tree, AMR provides a single, traversable structure that takes all words into account. It is not a disconnected set of annotation layers. Unlike a parse tree, the AMR is abstract. It may represent any number of natural language sentences. AMR does not annotate the individual words in a sentence, like a dependency parse does.

AMR implements a simplified, standard neo-Davidsonian semantics [Davidson 1967, Higginbotham 1985], using standard feature structure representation [Shieber 1986, Carpenter 1992]. AMR’s formal origins are in unification systems [Kay 1979, Knight 1989, Moore 1989] and natural language generation [Mann 1982, Elhadad 1988, Knight & Hatzivassiloglou 1995]. Predicates senses and core semantic roles in AMR are drawn from the **amazing OntoNotes project**.

AMR does not say anything about how it wants to be processed. It is closer to English than to other languages. It is not an interlingua.

Example



This AMR means (roughly): There is a wanting event, whose ARG0 (wanter) is a boy, and whose ARG1 (wanted thing) is a believing event. This believing event has an ARG0 (believer), which is a girl, and it has an ARG1 (believed thing), which is the **same boy** just mentioned. Here, `boy` plays two roles: (1) it is the ARG0 of `want-01`, and (2) it is the ARG1 of `believe-01`. The AMR captures this with two directed edges pointing to the same node. (Per OntoNotes, predicate senses are marked with suffixes like `-01` and `-02`, while ARG0, ARG1, etc., denote core, predicate-specific roles.)

Here is a text-friendly way to write the same AMR:

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (b2 / believe-01
             :ARG0 (g / girl)
             :ARG1 b))
```

The variables `w`, `b`, `b2`, and `g` correspond to internal nodes in the graph above. Note that b appears twice in this format, the first time as `(b / boy)` and the second time simply as `b`.

This AMR can also be viewed as conjunction of logical triples, omitting root information:

```
instance(w, want-01) ^      /* w is an instance of wanting */
instance(b, boy) ^          /* b is an instance of boy */
instance(b2, believe-01) ^  /* b2 is an instance of believing */
instance(g, girl) ^         /* g is an instance of girl */
ARG0(w, b) ^                /* b is the wanter in w */
ARG1(w, b2) ^               /* b2 is the wantee in w */
ARG0(b2, g) ^               /* g is the believer in b2 */
ARG1(b2, b)                 /* b is the believee in b2 */
```

Abstraction away from English

The AMR above can be expressed variously in English:

> The boy wants the girl to believe him.
>
> The boy wants to be believed by the girl.
>
> The boy has a desire to be believed by the girl.
>
> The boy's desire is for the girl to believe him.
>
> The boy is desirous of the girl believing him.

etc.

The concept `want-01` might be realized as a verb (“wants”), a noun (“desire”), or an adjective (“desirous”).

We think of AMR leaf-labels as concepts rather than words. We do not point to an element in an AMR and say “that is a noun” or “that is a verb”. Rather, we say “that is an object” or “that is an event”.

A single entity (“boy”) can play multiple roles simultaneously (e.g., “ARG0” of `want-01`, and “ARG1” of `believe-01`). The AMR does not talk about pronouns or zero-pronouns, though these are natural mechanisms for expressing multiple roles in English.

In many cases, English function words do not show up at all in AMR:

```
(a / adjust-01
   :ARG0 (b / girl)
   :ARG1 (m / machine))
```

> The girl **made** adjustments **to** the machine.
>
> The girl adjusted the machine.
>
> The machine **was** adjusted **by** the girl.

```
(k / kill-01
   :time (y / yesterday))
```

> The killing **happened** yesterday.
>
> The killing **took place** yesterday.

```
(a / and
   :op1 (b / boy)
   :op2 (g / girl))
```

> the boy and the girl
>
> **both** the boy and the girl

```
(b / boat
   :poss (h / he))
```

> his boat
>
> his **own** boat

More Logical than Syntax

AMR strives for a more logical, less syntactic representation. For example, “the boy must not go” and “the boy may not go” are syntactically similar, but the placement of negation (`:polarity -`) is very different in the two AMRs:

```
(o / obligate-01
   :ARG2 (g / go-02
            :ARG0 (b / boy)
            :polarity -))
```

> The boy must not go.
>
> It is obligatory that the boy not go.

```
(p / permit-01
   :ARG1 (g / go-02
            :ARG0 (b / boy))
   :polarity -)
```

> The boy may not go.
>
> The boy is not permitted to go.
>
> It is not permissible for the boy to go.
>
> The boy does not have permission to go.

The AMR transparently represents what is being negated. Note that the concept `permit-01` can be realized as a modal, a participle, or a noun.

Focus

The root of an AMR binds its contents into a single, traversable directed graph. It also serves as a rudimentary representation of overall focus. So we have:

```
(w / white
   :domain (m / marble))
```

> The marble is white.
>
> the whiteness of the marble

```
(m / marble
   :mod (w / white))
```

> the white marble
>
> the marble that is white

```
(s / see-01
   :ARG0 (b / boy)
   :ARG1 (w / white
            :domain (m / marble)))
```

> The boy sees that the marble is white.
>
> The boy sees the whiteness of the marble.

```
(s / see-01
   :ARG0 (b / boy)
   :ARG1 (m / marble
            :mod (w / white)))
```

> The boy sees the white marble.
>
> The boy sees the marble that is white.

We can write `:domain-of` as an inverse of `:domain`, but we often shorten this to `:mod`.

Inverse roles are useful for maintaining a single rooted structure, e.g.:

```
(s / see-01
   :ARG0 (b / boy)
   :ARG1 (g / girl
            :ARG0-of (w / want-01
                        :ARG1 b)))
```

> The boy saw the girl who wanted him.
>
> The boy saw the girl who he was wanted by.
>
> The girl who wanted the boy was seen by him.

In this AMR, the role `:ARG0-of` connects `girl` with `want-01` in a natural way.

To re-focus an AMR, we can "lift up" any node to the root, and then imagine all other nodes falling down. For example, if we lift up the `w` node above, we get the same content, but rearranged:

```
(w / want-01
   :ARG0 (g / girl
            :ARG1-of (s / see-01
                        :ARG0 (b / boy)))
   :ARG1 b)
```

> The girl who was seen by the boy wants him.
>
> The boy is wanted by the girl he saw.

This is a matter of focusing: the first AMR (rooted by `see-01`) is about the seeing, while the second AMR (rooted by `want-01`) is about the wanting.

Another example of an inverse role (`:instrument-of`):

```
(c / change-01
   :ARG1 (d / document
            :instrument-of (r / regulate-01)))
```

> The regulatory documents were changed.

AMR slogans

Here are some slogans that make it easier to work with AMR:

- AMR captures the rough meaning of a sentence in a single, traversable directed acyclic graph.
- AMR does not say anything about how it wants to be processed.
- AMR is not an interlingua.
- There are no nouns and verbs in AMR.
- When we write AMR by hand, constituent trees and dependency trees are generally not needed.

- We cannot read off a unique English sentence from an AMR.

An AMR is like a foreign-language translation. Someone who creates an AMR from English may not provide links between AMR concepts and English word tokens. But it is reasonable to think about doing such an alignment later (perhaps automatically), just as it is reasonable to align tokens in bilingual texts.

Limitations of AMR 1.2

AMR 1.2 is over-simple in many ways:

- It is geared toward English and the vocabulary of English.
- It does not represent quantifier scope, or even universal quantification at all.
- It does not represent co-references that cross sentence boundaries.
- It drops grammatical number, tense, aspect, quotation marks, etc.
- It does not deeply capture many noun-noun or noun-adjective relations.
- It does not include deep frames such as Earthquake (with roles for magnitude, epicenter, casualties, etc) or Pregnancy (with roles for mother, father, baby gender, time since inception, etc). AMR 1.2 looks forward to AMR 2.0!

Part II. Concepts and relations

Concepts are tokens that appear at the leaves of AMR graphs. AMR does not formally carve the world up into events, objects, features, etc., though we may refer to a certain concept instance as an event. The following AMR has three concepts (boy, want, believe):

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (b2 / believe-01
             :ARG0 b))
```

> The boy wants to believe.

The slash (`/`) is shorthand for the `:instance` relation. This relation shows up more clearly in AMR graph format (see Introduction).

Concepts in AMR 1.2 are usually written with English words or phrases. Concepts with core semantic relations may have sense tags, to identify the semantic frame:

```
(b / believe-01
   :ARG0 (b / boy))
```

> The boy believes.

AMR semantic relations are best described through examples; see the next section (Phenomena). Here, we only provide a brief summary listing.

Core `:ARGx` roles. AMR 1.2 uses numbered `:ARGx` from OntoNotes:

```
:ARG0, :ARG1, :ARG2, :ARG3, :ARG4, :ARG5
```

Non-core roles:

```
:accompanier, :age
:beneficiary
:compared-to, :concession, :condition, :consist-of
:degree, :destination, :direction, :domain, :duration
:example, :extent
:frequency
:instrument
:location
:manner, :medium, :mod, :mode
:name
:ord
:part, :path, :polarity, :poss, :purpose
:quant
:scale, :source, :subevent
:time, :topic, :unit
:value
```

Roles used in date-entity:

```
:calendar, :century, :day, :dayperiod, :decade, :era, :month, :quarter, :season, :timezone, :weekday, :year, :year2
```

Roles of the form `:opx` are used in conjunctions, and in certain types of locations and times:

```
:op1, :op2, :op3, :op4, …
```

Roles of the form `:prep-X` are used in cases where there is no good relation from the list above. AMR likes to avoid these. Here is a partial list. It is only partial, as other `:prep-X` relations are legal AMR.

```
:prep-against, :prep-along-with, :prep-amid, :prep-among, :prep-as, :prep-at
:prep-by
:prep-concerning, :prep-considering
:prep-despite
:prep-except, :prep-excluding
:prep-following, :prep-for, :prep-from
:prep-in, :prep-in-addition-to, :prep-in-spite-of, :prep-into
:prep-like
:prep-on, :prep-on-behalf-of, :prep-opposite
:prep-per
:prep-regarding
:prep-save, :prep-such-as
:prep-through, :prep-to, :prep-toward
:prep-under, :prep-unlike
:prep-versus
:prep-with, :prep-within, :prep-without
```

etc.

Some conjunctions are also not well-covered under the list of non-core roles. AMR also likes to avoid these, but sometimes we have no good alternative:

```
:conj-as-if
```

etc.

All relations above have inverses of the form `:X-of`.

```
:ARG0-of, :ARG1-of
:location-of
```

etc.

Part III. Phenomena

Core roles

Core roles are taken from the OntoNotes semantic role layer. OntoNotes predicates are sense-labeled words (e.g., `sentence-01`). They are predicate-specific and numbered. For example:

- `:ARG0` of `charge-01` is the person doing the charging.
- `:ARG1` of `charge-01` is the person being charged.
- `:ARG2` of `charge-01` is the role or crime (for which `:ARG0` is charging `:ARG1`).

- `:ARG0` of `sentence-01` is the person doing the sentencing.
- `:ARG1` of `sentence-01` is the person being sentenced.
- `:ARG2` of `sentence-01` is the role or crime.

- `:ARG0` of `fine-01` is the person doing the fining.
- `:ARG1` of `fine-01` is the amount of the fine (e.g., $1000).
- `:ARG2` of `fine-01` is the person being fined.
- `:ARG3` of `fine-01` is the role or crime.

Note the predicate-sensitivity: a person might naturally be the `:ARG1` of `sentence-01`, but simultaneously the `:ARG2` of `fine-01`.

One semantic frame may be realized in vastly different ways in English:

```
(d / describe-01
   :ARG0 (h / he)
   :ARG1 (m / mission)
   :ARG2 (f / failure))
```

> He described the mission as a failure.
>
> As he described it, the mission was a failure.
>
> His description of the mission: failure.

Here, the AMR does not worry about representing the words “as”, “it”, or “was”.

Note that OntoNotes documentation for predicates and roles is often loose or humorous. For example, the `:ARG0` and `:ARG1` of `research-01` are nicknamed “student” and “subject”, but this does not mean that `research-01` is restricted to situations where the `:ARG0` is literally a student.

If OntoNotes is missing a predicate, AMR accepts `-00`:

```
(c / comeback-00
   :ARG0 (b / band))
```

> the band experienced a comeback

Modality

AMR represents syntactic modals with concepts like `possible`, `likely`, `obligate-01`, `permit-01`, `recommend-01`, `prefer-01`, etc.:

```
(p / possible
   :domain (g / go-02
              :ARG0 (b / boy)))
```

> The boy can go.
>
> It is possible that the boy goes.

```
(o / obligate-01
   :ARG2 (g / go-02
            :ARG0 (b / boy)))
```

> The boy must go.
>
> The boy is obligated to go.
>
> It is obligatory that the boy go.

```
(o / permit-01
   :ARG1 (g / go-02
            :ARG0 (b / boy)))
```

> The boy may go.
>
> The boy is permitted to go.
>
> It is permissible that the boy go.

```
(p / possible
   :domain (r / rain-01))
```

> It may rain.
>
> It might rain.
>
> Rain is possible.
>
> It's possible that it will rain.

```
(r / recommend-01
   :ARG1 (g / go-02
            :ARG0 (b / boy)))
```

> The boy should go.
>
> It is recommended that the boy go.

```
(l / likely
   :domain (g / go-02
```

```
            :ARG0 (b / boy)))
```

> The boy is likely to go.
>
> It is likely that the boy will go.

AMR ignores the modal “would”, except in cases like:

```
(p / prefer-01
   :ARG0 (b / boy)
   :ARG1 (g / go-02
            :ARG0 b))
```

> The boy would rather go.
>
> The boy prefers to go.

Another example:

```
(u / use-03
   :ARG0 (i / i)
   :ARG1 (w / work-01
            :ARG0 i))
```

> I am used to working.

Negation

AMR represents negation logically, using `:polarity`.

```
(g / go-02
   :ARG0 (b / boy)
   :polarity -)
```

> The boy doesn’t go.

```
(p / possible
   :domain  (g / go-02
               :ARG0 (b / boy))
   :polarity -)
```

> The boy can’t go.
>
> It’s not possible for the boy to go.

```
 (p / possible
    :domain (g / go-02
               :ARG0 (b / boy)
               :polarity -))
```

> It is possible for the boy not to go.
>
> It is possible for the boy to not go.

```
(p / obligate-01
   :ARG2 (g / go-02
            :ARG0 (b / boy))
   :polarity -)
```

> The boy doesn't have to go.
>
> It's not necessary for the boy to go.

```
(p / obligate-01
   :ARG2 (g / go-02
            :ARG0 (b / boy)
            :polarity -))
```

> The boy must not go.
>
> It's obligatory that the boy not go.

```
(t / think-01
   :ARG0 (b / boy)
   :ARG1 (w / win-01
            :ARG0 (t2 / team
                      :poss b)
            :polarity -))
```

> The boy thinks his team won't win.
>
> The boy doesn't think his team will win. (colloquially, ambiguously)

```
(t / think-01
   :ARG0 (b / boy)
   :ARG1 (w / win-01
            :ARG0 (t2 / team
                      :poss b))
   :polarity -)
```

> It's not true that the boy thinks his team will win.
>
> The boy doesn't think his team will win. (colloquially, ambiguously)

```
(h / have-01
   :polarity -
   :ARG0 (i / i)
   :ARG1 (m / money))
```

> I don't have any money.
>
> I have no money.

```
(e / eat-01
   :polarity -
   :ARG0 (p / person
            :mod (e / every)))
```

> No one ate.
>
> Every person failed to eat.

Negative English affixes are also represented with the `:polarity` role:

```
(a / appropriate
  :polarity -
  :domain (t / thing
            :ARG1-of (c / comment-02)))
```

> the comment is inappropriate
>
> the comment is not appropriate

```
(c / comment
   :mod (a / appropriate
          :polarity -))
```

> the inappropriate comment
>
> the comment that is appropriate
>
> the comment that is not appropriate

Wh-Questions

To capture wh-questions, AMR uses the concept `amr-unknown` (in-place!) to indicate wh-questions:

```
(f / find-01
   :ARG0 (g / girl)
   :ARG1 (a / amr-unknown))
```

> What did the girl find?

```
(f / find-01
   :ARG0 (g / girl)
   :ARG1 (b / boy)
   :location (a / amr-unknown))
```

> Where did the girl find the boy?

```
(f / find-01
   :ARG0 (g / girl)
   :ARG1 (b / boy)
   :manner (a / amr-unknown))
```

> How did the girl find the boy?

```
(f / find-01
   :ARG0 (g / girl)
   :ARG1 (t / toy
            :poss (a / amr-unknown)))
```

> Whose toy did the girl find?

```
(r / run-01
   :ARG0 (g / girl)
   :mod (f / fast
          :degree (a / amr-unknown)))
```

> How fast did the girl run?

```
(g / see-01
   :ARG0 (g / girl)
   :ARG1 (a / amr-unknown
            :mod (p / purple)))
```

> What purple thing did the girl see?

```
(l / lead-01
   :ARG0 (s / she)
   :ARG1 (a / amr-unknown
            :domain (i / investigate-01)))
```

> Which investigation did she lead?

Note that wh- words in relative clauses are treated differently, using inverse roles instead of `amr-unknown`:

```
(k / know-01
   :ARG0 (i / i)
   :ARG1 (p / person
            :ARG1-of (s / see-01
                        :ARG0 (y / you))))
```

> I know who you saw.
>
> I know the person you saw.

Other interrogatives & imperatives

AMR uses `:mode` to indicate yes-no questions:

```
(f / find-01
   :ARG0 (g / girl)
   :ARG1 (b / boy)
   :mode interrogative)
```

> Did the girl find the boy?

```
(f / find-01
   :ARG1 (b / boy)
   :mode interrogative)
```

> Was the boy found?

AMR also uses `:mode` for yes-no embedded clauses:

```
(k / know-01
   :polarity -
   :ARG0 (b / boy)
   :ARG1 (c / come-01
            :ARG1 (g / girl)
            :mode interrogative))
```

> The boy doesn't know whether the girl came.
>
> The boy doesn't know if the girl came.

We contrast this with:

```
(k / know-01
   :polarity -
   :ARG0 (b / boy)
   :ARG1 (c / come-01
            :ARG1 (g / girl)))
```

> The boy doesn't know that the girl came.
>
> The boy doesn't know the girl came.

`:mode` is also used for imperatives & exclamations. Exclamatory imperatives are just imperatives in AMR.

```
(g / go
   :mode expressive
   :ARG0 (w / we))
```

> We went!

```
(g / go
   :mode imperative
   :ARG0 (y / you))
```

> Go.
>
> Go!

```
(g / go
   :mode imperative
   :ARG0 (w / we))
```

> Let's go.
>
> Let's go!

Articles, plurals, tense, aspect, quotes, hyphens

AMR 1.2 does not represent event times (outside of the explicit `:time` relation), articles, plurals, or quotation marks:

```
(g / go-02
   :ARG0 (b / boy))
```

> The boy went.
>
> The boys went.
>
> A boy went.
>
> The boy goes.
>
> The boy will go.

Demonstratives are included:

```
(b / boy
   :mod (t / that))
```

> that boy
>
> those boys

```
(b / boy
   :mod (t / this))
```

> this boy
>
> these boys

Demonstrative pronouns are also included if they have no antecedent in the sentence:

```
(s / shame
   :domain (t / that))
```

> that is a shame.

If a hyphenated word can be broken down into component meanings, we do it:

```
(a / account
   :mod (m / market
           :mod (m2 / money)))
```

> money-market account

```
(p / president
   :mod (v / vice))
```

> vice-president
>
> vice president

But when it is hard to get component meanings out, then we leave it together:

```
(b / brother-in-law)
```

> brother-in-law

In any case, we never make the hyphen itself (“-”) into an AMR concept.

Implicit roles

AMR roles may be implicit when rendered in English. AMR includes such roles when there is no real debate about what is happening in the world. Consider:

```
(c / charge-05
   :ARG1 (h / he)
   :ARG2 (a / and
            :op1 (i / intoxicate-01
                    :ARG1 h
                    :location (p / public))
            :op2 (r / resist-01
                    :ARG0 h
                    :ARG1 (a2 / arrest-01
                              :ARG1 h))))
```

> He was charged with public intoxication and resisting arrest.

Here, the variable h appears four times, including as the ARG1 of `arrest-01`, because it is clear that h is resisting his own arrest (not someone else’s). However, we do not include anything to the effect of “the charging agent and the arresting agent are the same entity”, as that is debatable.

Implicit concepts

When we build AMR from text, we introduce implicit roles, but we generally do not introduce implicit *concepts*, e.g., `full` below:

```
(h / hopeful          NOT: (f / full
   :ARG1 (g / girl))          :poss (h / hope)
                              :ARG1 (g / girl))
```

> the hopeful girl

An exception is named entity types for entities that lack one, as covered below under Named Entities.

Main verb “be”

Predicate adjectives are usually represented with `:domain`, unless we have an adjective frame in OntoNotes:

```
(w / white
   :domain (m / marble))
```

> The marble is white.

“Noun is noun” constructions also use `:domain`:

```
(l / lawyer
   :domain (m / man))
```

> The man is a lawyer.

```
(m / man
   :mod (l / lawyer))
```

> the man who is a lawyer

“There is…” and “there are…” have simple AMR representations:

```
(b / boy)
```

> the boy
>
> There is a boy.

```
(b / boy
   :quant 4
   :ARG0-of (m / make-01
               :ARG1 (p / pie)))
```

> four boys making pies
>
> There are four boys making pies.

Nouns that invoke predicates

AMR’s principle is to maximize the use of OntoNotes predicates, regardless of English parts of speech. This section gives examples of this principle.

AMR represents events, not verbs. So, “destroy” and “destruction” have the same AMR representation. For consistency, AMR uses sense-tagged English verbs from OntoNotes:

```
(d / destroy-01
   :ARG0 (b / boy)
   :ARG1 (r / room))
```

> The boy destroyed the room.
>
> The boy’s destruction of the room
>
> The destruction of the room by the boy

By using `destroy-01`, we fully exploit the semantic frames in OntoNotes, which are most developed for English verbs.

We never say:

```
(d / destruction
```

```
   …)
```

or

```
(d / destruction-01
   …)
```

Recent versions of OntoNotes have noun predicate frames like `destruction-01`, but we do not want AMRs to contain both `destroy-01` and `destruction-01`. Therefore, we avoid `destruction-01`.

Some nominalizations (like “explosion”) refer to a whole event, while others (like “proposal”) can refer to role player in the event:

```
(e / explode-01)
```

> the explosion

```
(t / thing
   :ARG1-of (p / propose-01))
```

> the proposal
>
> the thing proposed
>
> what got proposed

We always search for an OntoNotes predicate, even if the noun is much more frequent than the verb (“opinion” is the thing that is “opined”):

```
(t / thing
   :ARG1-of (o / opine-01
               :ARG0 (b / boy)))
```

> the boy’s opinion
>
> the opinion of the boy
>
> that which was opined by the boy
>
> what the boy opined

Inverse roles are also used to represent many “-er” nouns. This enables us to make maximal use of use of OntoNotes predicate frames, instead of defaulting to “`:mod`” or “`:poss`” or “`:prep-in`”:

```
(o / organization                NOT:  (m / maker
   :ARG0-of (m / make-01                  :mod (c / chip))
               :ARG1 (c / chip)))
```

> chip maker
>
> maker of chips

```
(p / person
   :ARG0-of (i / invest-01))
```

> investor

```
(p / person
   :ARG0-of (i / invest-01
               :ARG2 (b / bond)))
```

> bond investor

```
(p / person                        NOT:  (i / investor
   :ARG0-of (i / invest-01                    :mod (s / small))      “I can’t see you!”
               :mod (s / small)))
```

> small investor

```
(p / person
   :ARG0-of (i / invest-01)
   :mod (n / nerd))
```

> nerdy investor

When a noun’s meaning is significantly different from the verbal form, then AMR does not break down its meaning. For example, a “treasurer” is not essentially someone who treasures:

```
(t  / treasurer)           NOT:   (p / person
                                     :ARG0-of (t / treasure-01))
```

> treasurer

```
(p / president)            NOT:   (p / person
                                     :ARG0-of (p2 / preside-01))
```

> president

Even when the meaning of an “-er” noun does break down comfortably, AMR does not automatically reach for an inverse (“-of”) role. For example, “the boy is a hard worker” just means the boy works hard:

```
(w / work-01               NOT:   (b / boy
   :ARG0 (b / boy)                  :ARG0-of (w / work-01
   :manner (h / hard))                         :manner (h / hard)))
```

> the boy is a hard worker
>
> the boy works hard

Adjectives that invoke predicates

Like nouns, adjectives also invoke predicates:

```
(m / man
   :ARG0-of (a / attract-01))
```

> the attractive man

```
(a / attract-01
   :ARG0 (m / man))
```

> the man is attractive
>
> the man attracts

```
(a / attract-01
   :ARG0 (m / man)
   :ARG1 (w / woman))
```

> the man is attractive to women
>
> the man attracts women

Adjectives following “be” can often be represented with OntoNotes verbal predicates:

```
(a / realize-01           NOT:  (a / aware
   :ARG0 (s / soldier)             :ARG1 (s / soldier)
   :ARG1 (b / battle))             :prep-of (b / battle))
```

> The soldier was aware of the battle.
>
> The soldier realized there was a battle.

Many adjectives have natural English verbal predicates:

- be aware (of X) – `realize-01`
- be worth (X) – `value-01`
- be like (X) – `resemble-01`
- be afraid (of X) – `fear-01`

Other adjectives do not, in which case we use the adjective as the predicate name.

- be responsible (for X) – `responsible-41`
- be nervous (about X) – `nervous-41`
- be serious (about X) – `serious-41`
- be efficient (at X) – `efficient-41`

```
(r / responsible-41
   :ARG0 (b / boy)
   :ARG1 (w / work))
```

> The boy is responsible for the work.
>
> The boy is responsible for doing the work.
>
> The boy has the responsibility for the work.

In this way, we avoid awkward syntactic representations for English function words (like “for” and “has”). “The boy responsibles the work” isn’t good English, though it is good Chinese.

ARG0 often refers to the thing being described by the adjective, while ARG1 names the next most natural argument. We use ARG1/ARG2 if the adjective is not agentive.

How about adjectives like “sad”, “white”, and “free”? Should we use `sadden-01`, `whiten-01`, and `free-01`? Just because something is white, it doesn’t mean that it was whitened. In such cases, we only use OntoNotes verbal predicates if there is an implied event or process:

```
(s / sad
   :domain (g / girl))
```

> The girl is sad.

```
(s / sadden-01
   :ARG1 (g / girl)
   :ARG2 (d / disaster))
```

> The girl was saddened by the disaster.
>
> The disaster saddened the girl.

Almost all “-ed” adjectives (e.g., “acquainted”) immediately suggest OntoNotes verb frames. For example:

```
(a / acquaint-01
   :ARG1 (b / boy)
   :ARG2 (m / magic))
```

> the boy is acquainted with magic

The `:ARG0` in such cases (here, the acquainter) is usually unspecified. One may quibble that a boy could become acquainted with magic without someone acquainting him, but maybe he acquainted himself.

OntoNotes is a resourceful “-ed” tool. If you see “X was fed up with Y”, don't be surprised that `feed-03` solves the AMR. What feeder fed Y to X? We just leave the `:ARG0` blank.

By now, these AMRs should not surprise:

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (p / please-01
            :ARG0 b))
```

> Boys are eager to please.
>
> Boys want to please.
>
> Boys are desirous of pleasing.

```
(e / easy
   :domain (p / please-01
              :ARG1 (g / girl)))
```

> Girls are easy to please.
>
> It is easy to please girls.
>
> Pleasing girls is easy.

Note that `please-01` and `girl` are closely related semantically (via `:ARG1`), though not contiguous in “girls are easy to please”. When we see “girls are easy to please”, we automatically re-formulate that as the more logical “easy(please(girl))”. Likewise, with “you are safe to drink this” … “safe(drink(you, this))”.

English adjectives can be formed from verbs and nouns in other ways, e.g., by adding “-able” or “-ful”. If the resulting adjective has its own idiosyncratic meaning, then we do not break it down further. But if we can break it down without introducing new concepts (only relations), then we go ahead:

```
(s / sandwich
   :ARG1-of (e / eat-01
               :mod (p / possible)))
```

> an edible sandwich
>
> a sandwich that can be eaten
>
> a sandwich whose consumption is possible

```
(s / sandwich
   :ARG1-of (e / eat-01
               :mod (p / possible
                       :polarity -)))
```

> an inedible sandwich

English is a wily opponent though. If you own a taxable fund, getting taxed is more than just a possibility, my friend:

```
(f / fund                   NOT: (f / fund
   :ARG3-of (t / tax-01))           :ARG3-of (t / tax-01
                                              :mod (p / possible)))
```

> a taxable fund

Adverbs with -ly

Adverbs get stemmed to adjective form:

```
(o / observe-01
  :ARG0 (i / i)
  :ARG1  (m / move-01
                 :ARG0 (a / army)
                 :manner (q / quick)))
```

> I observed that the army moved quickly.
>
> I observed the quick movement of the army
>
> I observed the army moving quickly.

Non-core roles

We have seen roles like `:time` and `:location`. AMR includes other non-core roles:

`:source`

`:destination`

```
(d / drive-01
   :ARG0 (h / he)
   :direction (w / west)
   :source (c / city :name (n / name :op1 "Houston"))
   :destination (c2 / city :name (n2 / name :op1 "Austin")))
```

> He drove west, from Houston to Austin.

`:path`

```
(d / drive-01
   :ARG0 (i / i)
   :destination (c / city :name (n / name :op1 "Indianapolis"))
   :path (r / road :name (n2 / name :op1 "I-65")))
```

> I drove to Indianapolis on I-65.

```
(d / drive-01
   :ARG0 (i / i)
   :path (t / tunnel))
```

> I drove through the tunnel.

`:beneficiary`

`:accompanier`

```
(s / sing-01
   :ARG0 (s2 / soldier)
   :beneficiary (g / girl)
   :time (w / walk-01
            :ARG0 g
            :accompanier s2
            :destination (t / town)))
```

> The soldier sang to the girl as he walked with her to town.

`:topic`

```
(i / information :polarity -
```

```
   :topic (c / case))
```

> There is no information about the case.

```
(p / person :name (n / name :op1 "Jay" :op2 "Bartroff")
   :ARG0-of (h / have-org-role-91
               :ARG1 (u / university :name (n2 / name :op1 "USC"))
               :ARG2 (p2 / professor
                         :mod (a / associate)
                         :topic (m / mathematics))))
```

> USC Associate Professor for Mathematics Jay Bartroff

`:duration`

```
(w / work-01
   :ARG0 (h / he)
   :duration (t / temporal-quantity
                :quant 2
                :unit (h2 / hour)))
```

> He worked for two hours.

`:instrument`

`:instrument` describes the physical object used in an action, typically a tool, device, weapon, or a body part such as a finger or fist.

If you want to describe how or in what way something is done, use the more general `:manner` instead.

```
(e / eat-01
   :ARG0 (i / i)
   :ARG1 (p / pasta)
   :instrument (f / fork))
```

> I ate pasta with a fork.

```
(a / attack-01
   :ARG0 (c / country :name (n / name :op1 "Iraq"))
   :instrument (m / missile))
```

> Iraq launched a missile attack.

`:medium`

The role `:medium` is used for channels of communications such as a newspaper, a TV channel, the web, YouTube, Facebook, a speech, as well as languages:

```
(t / talk-01
      :ARG0 (s / she)
      :ARG2 (h / he)
      :medium (l / language :name (n / name :op1 "French")))
```

> She talked to him in French.

```
(a / announce-01
  :ARG0 (p / person :name (n / name :op1 "John"))
  :ARG1 (b / bear-02
          :ARG1 (p2 / person
                  :ARG0-of (h / have-rel-role-91
                             :ARG1 p
                             :ARG2 (s / son))))
  :medium (p3 / product :name (n2 / name :op1 "Twitter")))
```

> John announced the birth of his son on Twitter.

`:manner`

`:manner` is used to annotate any description answering "How is something done?" that is not already covered by a more specific role such as `:instrument` or `:medium`.

```
(s / sing-01
   :ARG0 (b / boy)
   :manner (b2 / beautiful
              :degree (v / very)))
```

> The boy sang very beautifully.

```
(d / decorate-01
   :ARG0 (h / he)
   :ARG1 (r / room)
   :manner (c / creative))
```

> He decorated the room in a creative way.

`:manner` also describes the method or action to do something, sometimes referred to as *means*:

```
(p / propose-01
   :ARG0 (p3 / person
            :ARG0-of (h2 / have-org-role-91
                       :ARG2 (m / mayor)))
   :ARG1 (l / lower-01
            :ARG1 (c / crime)
            :manner (h / hire-01
                       :ARG2 (p4 / person
                                :ARG0-of (h3 / have-org-role-91
                                           :ARG1 (p2 / police)
                                           :ARG2 (o / officer))
                                :mod (m2 / more)))))
```

> The mayor proposed to lower crime by hiring more police officers.

`:manner` also describes modes of transportation:

```
(g / go-02
   :ARG0 (p / person :name (n / name :op1 "Nicole"))
   :ARG4 (c / country :name (n2 / name :op1 "England"))
```

```
   :manner (t / train))
```

> Nicole went to England by train.

`:purpose`

```
(g / go-02
   :ARG0 (h / he)
   :ARG4 (s / store)
   :purpose (b / buy-01
               :ARG0 h
               :ARG1 (w / wood
                        :purpose (f / fence
                                    :mod (n / new)))))
```

> He went to the store to buy wood for a new fence.

`:cause`

```
(m / murmur-01
  :ARG0 (b / boy)
  :manner (s / soft)
  :purpose (s2 / soothe-01
              :ARG1 (g / girl))
  :cause (w / worry-01
            :ARG0 b
            :topic g))
```

> The boy murmured softly to soothe the girl, because he worried about her.

Note: The AMR Editor automatically reifies :cause and :cause-of to cause-01, so we call `:cause` a *shortcut*.

`:concession`

```
 (c / continue-01
    :ARG1 (g / game)
    :concession (r / rain-01))
```

> The game continued although it rained.
>
> The game continued despite the rain.

```
(c / continue-01
   :ARG1 (g / game)
   :concession (e / even-if
                  :op1 (r / rain-01)))
```

> The game will continue even if it rains.

```
(f / fear-01
   :ARG0 (t / they)
   :ARG1 (h / he)
   :concession (e / even-when
```

```
                :op1 (i / imprison-01
                       :ARG1 h)))
```

> They feared him even when he was behind bars.

`:condition`

```
(s / sing-01
   :ARG0 (b / boy)
   :condition (g / give-01
                 :ARG1 (m / money)
                 :ARG2 b))
```

> The boy will sing if he is given money.
>
> If the boy is given money, he will sing.
>
> The boy will sing in case of a money donation.

```
(s / sing-01
   :ARG0 (b / boy)
   :polarity -
   :condition (g / give-01
                 :ARG1 (m / money)
                 :ARG2 b))
```

> The boy will sing unless he is given money.
>
> Unless the boy is given money, he will sing.

In AMR, `X :cause Y` means that the cause of X is Y. Likewise, `Y :cause-of X` means Y is the cause of X. (See the section Reification below about using the concept `cause-01` instead of `:cause` or `:cause-of`.)

```
 (s / strike-01
   :ARG0 (t / torpedo)
   :cause-of (d / damage-01
                :ARG1 (s2 / ship)))
```

> The torpedo struck, causing the ship to be damaged.
>
> The torpedo struck, causing damage to the ship.
>
> The torpedo struck, damaging the ship.

It can be difficult to tease apart `:purpose` from `:cause`. For example, “I visited her because she was sick” (cause) or “I visited her to deliver the news” (purpose).

In the AMR Editor, :cause/:cause-of is automatically converted to cause-01. (See section on reification below.)

Occasionally, a numbered `:ARGx` role will refer exactly to `:location`, `:beneficiary`, or some other named non-core role. In this case, we use the `:ARGx` role, e.g.:

```
(p / provide-01            NOT:  (p / provide-01
   :ARG0 (b / boy)                  :ARG0 (b / boy)
   :ARG1 (c / chocolate)            :ARG1 (c / chocolate)
   :ARG2 (g / girl))                :beneficiary (g / girl))
```

> The boy provided chocolate to the girl.
>
> The boy provided the girl with chocolate.

Sometimes it isn't clear what `:location`, `:time`, etc., should modify in AMRs involving creation events. We tend to put them on the event, rather than on the created thing:

```
(b / build-01                        NOT:  (b / build-01
  :ARG0 (t / they)                           :ARG0 (t / they)
  :ARG1 (b2 / bridge)                        :ARG1 (b2 / bridge :location …))
  :location (s / state :name (n / name :op1 "Maryland"))
  :time (d / date-entity :month 12))
```

> They built the bridge in Maryland in December.
>
> The bridge was built by them in Maryland in December.

Here are more non-core roles:

`:part`

```
(e / engine
   :part-of (c / car))
```

> the engine of the car
>
> the car's engine

```
(u / unit
   :part-of (c / company))
```

> a unit of the company
>
> the company's unit

We do not use `:part` for set membership, as in the CEO of a company.

```
(s / south
   :part-of (c / country :name (n / name :op1 "France")))
```

> the south of France
>
> southern France

`:subevent`

```
(w / win-01
   :ARG0 (b / boy)
   :ARG1 (r / race-01
            :subevent-of (g / game :name (n / name :op1 "Olympics"))))
```

> The boy won the race in the Olympics.

`:consist-of`

```
(r / ring
   :consist-of (g / gold))
```

> a ring of gold

```
(t / team
   :consist-of (m / monkey))
```

> a team of monkeys

`:example`

```
(c / company
   :example (a / and
               :op1 (c2 / company :name (n / name :op1 "IBM"))
               :op2 (c3 / company :name (n2 / name :op1 "Google"))))
```

> companies like IBM and Google

`:direction`

```
(d / drive-01
   :ARG0 (h / he)
   :direction (w / west))
```

> He drove west.

`:frequency`

`:frequency` describes how often something occurs.

```
(m / meet-03
   :frequency 3
   :ARG0 (w / we))
```

> We met three times.

The special frame `rate-entity-91` is used to describe recurring events and other rate entities such as "every 3000 miles" or "$3 per gallon".

```
(r / rate-entity-91
   :ARG1 2
   :ARG2 (t / temporal-quantity
            :quant 1
            :unit (y / year)))
```

> twice a year

```
(p / play-01
   :ARG0 (w / we)
   :ARG1 (b / bridge)
   :frequency (r / rate-entity-91
                 :ARG4 (d / date-entity
                          :weekday (w2 / wednesday)
                          :dayperiod (a / afternoon))))
```

> We play bridge every Wednesday afternoon.

Core roles of `rate-entity-91`:

- `:ARG1` of `rate-entity-91` is the quantity (default: 1)
- `:ARG2` of `rate-entity-91` is the reference quantity ("per quantity")
- `:ARG3` of `rate-entity-91` is any *regular* interval between events ("every 2 months" - more specific than `:ARG2`)
- `:ARG4` of `rate-entity-91` is any entity on which recurring events happen

`:extent`

```
(g / go-15
      :ARG1 (r / road)
      :extent (f / forever))
```

> The road goes on forever.

Focus

Inverse relations are often used for focusing (see Introduction):

```
(s / sing-01
   :ARG0 (b / boy
            :source (c / college)))
```

> The boy from the college sang.

```
(b / boy
   :ARG0-of (s / sing-01)
   :source (c / college))
```

> The singing boy from the college
>
> There is a boy from the college who sang.

```
(c / college
   :source-of (b / boy
                 :ARG0-of (s / sing-01)))
```

> The college that the singing boy came from

The concept of focus only applies at the very top (root) of the AMR. After a root concept is selected, there are no more focus considerations -- all else is driven strictly by semantic relations. For example, once we have selected `c / college` at the root, then `:source-of` must be filled with `boy`, not with `sing-01`.

Reification

Sometimes we want to use an AMR relation as a first-class concept. Converting a role into a concept is called *reification*. Here is an example, where the relation `:cause` is replaced by `cause-01`. Instead of `x :cause y`, we have `x :ARG1-of (c / cause-01 :ARG0 y)`.

```
AMR without reification:        AMR with reification:
(l / leave-01                   (l / leave-01
   :ARG0 (g / girl)                :ARG0 (g / girl)
   :cause (a / arrive-01           :ARG1-of (c / cause-01
             :ARG1 (b / boy)))                 :ARG0 (a / arrive-01
                                                      :ARG1 (b / boy))))
```

> The girl left because the boy arrived.

AMR without reification is simpler, so why would we want to reify? One reason is to make a relation the focus of an AMR fragment. For example, suppose we know there is a knife in the drawer. We might try focusing on the knife:

```
(k / know
   :ARG0 (w / we)
   :ARG1 (k2 / knife
             :location (d / drawer)))
```

> We know the knife that is in the drawer. (???)

Or we might try focusing on the drawer:

```
(k / know
   :ARG0 (w / we)
   :ARG1 (d / drawer
            :location-of (k2 / knife)))
```

> We know the drawer where the knife is. (???)

But we really want to focus on the locating itself. AMR therefore supplies reifications for many relations. In the case of `:location`, the reification is `be-located-at-91`, allowing us to say:

```
(k / know
   :ARG0 (w / we)
   :ARG1 (b / be-located-at-91
            :ARG0 (k2 / knife)
            :ARG1 (d / drawer)))
```

> We know the knife is in the drawer.

Note that `be-located-at-00` has two roles, `:ARG0` (the thing that exists in space) and `:ARG1` (where the thing is).

We also use reification when we want to modify a relation. For example:

```
(k / know
   :ARG0 (w / we)
   :ARG1 (b / be-located-at-91
            :ARG0 (k2 / knife)
            :ARG1 (d / drawer)
            :polarity -
            :time (y / yesterday)))
```

> We know the knife was not in the drawer yesterday.

Here are the AMR reifications. Reifications often correspond to OntoNotes predicates, in which case, we just use the `:ARG` relations in the natural way.

> Read this chart as: `x :Relation y` = `x :Domain-of (z / Reification :Range y)`
>
> For example: `x :location y` = `x :ARG0-of (b / be-located-at-91 :ARG1 y)`

Relation	Reification	Domain	Range	Example
`:accompanier`	`accompany-01`	`:ARG0`	`:ARG1`	“she's with him”
`:age`	`age-01`	`:ARG1`	`:ARG2`	“she's 41 years old”
`:beneficiary`	`benefit-01`	`:ARG0`	`:ARG1`	“the 5k run is for kids”
`:cause`	`cause-01`	`:ARG1`	`:ARG0`	“he came 'cause of her”
`:concession`	`have-concession-91`	`:ARG1`	`:ARG2`	“he came despite of her”
`:condition`	`have-condition-91`	`:ARG1`	`:ARG2`	“he comes if she comes”
`:destination`	`be-destined-for-91`	`:ARG1`	`:ARG2`	“i'm off to Atlanta”
`:duration`	`last-01`	`:ARG1`	`:ARG2`	“it's 15 minutes long”
`:example`	`exemplify-01`	`:ARG0`	`:ARG1`	“cities such as Atlanta”
`:frequency`	`have-frequency-91`	`:ARG1`	`:ARG2`	“he came three times”
`:instrument`	`have-instrument-91`	`:ARG1`	`:ARG2`	“forks are for eating”
`:location`	`be-located-at-91`	`:ARG1`	`:ARG2`	“she's not here”
`:manner`	`have-manner-91`	`:ARG1`	`:ARG2`	“it was done quickly”
`:mod`	`have-mod-91`	`:ARG1`	`:ARG2`	“he is half Chinese"
`:name`	`have-name-91`	`:ARG1`	`:ARG2`	“the city formerly named Constantinople”
`:part`	`have-part-91`	`:ARG1`	`:ARG2`	“the roof of the house”
`:polarity`	`have-polarity-91`	`:ARG1`	`:ARG2`	“I don't know.”
`:poss`	`own-01`, `have-03`	`:ARG0`	`:ARG1`	“that dog's not mine”
`:purpose`	`have-purpose-91`	`:ARG1`	`:ARG2`	“it's to eliminate bugs”
`:quant`	`have-quant-91`	`:ARG1`	`:ARG2`	“there are 4 rabbits”
`:source`	`be-from-91`	`:ARG1`	`:ARG2`	“she's from Ipanema”
`:subevent`	`have-subevent-91`	`:ARG1`	`:ARG2`	“presentation at a conference”
`:subset`	`include-91`	`:ARG2`	`:ARG1`	“10% of the workers”

`:time`	`be-temporally-at-91`	`:ARG1`	`:ARG2`	“the party is on friday”
`:topic`	`concern-02`	`:ARG0`	`:ARG1`	“the show's about me”

These relations do not have reifications:

- `:ARG0`, `:ARG2`, `:ARG2`, ... `:op1`, `:op2`, `:op3`, `:op4`, ...
- `:calendar`, `:century`, `:day`, `:dayperiod`, `:decade`, `:era`, `:month`, `:quarter`, `:season`, `:timezone`, `:weekday`, `:year`, `:year2`
- `:unit`, `:value`, `:mod`, `:mode`, `:compared-to`, `:degree`, `:direction`, `:name`, `:scale`

Now, the question remains: when to reify?

One potential answer is “whenever you feel like it”. Unfortunately, a single sentence may receive two different AMRs. Either of the following AMRs could reasonable represent “The girl left because the boy arrived”, with neither being canonical:

```
AMR without reification:        AMR with reification:
(l / leave                      (l / leave
   :ARG0 (g / girl)                :ARG0 (g / girl)
   :cause (a / arrive              :ARG1-of (c / cause-01
             :ARG0 (b / boy)))                 :ARG0 (a / arrive
                                               :ARG0 (b / boy))))
```

Furthermore, we might reasonably tend to use the first AMR for “The girl left because the boy arrived”, but prefer the second one for “The girl left, due to the boy's arrival”. So we cannot guarantee that these two sentences get the same AMR.

A second potential answer is “reify all the time”, which would completely eliminate relations like `:cause`, `:location`, and `:subset` from AMR, in favor of concepts like `cause-01`, `be-located-at-00`, and `include-91`. But this is cumbersome -- it is easy and typical to simply type `:location`.

The resolution: we consider “AMR with reification” to be “real AMR”, with non-reified relations as semantic sugar. Therefore, if you are translating English into AMR, the rule is “whenever you feel like it”, because your AMRs will be normalized into reified form behind the scenes.

Phrasal verbs

AMR strips away light-verb constructions:

```
(a / adjust-01
   :ARG0 (g / girl)
   :ARG1 (m / machine))
```

> The girl adjusted the machine.
>
> The girl made an adjustment to the machine.

```
(t / bathe-01
   :ARG0 (b / boy))
```

> The boy bathed.
>
> The boy took a bath.

It also combines verb-particle constructions, using OntoNotes predicate frames. Here `look-05` is defined as “look up: seek”.

```
(l / look-05
   :ARG0 (b / boy)
   :ARG1 (a / answer))
```

> The boy looked up the answer.
>
> The boy looked the answer up.

Sometimes a particle doesn't change the meaning of the verb very much, but OntoNotes may still have two separate predicates. For example, `close-06` means "become nearer", while `close-07` is "close in: become nearer". In such cases, AMR canonicalizes to the *non-particle* frame, e.g., `close-06`.

Prepositions

Most prepositions that signal semantic frame elements are dropped in AMR:

```
(s / default-01
   :ARG1 (n / nation)
   :time (d / date-entity
            :month 6))
```

> The nation defaulted in June.

```
(d / die-01
   :ARG1 (m / man)
   :location (h / house
                :poss m))
```

> The man died in his house.

But time and location prepositions are kept if they carry additional information, using AMR's `:opN`. This `:op1` is different from the `:op1` used in conjunctions.

```
(s / default-01
   :ARG0 (n / nation)
   :time (b / after
            :op1 (w / war-01)))
```

> The nation defaulted after the war.

```
(d / die-01
   :ARG1 (m / man)
   :location (n / near
                :op1 (h / house
                        :poss m)))
```

> The man died near his house.

```
(d / die-01
   :ARG1 (m / man)
   :location (b / between
                :op1 (h / house)
```

```
                :op2 (r / river)))
```

> The man died between the house and the river.

Sometimes, the content of a prepositional phrase cannot be easily slotted into a predicate-argument structure, or into a generic role like `:time` or `:location`. AMR cringes while employing a default `:prep-x` representation:

```
(s / sue-01
   :ARG1 (h / he)
   :prep-in (s / case))
```

> He was sued in the case.

AMR combines phrasal prepositions:

```
(f / file
   :ARG1 (b / brief)
   :prep-on-behalf-of (g / government))
```

> The brief was filed on behalf of the government.

By tradition, the frequent phrase “according to” gets special handling:

```
(s / say-01
   :ARG0 (s2 / source
             :mod (g / government))
   :ARG1 (k / kill-01
            :time (y / yesterday)))
```

> According to government sources, the killing happened yesterday.
>
> Government sources said that the killing happened yesterday.

Relative clauses

AMR frequently represents relative clauses with inverse roles, as described in the Introduction:

```
(b / believe-01
   :ARG0 (b2 / boy))
```

> The boy believes.

```
(b / boy
   :ARG0-of (b2 / believe-01))
```

> the boy who believes

English also uses relative clauses when negating a pre-nominal adjective is difficult (“the not-black car”):

```
(c / car
   :mod (b / black))
```

> the black car

```
(c / car
   :mod (b / black
           :polarity -))
```

> the car that is not black

Japanese simply marks adjectives with a negative suffix.

Multiple relations with the same name

An entity may have several relations with the same name:

```
(s / system
   :mod (l / law)
   :mod (s2 / city
           :name (n / name :op1 "Shanghai")))
```

> the Shanghai legal system

```
(b / boy
   :ARG0-of (w / want-01
               :ARG1 (b2 / believe-01
                        :ARG1 (g / girl)))
   :ARG0-of b2)
```

> the boy who wants to believe the girl

Conjunctions

To represent conjunction, AMR uses concepts `and`, `or`, `contrast-01`, `either`, and `neither`, along with `:opx` relations:

```
(a / and
   :op1 (b / boy)
   :op2 (g / girl))
```

> the boy and the girl

```
(a / either
   :op1 (b / boy)
   :op2 (g / girl)
   :op3 (d / dog))
```

> either the boy, the girl, or the dog

Conjoined adjectives are done without `and`:

```
(b / ball
```

```
   :mod (b2 / big)
   :mod (h / heavy))
```

> the big, heavy ball
>
> the big and heavy ball

`:opx` is also used for clauses:

```
(a / and
   :op1 (c / shout-01)
   :op2 (l / leave-01
           :ARG0 (b / boy)))
```

> There was shouting, and the boy left.

```
(c / contrast-01
   :ARG1 (c2 / shout-01)
   :ARG2 (l / stay-01
            :ARG1 (b / boy)))
```

> There was shouting, but the boy stayed.

Sometimes, an `:op1` or `:ARG1` may be missing:

```
(c / contrast-01
   :ARG2 (l / stay-01
            :ARG1 (b / boy)))
```

> But the boy stayed.

AMR aims for a logical representation even when English elides core actors:

```
(a / and
   :op1 (c / shout-01
           :ARG0 (b / boy))
   :op2 (l / leave-01
           :ARG0 b))
```

> The boy shouted and left.

The need for this is evident when an entity plays different roles in different predicates:

```
(a / and
   :op1 (a2 / arrive-01
            :ARG1 (b / boy))
   :op2 (l / kill-01
           :ARG1 b
           :manner (p / prompt)))
```

> The boy arrived and was promptly killed.

However, AMR “pulls out” non-core roles like `:time` and `:location`. Here, `:time` modifies the entire conjunction rooted by `and`:

```
(a / and
   :time (d / date-entity
            :weekday (t / tuesday))
   :op1 (a2 / arrive-01
            :ARG1 (b / boy))
   :op2 (l / leave-01
          :ARG0 b))
```

> The boy arrived and left on Tuesday.
>
> On Tuesday, the boy arrived and left.

Quantifiers and scope

AMR does not have a deep representation for quantifiers. It only canonicalizes their position:

```
(l / leave-01
   :ARG0 (b / boy
            :mod (a / all)))
```

> The boys all left.
>
> All the boys left.
>
> Each of the boys left.

```
(l / leave-01
   :ARG0 (b / boy
            :mod (n / no)))
```

> No boy left.
>
> None of the boys left.

```
(l / leave-01
   :ARG0 (b / boy
            :mod (a / all
                    :polarity -)))
```

> Not all of the boys left.

```
(l / leave-01
   :ARG0 (p / person
           :mod (a / all
                   :polarity -)))
```

> Not everyone left.

The placement of `:polarity` can be troublesome. Consider:

```
(b / believe-01
   :ARG0 (g / girl)
   :ARG1 (w / work-01
```

```
            :ARG0 (b / boy)
            :manner (h / hard)))
```

> The girl believes that the boy works hard.

If we want to represent “the girl doesn’t believe that the boy works hard”, we have to decide whether to place the negative polarity under “believe” or “work” or “hard”. Here it should go under “hard”:

```
(b / believe-01
   :ARG0 (g / girl)
   :ARG1 (w / work-01
            :ARG0 (b2 / boy)
            :manner (h / hard
                      :polarity -)))
```

> The girl believes that the boy works in a not-hard manner.
>
> The girl believes that the boy doesn’t work hard. (colloquially)
>
> The girl doesn’t believe that the boy works hard. (colloquially)

If we put `:polarity` elsewhere, we change the meaning:

```
(b / believe-01
   :ARG0 (g / girl)
   :ARG1 (w / work-01
            :polarity -
            :ARG0 (b2 / boy)
            :manner (h / hard)))
```

> The girl believes that the boy refrains from work, in a hard manner.

```
(b / believe-01
   :polarity -
   :ARG0 (g / girl)
   :ARG1 (w / work-01
            :ARG0 (b2 / boy)
            :manner (h / hard)))
```

> It’s not true that the girl believes the boy works hard.

```
(b / believe-01
   :ARG0 (g / girl
            :polarity -)
  :ARG1 (w / work-01
           :ARG0 (b2 / boy
                     :polarity -)
           :manner (h / hard)))
```

> The non-girl believes that the non-boy works hard.

AMR apologizes for not advising on the placement of negation with respect to quantifiers.

Degree

Comparatives and superlatives are represented by `:degree` and `:compared-to`, e.g.:

```
(b / bright
   :domain (b2 / boy
               :mod (t / that))
   :degree (m / more))
```

> That boy is brighter.
>
> That boy is more bright.

```
(b / bright
   :domain (b2 / boy
               :mod (t / that))
   :degree (m / most))
```

> That boy is the brightest.
>
> That boy is the most bright.

```
(p / plan-01
   :time (e / early
            :degree (m / more)))
```

> the earlier plan

```
 (p / plan-01
    :mod (g / good
            :degree (m / more)))
```

> a better plan

```
(p / plan-01
   :mod (g / bad
           :degree (m / more)))
```

> a worse plan

```
(p / plan-01
   :mod (e / extreme
           :degree (t / too)))
```

> a plan that is too extreme

```
(t / tall
   :degree (m / more)
   :domain (g / girl)
   :compared-to (b / boy))
```

> the girl is taller than the boy

```
(g / girl
   :mod (t / tall
           :degree (m / most)
           :compared-to (t2 / team))
   :domain (s / she))
```

> she is the tallest girl on the team

AMR apologizes, realizing that the girl is not taller than the whole team, but taller than each individual.

Variables and co-reference

If two variables are the same, then they refer to the same entity:

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (g / go-02
            :ARG0 b))
```

> The boy wants to go.

In English, overt and zero pronouns are often used to realize co-reference, but AMR uses variables instead:

```
(w / want-01
   :ARG0 (b / boy)
   :ARG1 (b2 / believe-01
             :ARG0 b
             :ARG1 b))
```

> The boy wants to believe himself.

If an overt pronoun has no antecedent within the sentence, AMR uses the pronoun:

```
(s / see-01
   :ARG0 (h / he)
   :ARG1 (s / they))
```

> He saw them.

Pronouns in AMR are always nominative (he, she, i, we, they, …) and never accusative (him, her, me, us, them, …). AMR uses the token `y'all` for you plural:

```
(s / see-01
   :ARG0 (i / i)
   :ARG1 (y / y'all))
```

> I saw you (plural).
>
> I saw y'all.
>
> I saw you guys.

Possession

The relation `:poss` (“possessed by”) is a very general form of possession. AMR uses it only for possessives and prepositional phrases with “of”:

```
(c / car
   :poss (h / he))
```

> his car
>
> the car of his

```
(t / titan
   :poss (n / nation))
```

> the nation’s titans

Not all possessives and “of” phrases are represented with `:poss`. AMR often uses `:part-of`, `:consist-of`, etc.

Pertainyms

Pertainym adjectives (e.g., “atomic, adj. = of, or pertaining to, atoms”) do not appear in AMR. Only the stemmed noun form is used, along with the `:mod` relation.

```
(v / virus
   :mod (m / microbe))
```

> microbial virus
>
> microbe virus

```
(w / war
   :mod (a / atom))
```

> atomic war

A pertainym may get stemmed to noun form, and then subsequently to verb form:

```
(p / problem
   :mod (b / behave-01))
```

> Behavioral problems.
>
> Behavior problems.
>
> Problems with behavior.
>
> Problems behaving.

When building AMR from English, the rule is to continue stemming toward verb form unless the meaning is significantly altered.

Ordinals

We use the role `:ord` and the concept `ordinal-entity` to express ordinals.

```
(p / planet
   :ord (o / ordinal-entity
           :value 2))
```

> the second planet
>
> planet number 2

```
(v / visit-01
   :ARG0 (w / we)
   :ord (o / ordinal-entity
           :value 1
           :range (t / temporal-quantity
                     :quant 10
                     :unit (y / year))))
```

> our first visit in 10 years

Subsets

We often refer to subsets when we speak. AMR uses roles `:subset` and `:subset-of`.

```
(d / die-01
   :ARG1 (s / soldier
            :quant 9
            :subset-of (s3 / soldier
                           :quant 20)))
```

> Nine of the twenty soldiers died.

```
(h / have-03
   :ARG0 (p4 / person
             :quant 4
             :subset-of (p2 / person
                            :ARG0-of (s / survive-01)
                            :quant 5)
             :subset (p3 / person
                         :quant 3
                         :ARG1-of (d3 / diagnose-01)))
   :ARG1 (d / disease))
```

> Four of the five survivors had the disease, including three who were diagnosed.

Features shared by a subset and its superset go into the superset only, e.g., “survive” above. The reification of `:subset` is `include-91`, so we can equivalently write:

```
(d / die-01
   :ARG1 (s / soldier
            :quant 9
```

```
            :ARG1-of (i / include-91
                        :ARG2 (s3 / soldier
                                  :quant 20))))
```

> Nine of the twenty soldiers died.

AMR is sparing with `:subset` -- otherwise things get out of control. For example, we do not use it for “Three of the workers at the plant,” but we rather just interpret this as “Three workers”.

Note: The AMR Editor automatically reifies :subset, :subset-of, :superset and :superset-of to include-91, so we call them *shortcuts*.

Named Entities

Any concept instance in AMR can have `:name` role. We are not restricted to a small set of fixed categories like countries and people: ships, pets, and computers can also have names.

```
(p / person
   :name (n / name
            :op1 "Mollie"
            :op2 "Brown"))
```

> Mollie Brown

```
(p / person
   :name (n / name
            :op1 "Mollie"
            :op2 "Brown")
   :ARG0-of (s / slay-01
               :ARG1 (o / orc)))
```

> the orc-slaying Mollie Brown
>
> Mollie Brown, who slays orcs

```
(s / ship
   :name (n / name
            :op1 "Titanic"))
```

> Titanic
>
> the Titanic
>
> the ship named Titanic

```
(c / city
   :name (n / name
            :op1 "Marina"
            :op2 "del"
            :op3 "Rey"))
```

> Marina del Rey
>
> the city of Marina del Rey

AMR strings words with `:opN`. It does not analyze semantic relationships inside a named entity. For example, in the “Stop Malaria Foundation”, we do not invoke the predicate `stop-01` with `malaria` as its `:ARG1`.

Abbreviations of proper names are not expanded, but abbreviated common nouns are expanded. Speaking of common words, we correct typos, and we normalize to American spelling, but we do not otherwise normalize variants.

```
(s / state
   :name (n / name
            :op1 "Calif."))
```

> Calif.

```
(r / rate
   :mod (a / advertise-01))
```

> advertising rates
>
> ad rates

When building AMRs for proper names or “-er” nouns, we need to fill the root concept (or top-level `:instance` role). In doing so, we face one of three situations.

(a) In general, unless the English text provides something more specific type, we fill the `:instance` slot from a special list of standard AMR named entity types, e.g. `person` and `company`. In such cases, we basically must hallucinate an entity type. For example:

```
(p / person
   :name (n / name
            :op1 "Pascale"))
```

> Pascale

```
(c / company
   :ARG0-of (m / make-01
               :ARG1 (c / chip)))
```

> the chip maker

However, we do not want some AMRs to say “person” and others to say “woman”, or some to say “company”, and others to say “organization”.

So when we are forced to hallucinate an entity type, AMR requires us to draw from this canonical list (borrowing from information extraction and question answering):

- **person**, family, animal, language, nationality, ethnic-group, regional-group, religious-group
- **organization**, company, government-organization, military, criminal-organization, political-party, school, university, research-institute, team, league
- **location**, city, city-district, county, local-region, state, province, country, country-region, world-region, continent, ocean, sea, lake, river, gulf, bay, strait, canal, peninsula, mountain, volcano, valley, canyon, island, desert, forest, moon, planet, star, constellation
- **facility**, airport, station, port, tunnel, bridge, road, railway-line, canal, building, theater, museum, palace, hotel, worship-place, market, sports-facility, park, zoo, amusement-park
- **event**, incident, natural-disaster, earthquake, war, conference, game, festival

- **product**, vehicle, ship, aircraft, aircraft-type, spaceship, car-make, work-of-art, picture, music, show, broadcast-program
- **publication**, book, newspaper, magazine, journal
- **natural-object**
- law, treaty, award, food-dish, disease

We always choose the most specific applicable type.

If none of these apply, then we use **thing**.

Some examples:

```
(a / award
   :name (n / name
            :op1 "Nobel"
            :op2 "Prize"))
```

> the Nobel Prize

```
(g / government-organization
   :name (n / name
           :op1 "NSA")
   :mod (c / country
       :name (n2 / name :op1 "America")))
```

> the American NSA

```
(n / natural-object
   :name (n2 / name
             :op2 "Lone"
             :op3 "Cypress"))
```

> the Lone Cypress

(b) If the text contains a more specific English term to describe the type of entity, we use it instead to fill the `:instance` role. For example:

```
(p / poet
   :name (n / name :op1 "William" :op2 "Shakespeare"))
```

> the poet William Shakespeare
>
> William Shakespeare, the poet

`poet` is more specific than `person`.

```
(v / village
   :name (n / name
            :op1 "Odinaboi"))
```

> the village of Odinaboi

`village` is more specific than `city`.

```
(d / doctor
```

```
   :name (n / name
            :op1 "Wu"))
```

> Doctor Wu

The following example texts mention `region`, `party` and `spacecraft`, but in these case we prefer the standard NE types `country-region`, `political-party` and `spaceship`, because the latter are more (or at least equally) specific:

```
(c / country-region
   :name (n / name :op1 "Darfur")
   :location (c2 / country :name (n2 / name :op1 "Sudan")))
```

> Sudan's Darfur region

```
(p / political-party
   :name (n / name :op1 "CDU")
   :mod (c / conservative)
   :mod (c2 / country :name (n2 / name :op1 "Germany")))
```

> Germany's conservative CDU party

```
(s / spaceship
   :name (n / name :op1 "Shenzhou"))
```

> the spaceship Shenzhou
>
> the Shenzhou spacecraft

Mere honorifics such as “Mr.”, “Mrs.”, etc. are included as part of the name:

```
(p / person
   :name (n / name
            :op1 "Mr."
            :op2 "Wu"))
```

> Mr. Wu
>
> Mister Wu

See the next section on "Special Frames for Roles" on how to annotate titles such as "President".

When faced with an appositive, AMR calmly inserts facts into slots:

```
(g / group
   :name (e / name
            :op1 "Elsevier"
            :op2 "N.V.")
   :mod (c / country
           :name (h / name
                    :op1 "Netherlands"))
   :ARG0-of (p2 / publish-01))
```

> Elsevier N.V. , the Dutch publishing group

We view this object semantically as a “group”, which happens to have a known `:name`, plus some a couple of other properties

that describe it.

(c) The text contains *multiple* English words vying for the same `:instance` slot. This happens occasionally. Because `:instance` is the only relation that cannot physically appear twice in AMR, we instead open up the inverse of `:domain` , i.e. the role `:mod`

```
(d / doctor :name (n / name :op1 "Seuss")
  :mod (p / poet))
```

> the poet Dr. Seuss

In all cases, hyphenated and possessive words inside names are kept intact, not broken up.
For example, "Dana-Farber Materials" only has `:op1` and `:op2`.

Special Frames for Roles

For roles in organizations, we use the frame have-org-role-91:

```
(p / person
   :name (n / name :op1 "Obama")
   :ARG0-of (h / have-org-role-91
               :ARG1 (c / country :name (n2 / name :op1 "US"))
               :ARG2 (p2 / president)))
```

> US President Obama

Core roles of `have-org-role-91`:

- `:ARG0` of `have-org-role-91` is the office holder, typically a person
- `:ARG1` of `have-org-role-91` is the organization, which could also be a GPE
- `:ARG2` of `have-org-role-91` is the title of the office held, e.g. president
- `:ARG3` of `have-org-role-91` is a description of responsibilty (rarely used)

Typical have-org-role-91 roles: ambassador, archbishop, bishop, CEO, chairman, chancellor, chief of staff, commissioner, congressman, deputy, dictator, director, emperor, empress, envoy, foreign minister, governor, king, mayor, monarch, officer, official, pope, premier, president, principal, professor, queen, secretary, senator, spokesman, spokeswoman, treasurer etc.

For roles that describe the relation between two people (or two other entities of the same type), we use the frame have-rel-role-91:

```
(h / have-rel-role-91
  :ARG0 (h2 / he)
  :ARG1 (i / i)
  :ARG2 (b / brother-in-law))
```

> He is my brother-in-law.

Core roles of `have-rel-role-91`:

- `:ARG0` of `have-rel-role-91` entity A
- `:ARG1` of `have-rel-role-91` entity B
- `:ARG2` of `have-rel-role-91` role of entity A (must be specified)
- `:ARG3` of `have-rel-role-91` role of entity B (often left unspecified)
- `:ARG4` of `have-rel-role-91` relationship basis (contract, case; rarely used)

Typical have-rel-role-91 roles: father, sister, husband, grandson, godfather, stepdaughter, brother-in-law; friend, boyfriend, buddy, enemy; landlord, tenant etc.

Exact numbers

AMR normalizes numbers:

```
(b / boy
   :quant 40000)
```

> forty thousand boys
>
> 40,000 boys

```
(a / atom
   :quant 1500000000)
```

> one and half billion atoms
>
> 1.5 billion atoms
>
> a billion and half atoms
>
> 1,500,000,000 atoms

Such normalization is often necessary when we translate between Asian-style 10,000-based numeration and Western-style 1,000-based numeration.

Approximate numbers

Approximate numbers are represented with this `:opN` notation:

```
(b / boy
   :quant (s / several
             :op1 100))
```

> several hundred boys

```
(b / boy
   :quant (m / more-than
             :op1 4000))
```

> more than four thousand boys
>
> more than 4000 boys

```
(b / boy
   :quant (m / between
             :op1 4000
             :op2 5000))
```

> between 4000 and 5000 boys
>
> between four and five thousand boys

Quantities

Exact quantities are represented by their type and `:unit` and `:quant` arguments.

```
(q / distance-quantity
   :unit (m / mile)
   :quant 10)
```

> ten miles
>
> 10 miles
>
> 10-mile

Approximate quantities are represented using `:opN` notation, as for approximate numbers:

```
(a / about
    :op1 (q / distance-quantity
            :unit (m / mile)
            :quant 10))
```

> about 10 miles

AMR views quantified expressions like “two gallons of milk” as “milk”:

```
(b / buy-01
   :ARG0 (w / woman)
   :ARG1 (m / milk
            :quant (q / volume-quantity
                      :unit (g / gallon)
                      :quant 2)))
```

> The woman bought two gallons of milk.

For stretches of time and relative times, AMR uses `temporal-quantity`.
(For absolute times, AMR uses `date-entity`, described in the next section.)

```
(t / temporal-quantity
   :unit (y / year)
   :quant 30)
```

> 30 years

```
(b / before
   :op1 (n / now)
   :duration (t / temporal-quantity
                :unit (y / year)
                :quant 30))
```

> during the past 30 years

```
(b / before
   :op1 (n / now)
   :quant (t / temporal-quantity
             :unit (y / year)
             :quant 30))
```

> 30 years ago

```
(b / before
   :op1 (n / now)
   :quant (m / more-than
             :op1 (t / temporal-quantity
                     :unit (y / year)
                     :quant 30)))
```

> more than 30 years ago

Disjunctions go high:

```
(o / or
   :op1 (t / temporal-quantity
           :unit (y / year)
           :quant 3)
   :op2 (t2 / temporal-quantity
            :unit (y2 / year)
            :quant 4))
```

> three or four years

```
(o / or
   :op1 (t / temporal-quantity
           :unit (m / month)
           :quant 6)
   :op2 (t2 / temporal-quantity
            :unit (y / year)
            :quant 1))
```

> six months or a year

Relative positions often include a quantity:

```
(c / crash-01
   :ARG1 (p / plane)
   :location (r / relative-position
                :op1 (g / city :name (n / name :op1 "Moscow"))
                :quant (d / distance-quantity
                          :unit (m / mile)
                          :quant 50)
                :direction (e / east)))
```

> The plane crashed 50 miles east of Moscow.
>
> The plane crash occurred 50 miles east of Moscow.

The X-quantity notation is only used for precise quantities. Vague quantities still use the `:quant` role:

```
(g / gather-01
   :ARG0 (p / person
            :quant (n / number
                      :mod (l / large))))
```

> A large number of people gathered.

Occasionally, the measurement itself is the primary concept:

```
(i / increase-01
   :ARG1 (n / number
            :quant-of (p / person)))
```

> The number of people increased.

Quantity types include: `monetary-quantity`, `distance-quantity`, `area-quantity`, `volume-quantity`, `temporal-quantity`, `frequency-quantity`, `speed-quantity`, `acceleration-quantity`, `mass-quantity`, `force-quantity`, `pressure-quantity`, `energy-quantity`, `power-quantity`, `voltage-quantity` (zap!), `charge-quantity`, `potential-quantity`, `resistance-quantity`, `inductance-quantity`, `magnetic-field-quantity`, `magnetic-flux-quantity`, `radiation-quantity`, `concentration-quantity`, `temperature-quantity`, `score-quantity`, `fuel-consumption-quantity`, `seismic-quantity`.

```
(q / monetary-quantity
   :quant 20
   :unit (d / dollar
            :mod (e / country
                    :name (n / name :op1 "Canada"))))
```

> C$20
>
> 20 Canadian dollars

Quantities where a `:quant 0` value does not represent a 0-quantity use `:scale` rather than `:unit`:

```
(q / seismic-quantity
   :quant 7.9
   :scale (r / richter))
```

> 7.9 on the Richter scale

Mathematical operators

The special concepts `product-of` and `sum-of` support the mathematical operators that the meaning of a text might include.

```
(r / reach-01
   :ARG0 (v / velocity
            :poss (a / aircraft))
   :ARG1 (p / product-of
            :op1 3
            :op2 (s / speed
                    :poss (s2 / sound))))
```

> The aircraft's velocity reached three times the speed of sound.

```
(f / finish-01
   :ARG0 (p / person :name (n / name :op1 "Patrick" :op2 "Makau"))
   :ARG1 (r / run-02
            :ARG0 p
            :ARG1 (m / marathon)
            :duration (s2 / sum-of
                        :op1 (t2 / temporal-quantity :quant 2
                                 :unit (h / hour))
                        :op2 (t3 / temporal-quantity :quant 3
                                 :unit (m2 / minute))
                        :op3 (t4 / temporal-quantity :quant 38
                                 :unit (s3 / second)))))
```

> Patrick Makau finished the marathon in 2 hours, 3 minutes and 38 seconds.

Other entities: dates, times, percentages, phone, email, URLs

These entities are described in standard, canonical forms:

```
 (d / date-entity
    :year 2012
    :month 2
    :day 29)
```

> February 29, 2012
>
> 29 February 2012
>
> 2/29/2012

```
(d / date-entity
   :year 2012)
```

> 2012
>
> the year 2012

```
(d / date-entity
   :month 4)
```

> April

```
(d / date-entity
   :weekday (f / friday))
```

> Friday

```
(d / date-entity
```

```
   :year 2012
   :month 2)
```

> February, 2012

```
(d / date-entity
   :month 2
   :day 29
   :weekday (w / wednesday))
```

> Wednesday, February 29

```
(d / date-entity
   :day 29)
```

> the 29th

```
(d / date-entity
   :month 2
   :day 29
   :weekday (w / wednesday)
   :time 16:30
   :timezone (z / PST))
```

> Wednesday, February 29, 16:30 PST

```
(d / date-entity
   :time 16:30)
```

> 16:30
>
> 4:30pm
>
> 4:30 in the afternoon
>
> half past four

```
(d / date-entity
   :era (h / heisei)
   :year 24
   :month 2
   :day 29
   :calendar (j / country :name (n / name :op1 "Japan")))
```

> February 29, 24th year of Heisei era

```
 (d / date-entity
    :year 2011
    :quarter 4)
```

> 4th quarter, 2011
>
> 2011Q4

```
(d / date-entity
   :year 2011
   :season (s / summer))
```

> Summer 2011

```
(d / date-entity
   :year 2011
   :year2 2012
   :season (w / winter))
```

> Winter 2011-2012

```
 (d / date-entity
    :year 2011
    :year2 2012
    :calendar (y / year
                 :mod (a / academia)))
```

> academic year 2011-2012

```
 (d / date-entity
    :year 2012
    :calendar (y / year
                 :mod (f / finance)
                 :mod (g / government-organization
                         :ARG0-of (g2 / govern-01
                                      :ARG1 (c / country
                                               :name (n / name
                                                        :op1 "United"
                                                        :op2 "States"))))))
```

> United States government fiscal year 2012

```
(d / date-interval
   :op1 (d2 / date-entity :year 2012 :month 3 :day 8)
   :op2 (d3 / date-entity :year 2012 :month 3 :day 9))
```

> March 8-9, 2012

```
(d / date-interval
   :op1 (d2 / date-entity :year 1939 :month 9 :day 1)
   :op2 (d3 / date-entity :year 1945 :month 5 :day 8))
```

> Sept. 1, 1939 - May 8, 1945

```
(p / percentage-entity :value 25)
```

> 25%
>
> twenty-five percent

> 25 percent

```
(p / phone-number-entity :value "18005551212")
```

> 1-800-555-1212
>
> 1 (800) 555-1212

```
(e / email-address-entity :value "president@whitehouse.gov")
```

> president@whitehouse.gov

```
(u / url-entity :value "www.whitehouse.gov")
```

> www.whitehouse.gov

AMR Freak Show

This section is optional reading. Just some mathematical curiosities of AMR that one bumps into eventually, of interest to mathematicians and children. First, the occasional AMR will have a cycle:

```
(w / woman
   :ARG0-of (n / nominate-01
               :ARG1 (b / boss
                        :poss w)))
```

> the woman who nominated her boss

Note how `w` refers to “the woman who nominated the boss of the woman who nominated the boss of the woman who nominated the boss of …”

Second, we have two different ways of encoding the same propositional content (“the boy likes to be believed”):

```
(l / like-01                    (l / like-01
   :ARG0 (b / boy)                 :ARG0 (b / boy
   :ARG1 (b2 / believe-01                  :ARG1-of (b2 / believe-01))
            :ARG1 b))              :ARG1 b2)
```

Sensible people will prefer the version on the left, though both versions relate the same conjunction of propositional triples.