AMR dependency parsing with a typed semantic algebra

Jonas Groschwitz*^, Matthias Lindemann*,
Meaghan Fowlie*, Mark Johnson^, Alexander Koller*

*Saarland University ^Macquarie University

ACL 2018
Melbourne, Australia
July 17

Matthias Lindemann
Saarland University

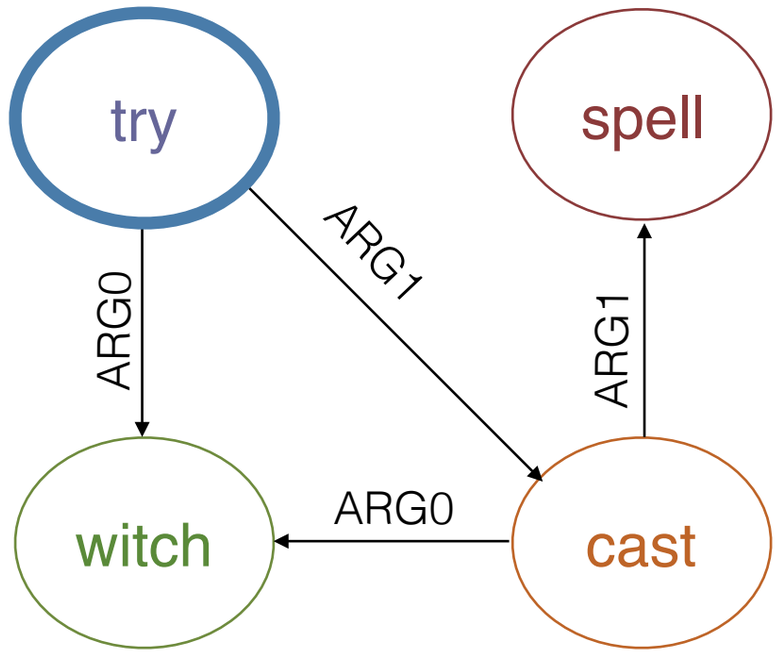
Meaghan Fowlie
Saarland University

Mark Johnson
Macquarie University

Alexander Koller
Saarland University

Abstract Meaning Representation (AMR)

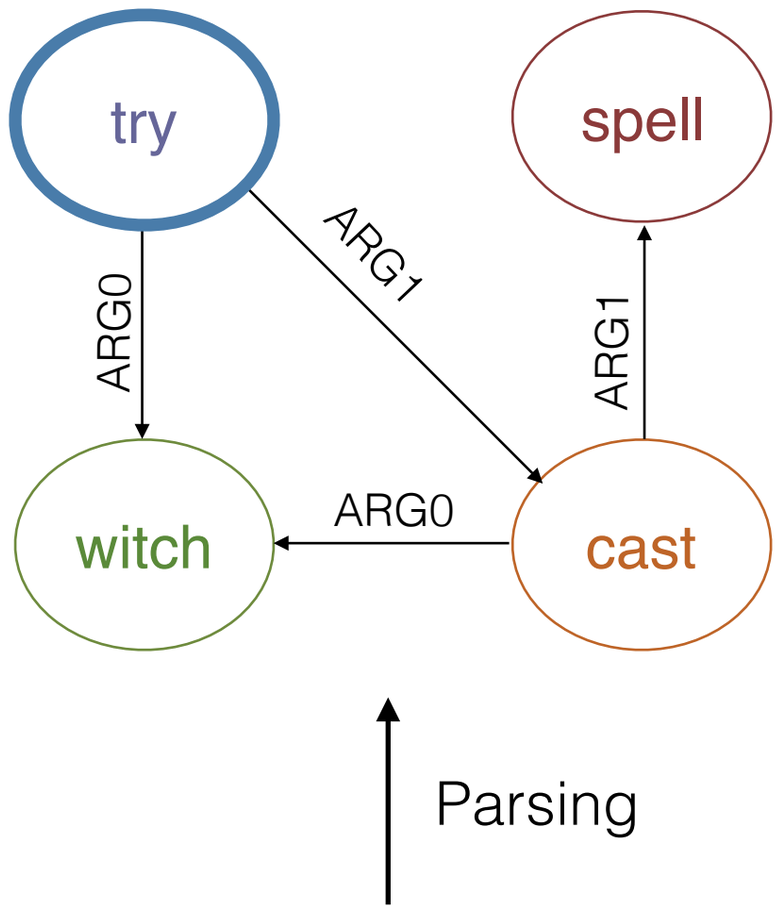
Banarescu et al. 2013



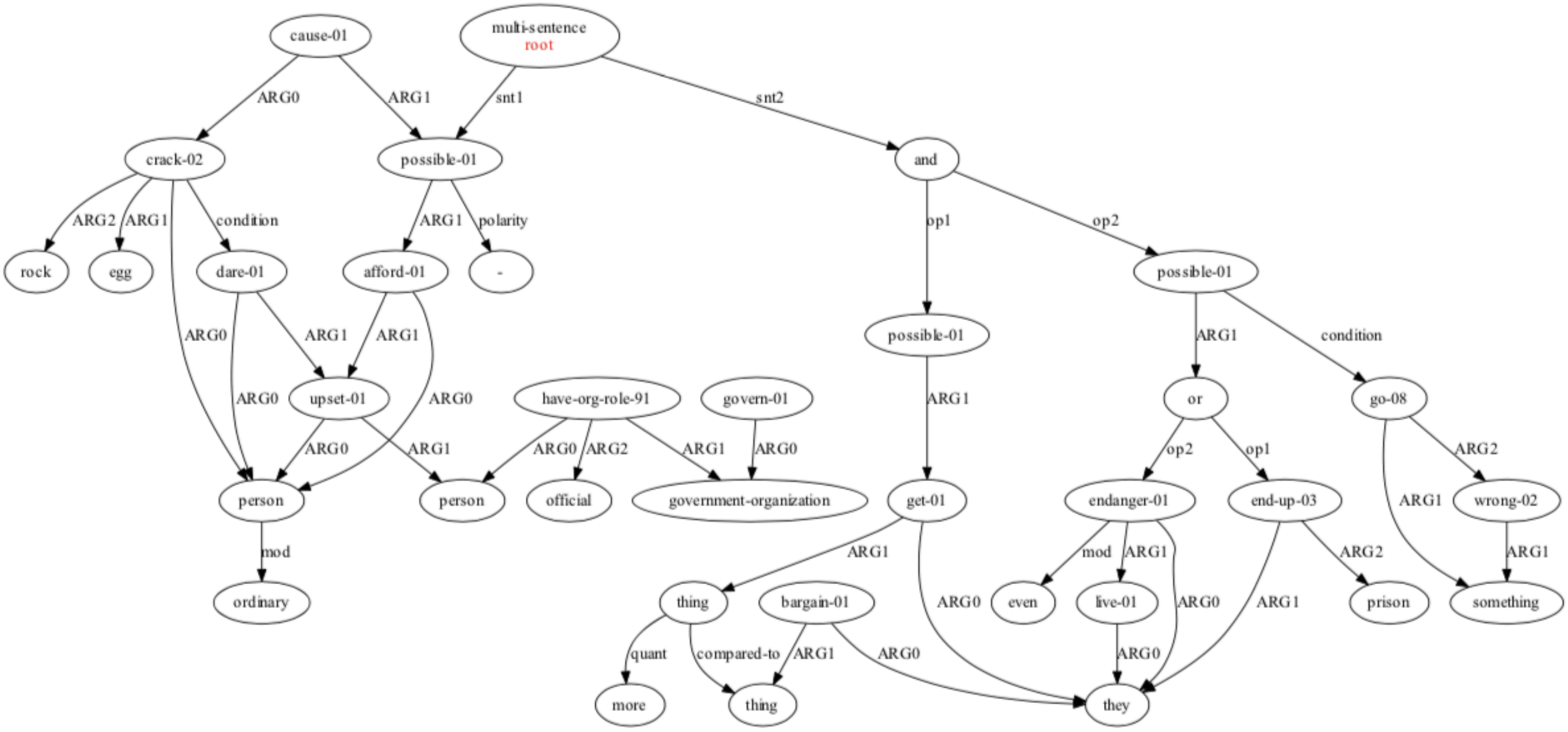
The witch tried to cast a spell

Abstract Meaning Representation (AMR)

Banarescu et al. 2013



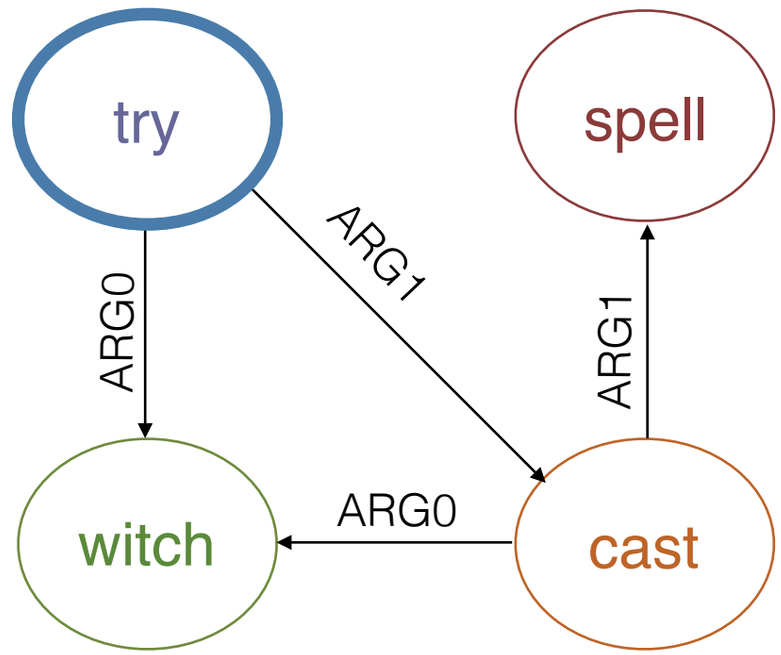
The witch tried to cast a spell


cause-01
multi-sentence
root
ARG0
ARG1
snt1
snt2
crack-02
possible-01
and
ARG2
ARG1
condition
ARG1
polarity
op1
op2
rock
egg
dare-01
afford-01
-
possible-01
ARG0
ARG1
ARG1
ARG1
condition
ARG0
ARG0
upset-01
ARG0
have-org-role-91
govern-01
ARG1
or
go-08
ARG0
ARG1
ARG0
ARG2
ARG1
ARG0
op2
op1
ARG2
person
person
official
government-organization
get-01
endanger-01
end-up-03
ARG1
wrong-02
mod
ARG1
ARG2
ARG1
mod
ordinary
thing
bargain-01
ARG0
even
live-01
ARG0
ARG1
prison
something
quant
compared-to
ARG1
ARG0
ARG0
more
thing
they

Classic AMR parser (e.g. JAMR 2014)

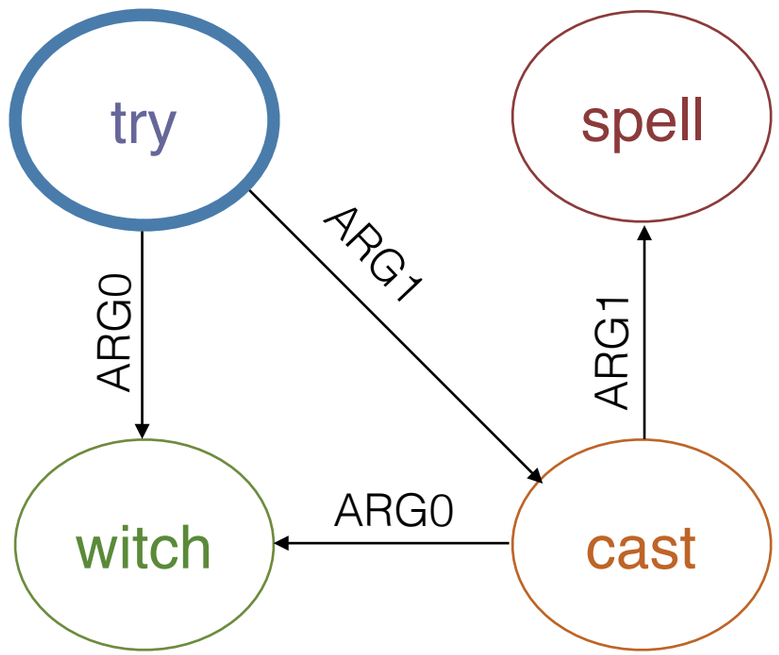
Step 1: Predict nodes

Step 2: Predict edges

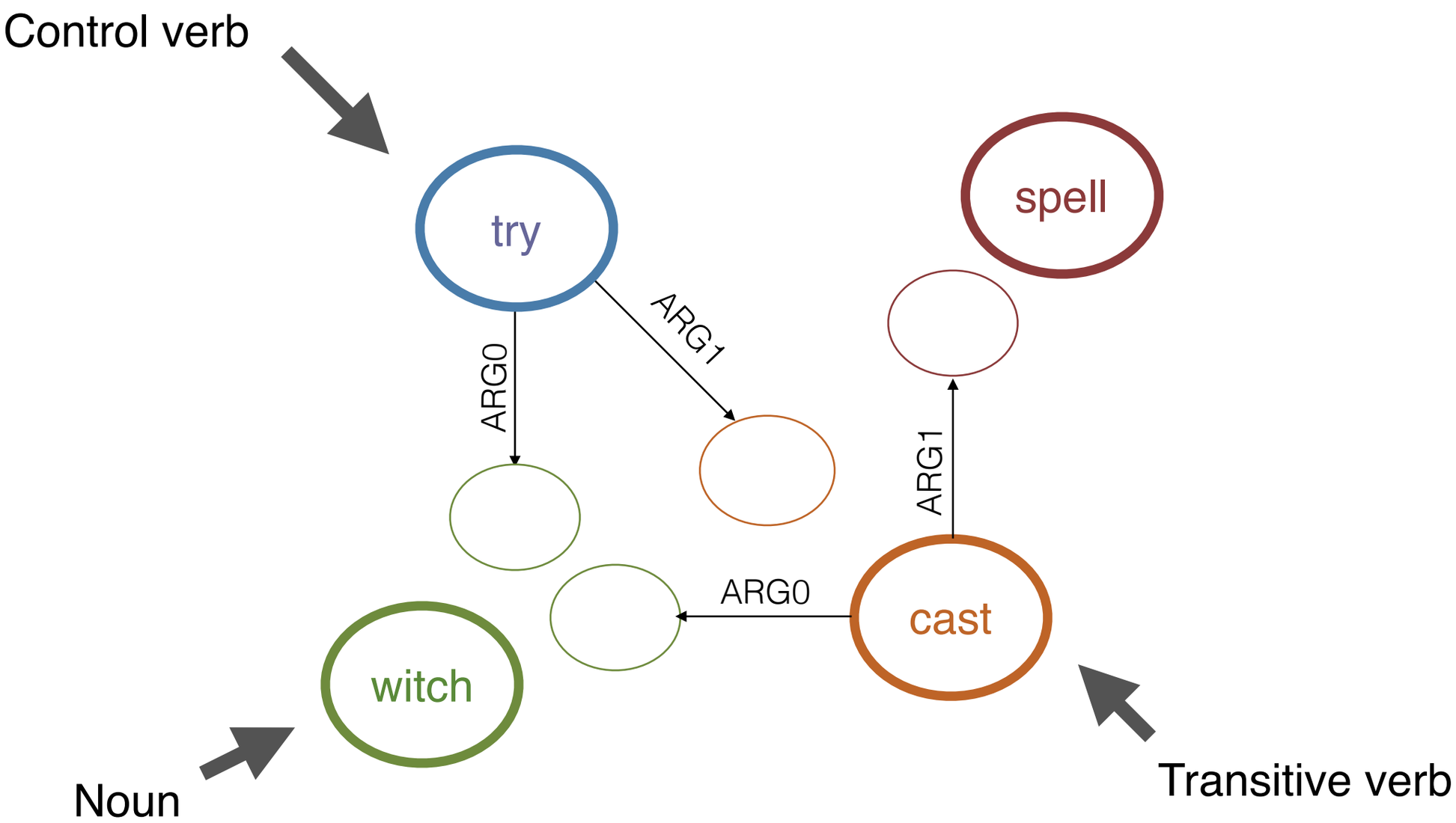


The witch tried to cast a spell

Not just nodes and edges

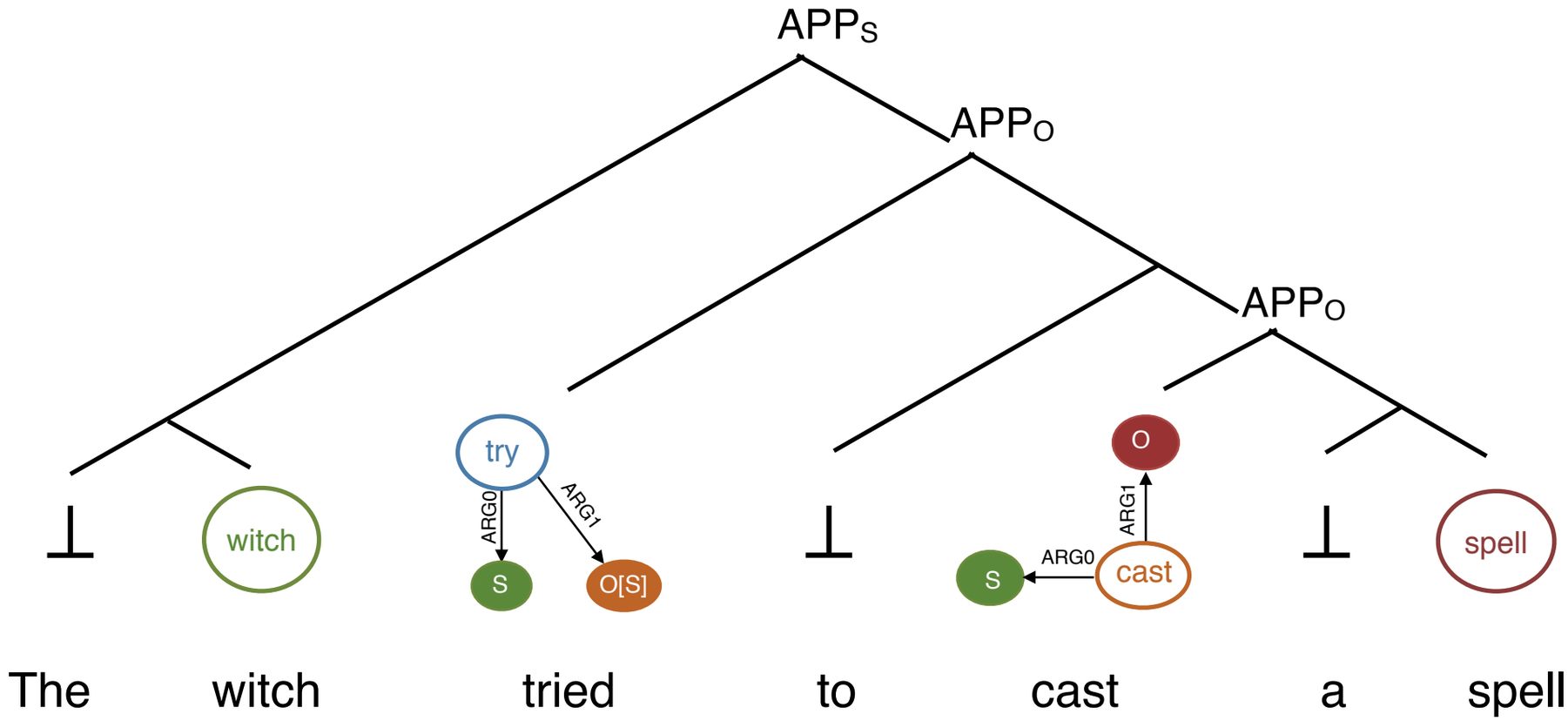


Not just nodes and edges



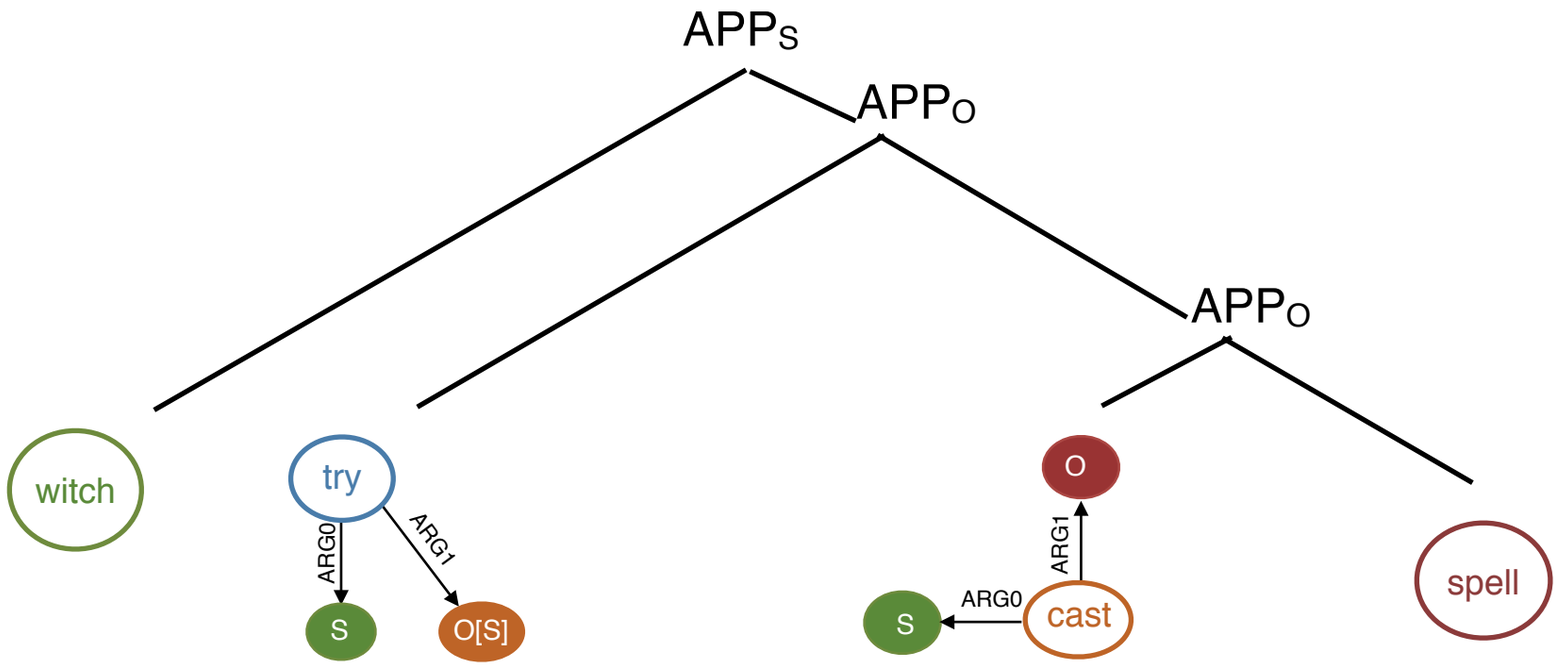
Hidden compositional structure

Principle of compositionality: the **meaning of a complex expression** is determined by the meanings of its **constituent expressions** and the **rules used to combine** them.



Hidden compositional structure

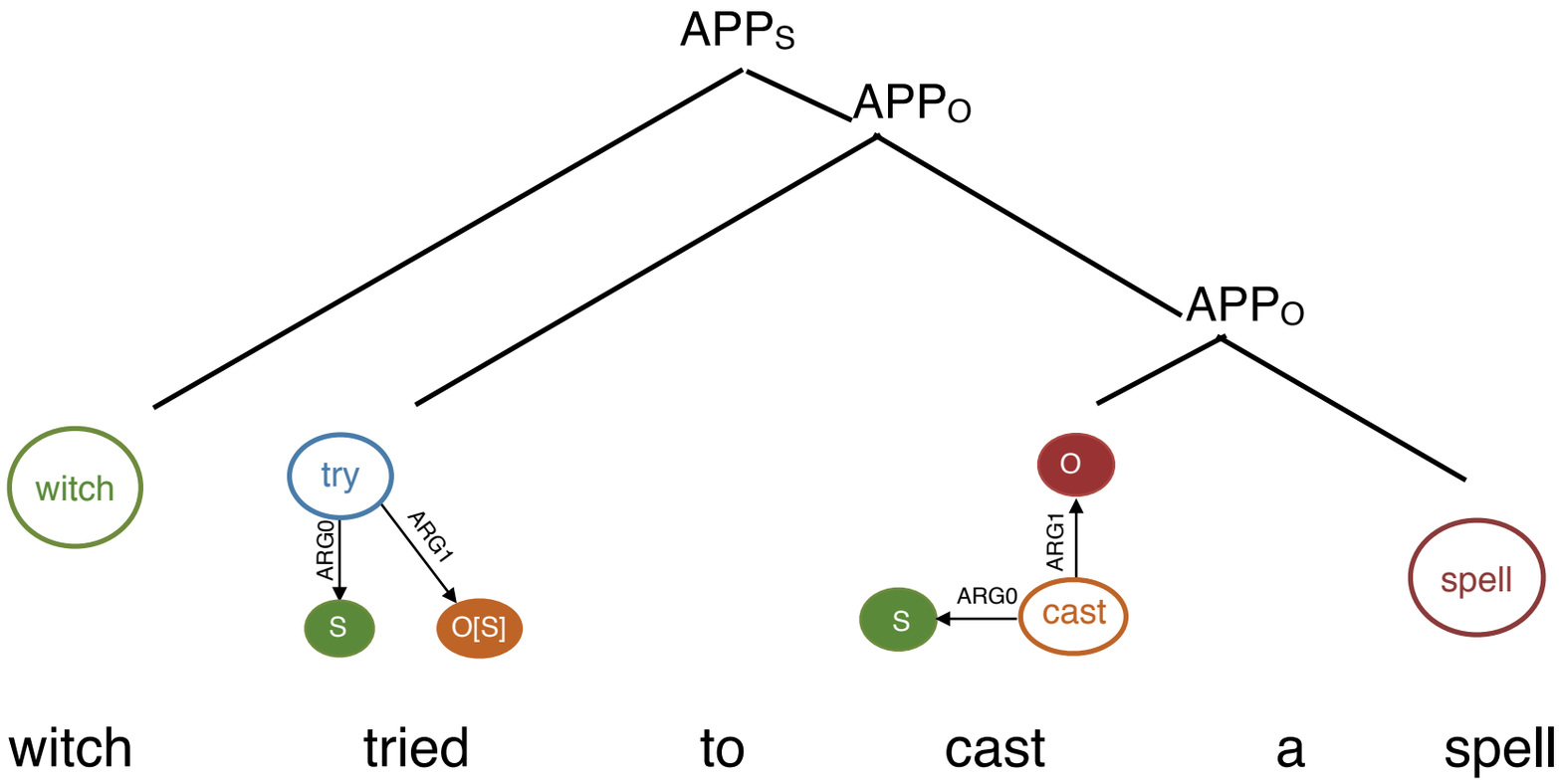
Principle of compositionality: the **meaning of a complex expression** is determined by the meanings of its **constituent expressions** and the **rules used to combine** them.



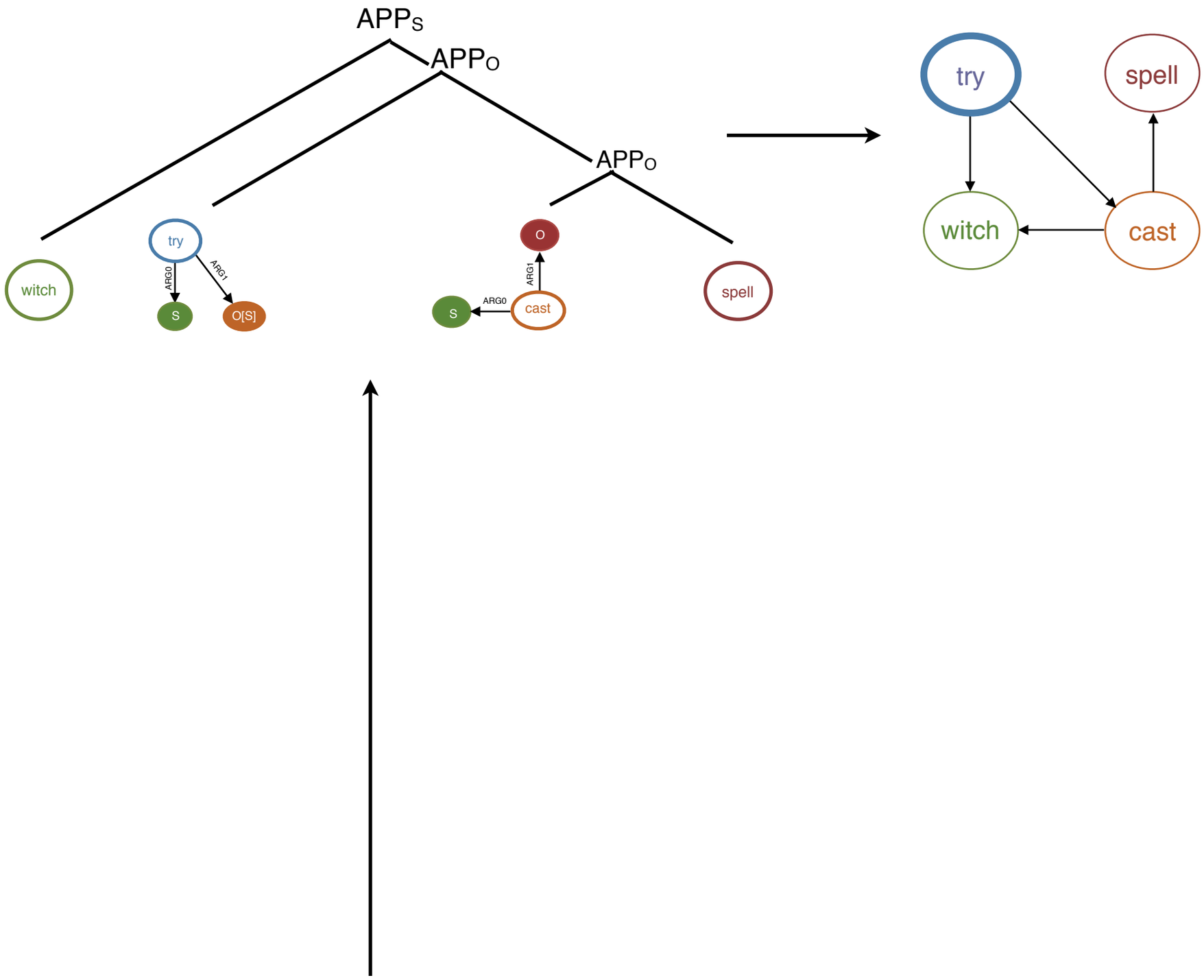
The witch tried to cast a spell

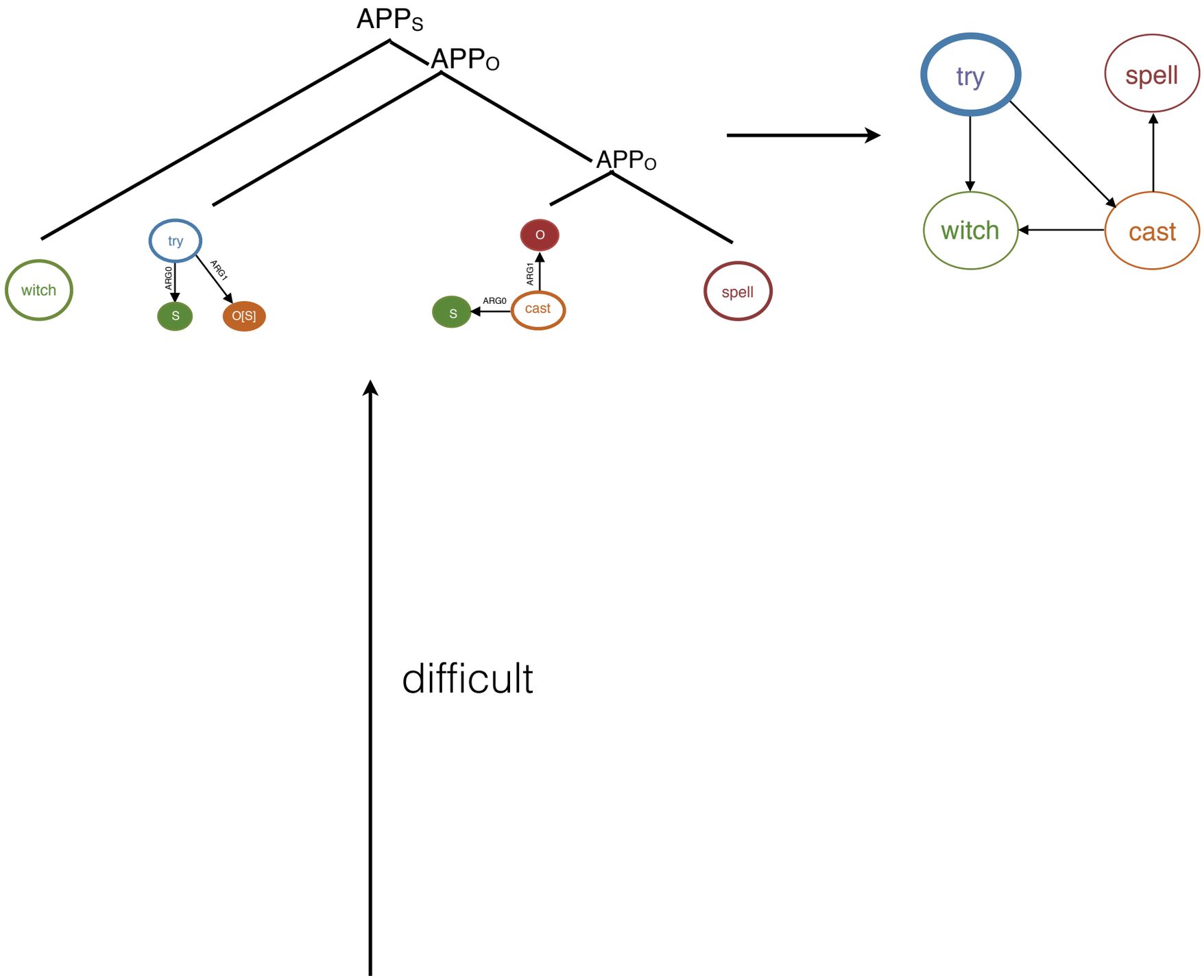
Hidden compositional structure

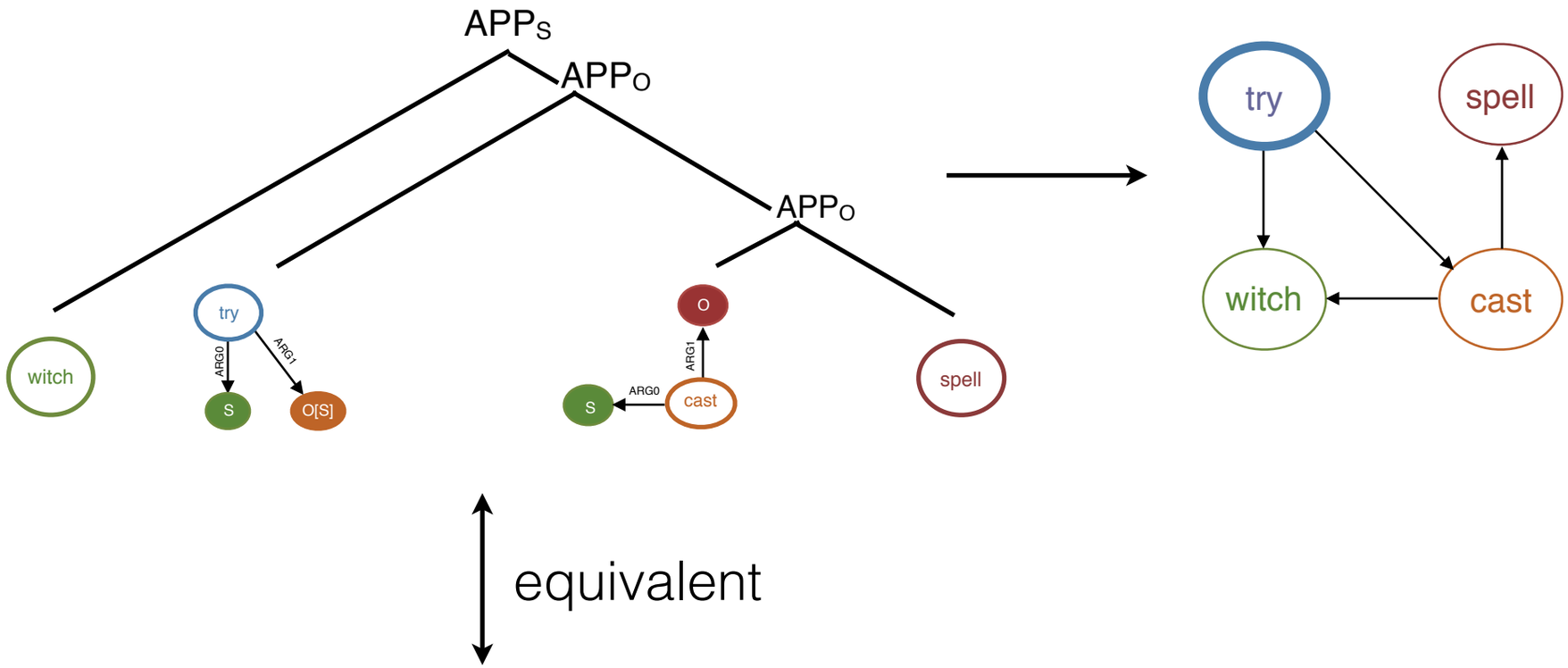
Principle of compositionality: the **meaning of a complex expression** is determined by the meanings of its **constituent expressions** and the **rules used to combine** them.

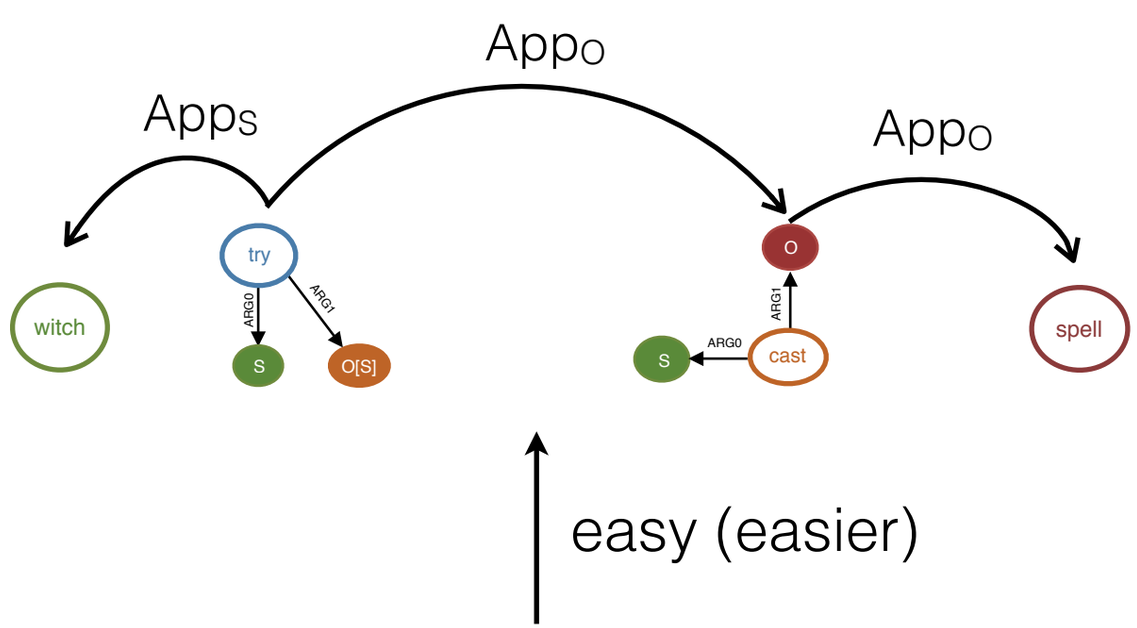


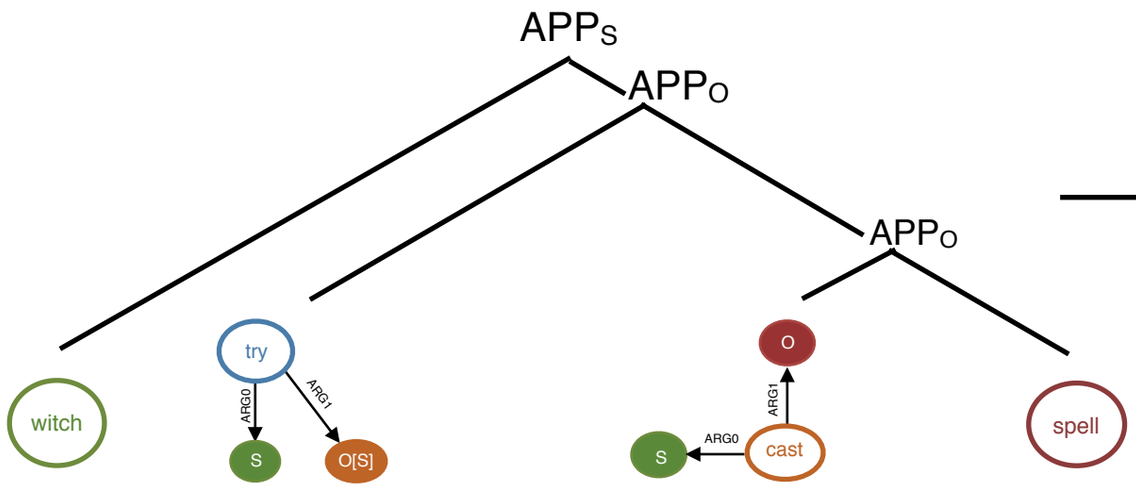
- Widely accepted in linguistics, long history (Frege 1800s)
- Use this knowledge to guide machine learning!

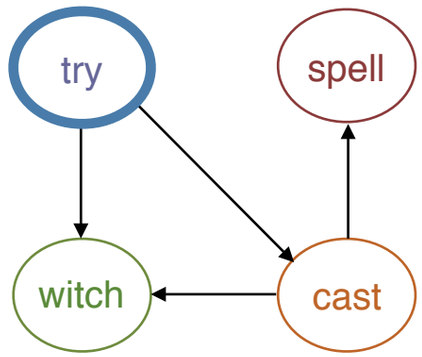

APPS
APPO
APPO
witch
try
ARG0
ARG1
S
O[S]
O
ARG1
S
ARG0
cast
spell
try
spell
witch
cast
The witch tried to cast a spell


APPS
APPO
APPO
witch
try
ARG0
ARG1
S
O[S]
O
ARG1
S
ARG0
cast
spell
try
spell
witch
cast
difficult
The witch tried to cast a spell

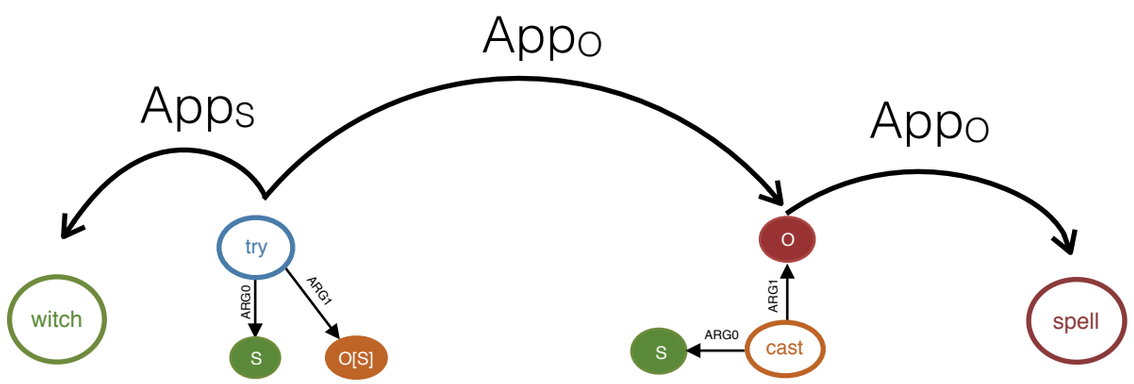

APPS
APPO
APPO
witch
try
ARG0
ARG1
S
O[S]
O
ARG1
S
ARG0
cast
spell
try
spell
witch
cast
equivalent


APPS
APPO
APPO
witch
try
ARG0
ARG1
S
O[S]
O
ARG1
S
ARG0
cast
spell
dependencies!
easy (easier)
The witch tried to cast a spell





equivalent



dependencies!

easy (easier)

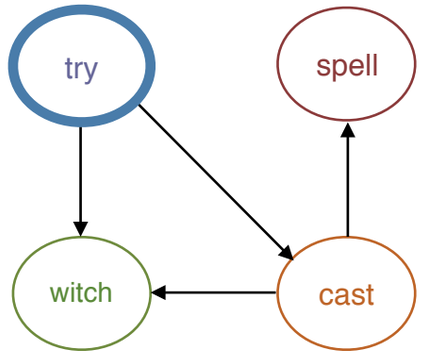
The witch tried to cast a spell

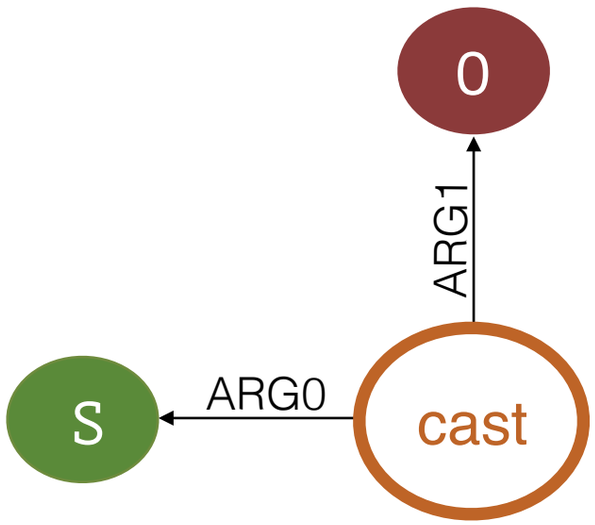
Part 1

Part 2

Apply-Modify (AM) Algebra

G. et al, IWCS 2017

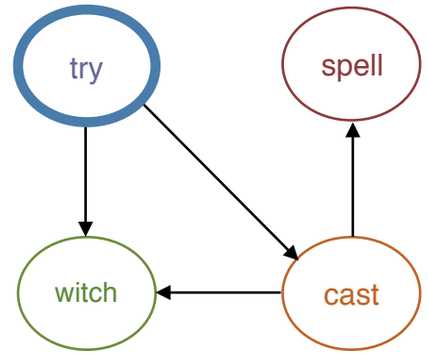




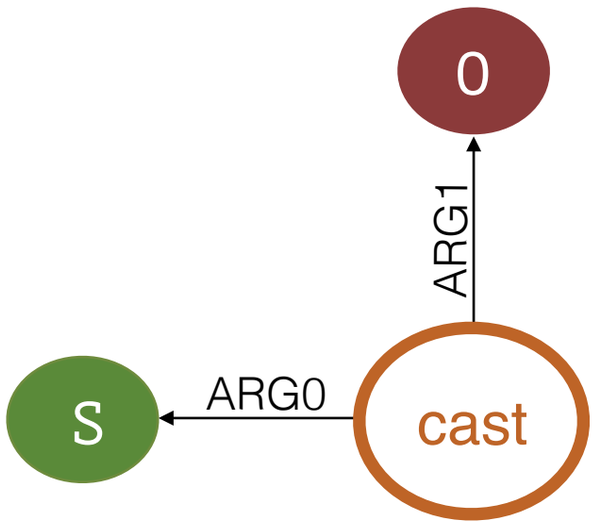
*HR algebra, Courcelle & Engelfriet 2012

Apply-Modify (AM) Algebra

G. et al, IWCS 2017



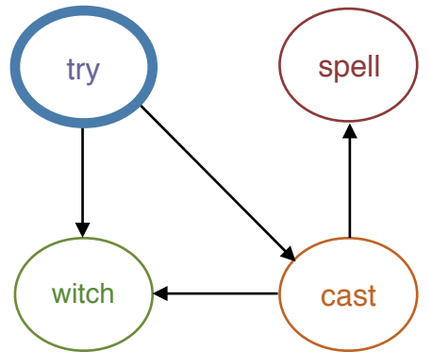
- Empty argument slots are labeled with sources* S,O,… (subject, object,…)



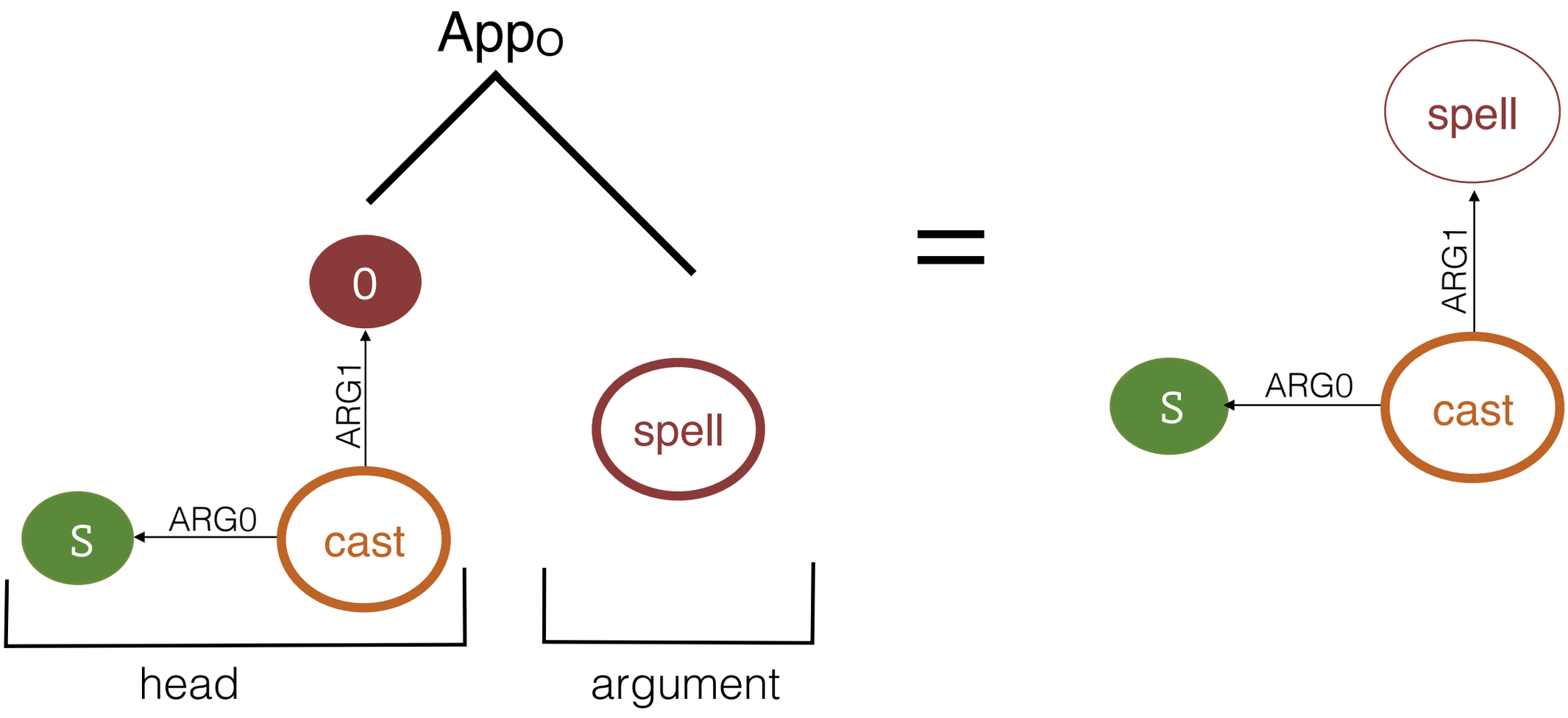
*HR algebra, Courcelle & Engelfriet 2012

Apply-Modify (AM) Algebra

G. et al, IWCS 2017

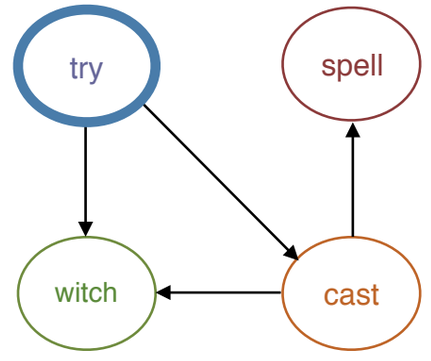


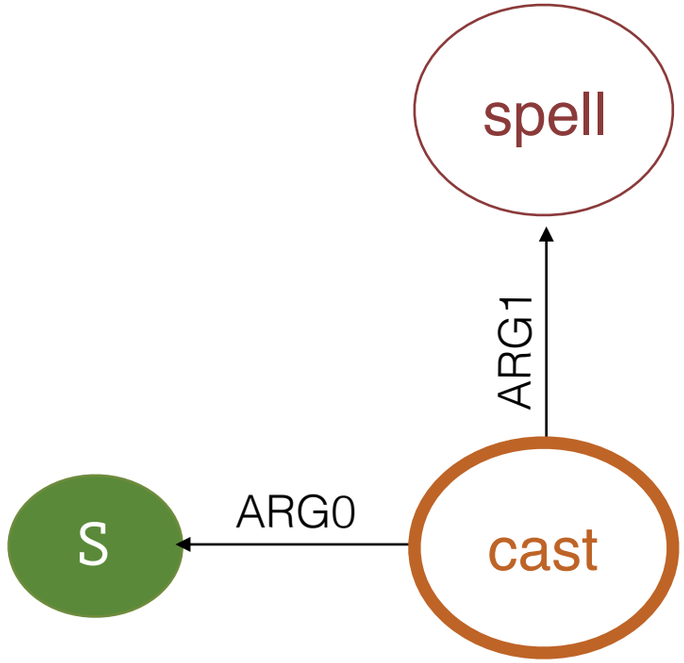
- Empty argument slots are labeled with sources* S,O,… (subject, object,…)
- Have ‘apply’ operation for each source, e.g. APP_O



*HR algebra, Courcelle & Engelfriet 2012

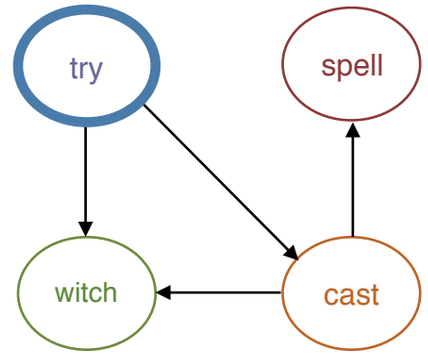
Typed AM Algebra

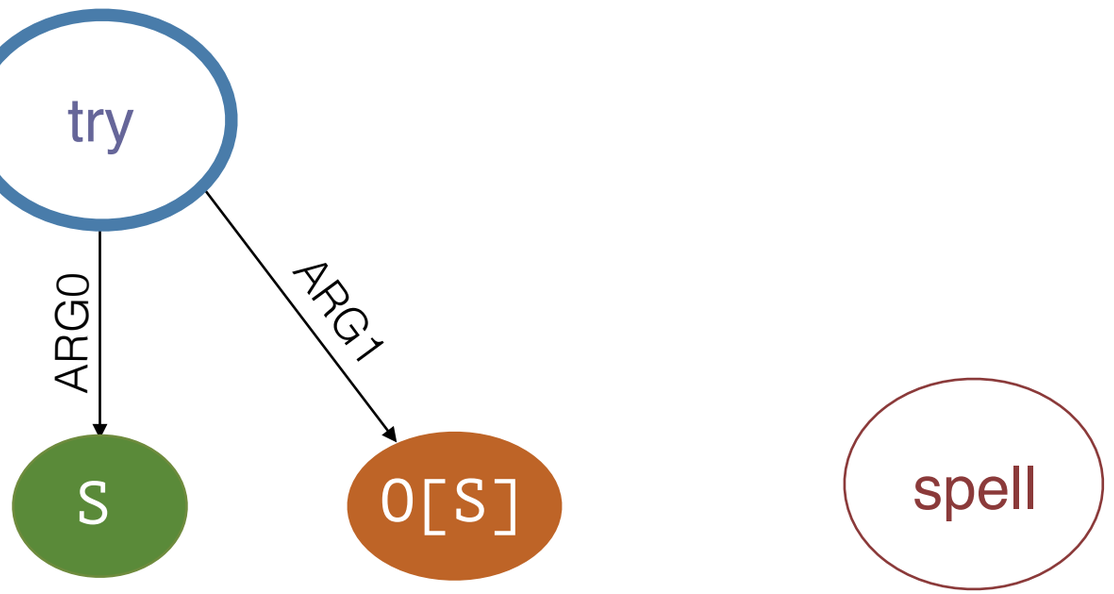




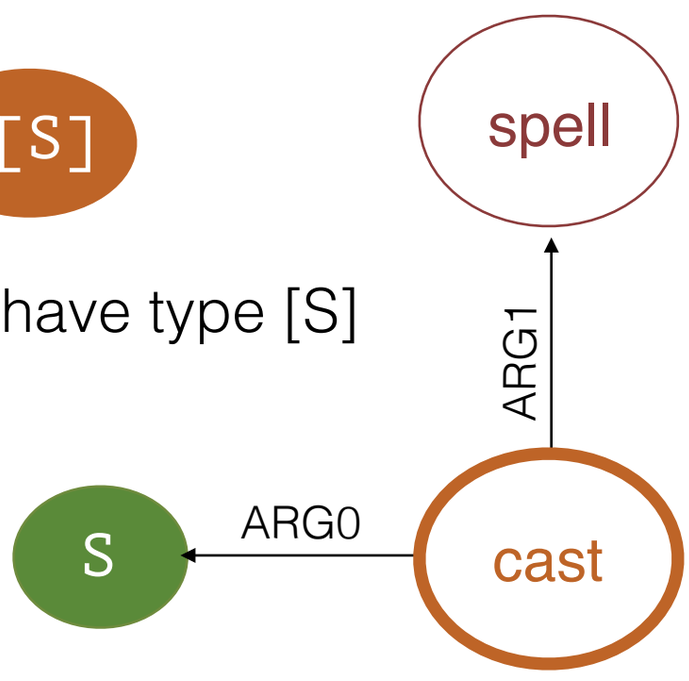
Has type [S]

Typed AM Algebra



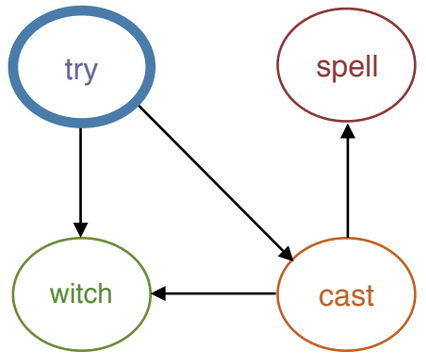


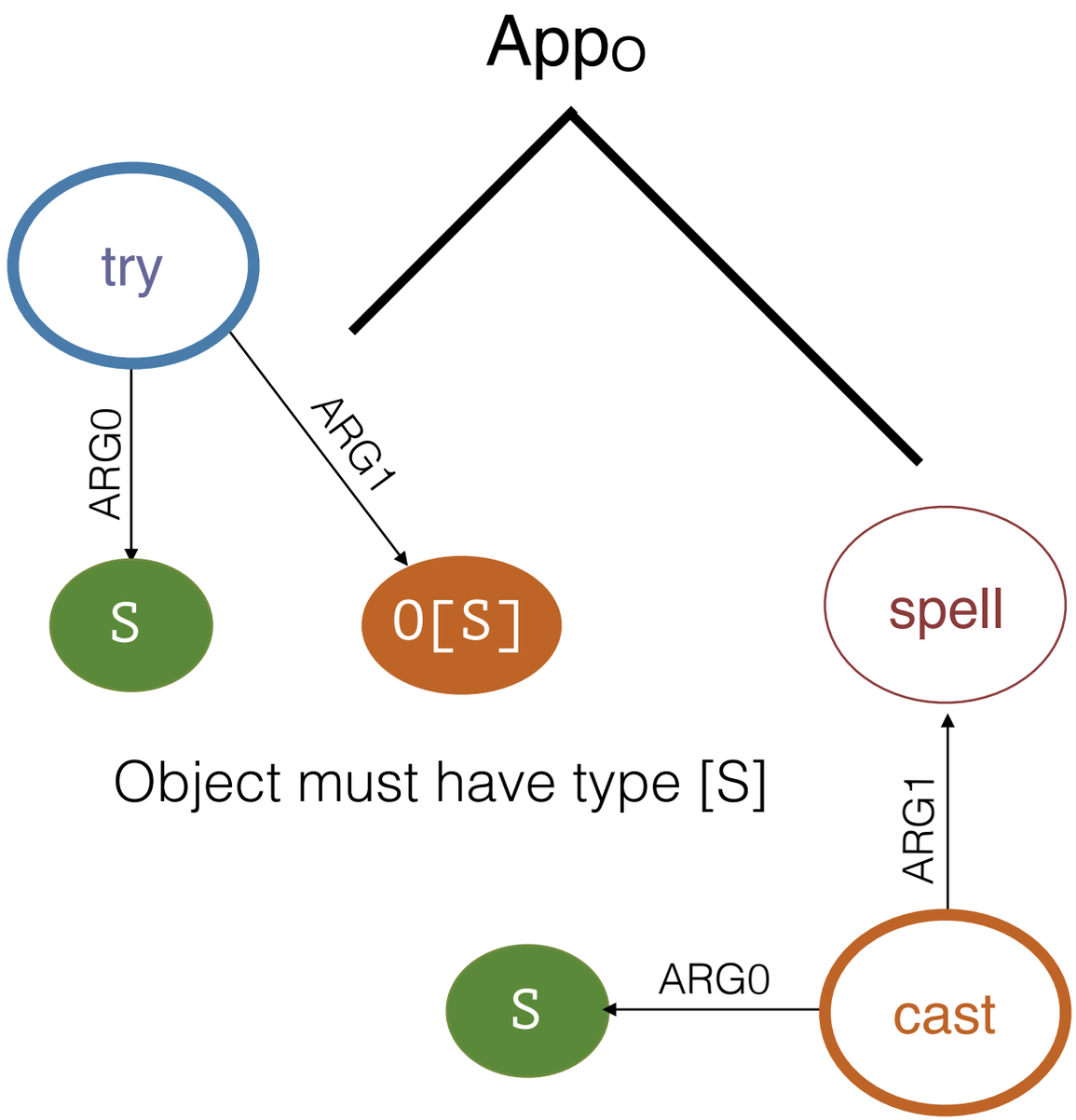
Object must have type [S]



Has type [S]

Typed AM Algebra

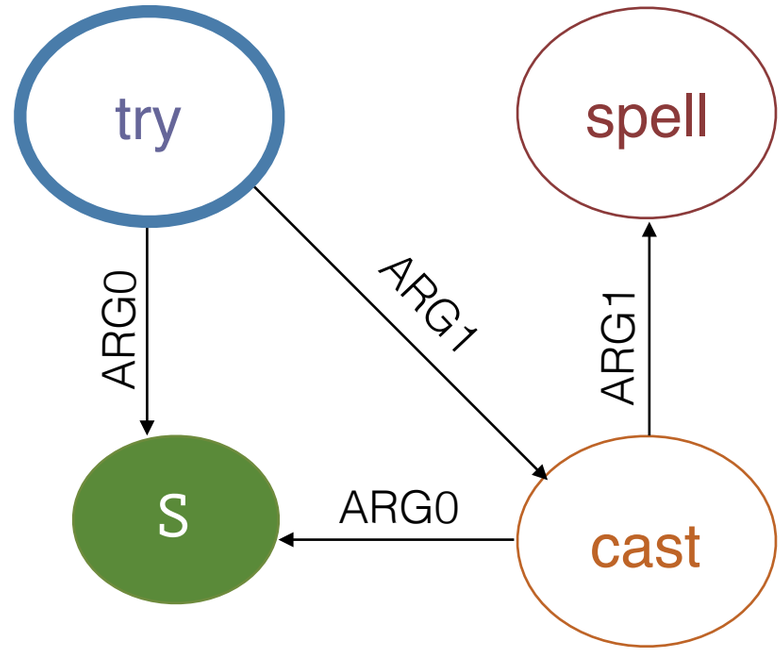




Object must have type [S]

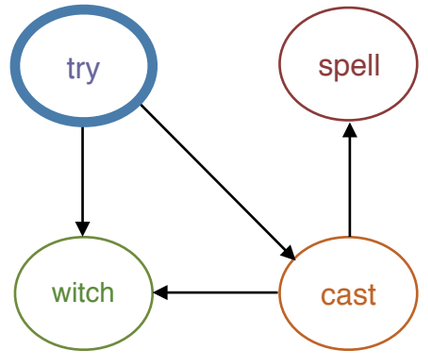
Has type [S]

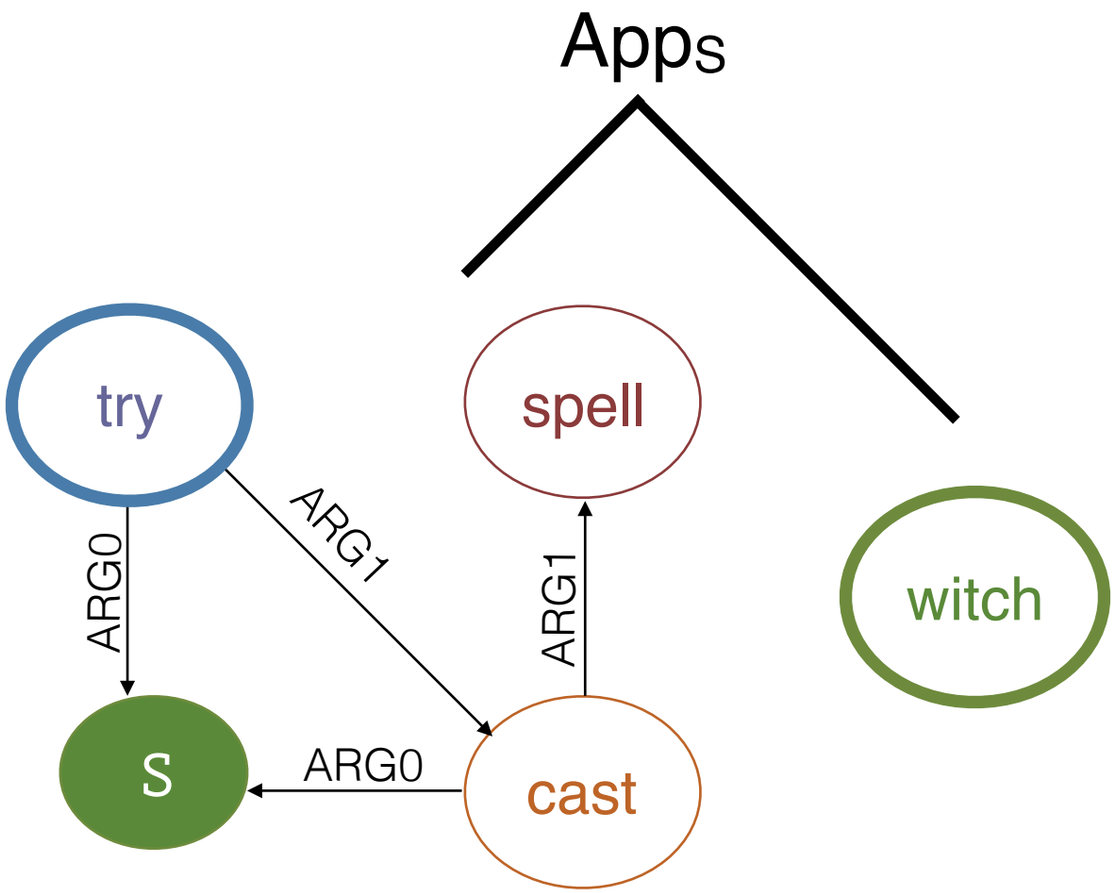
=



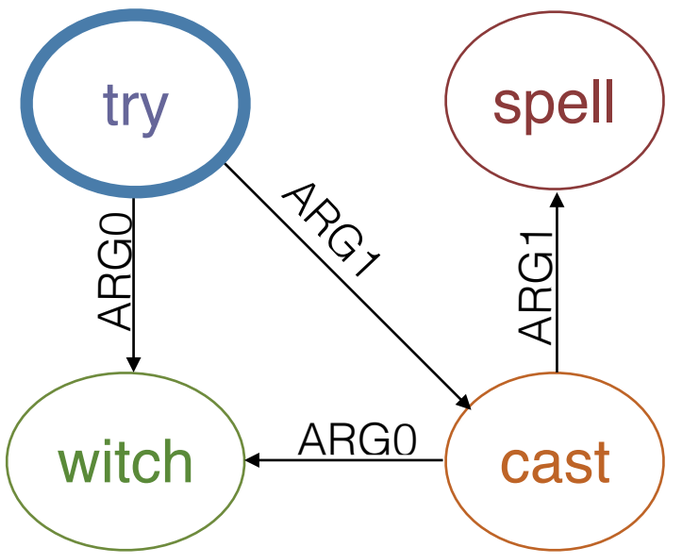
Matching sources automatically merge

Apply-Modify Algebra



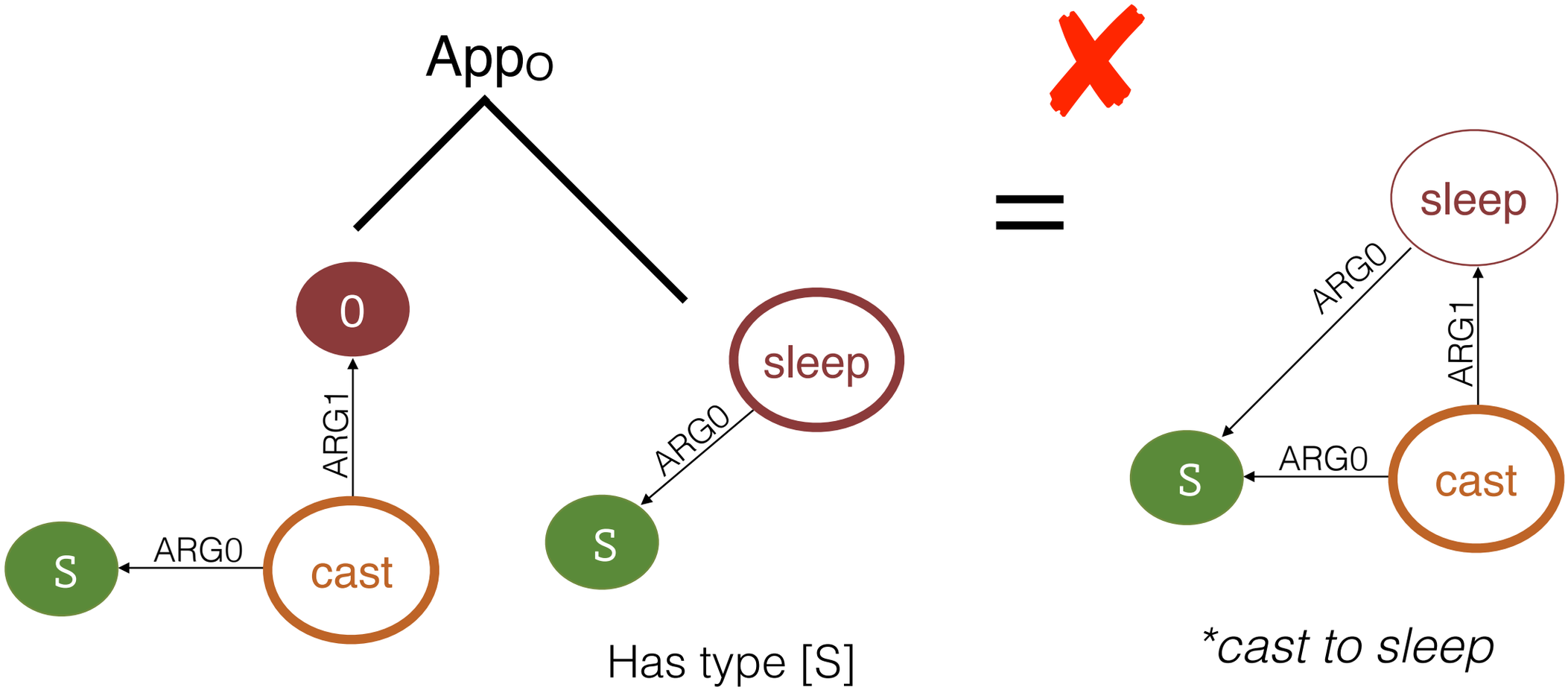


=



The witch tried to cast a spell

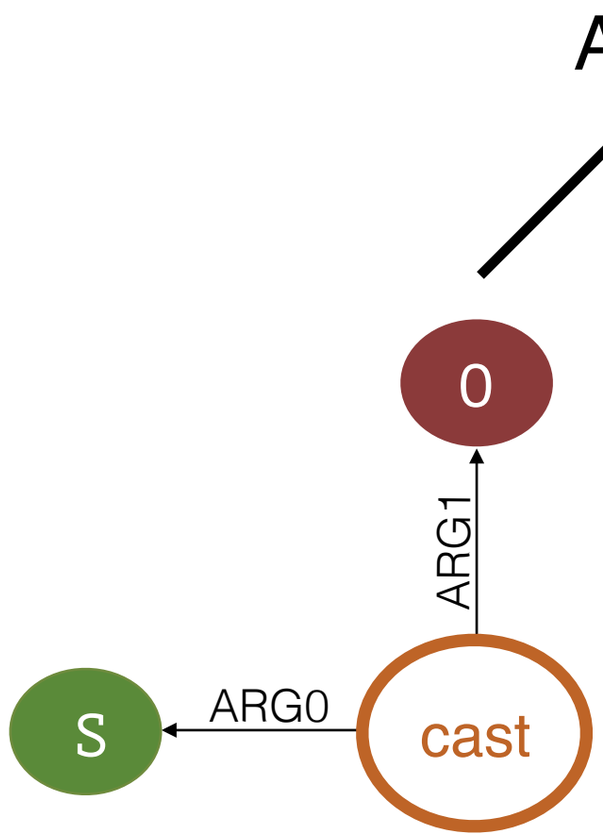
Types control reentrancies



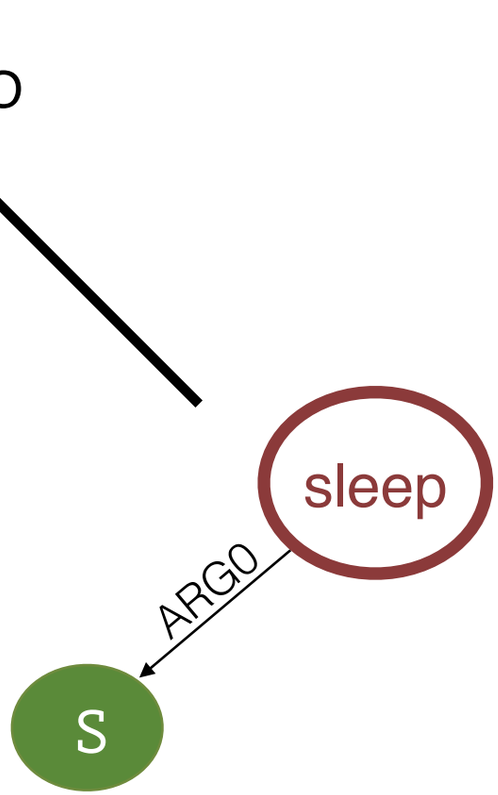
Has type [S]

**cast to sleep*

Types control reentrancies

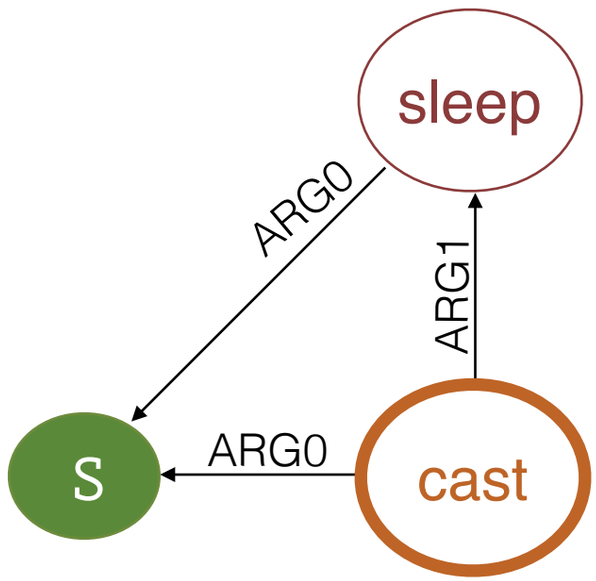


Object must have type [ ]



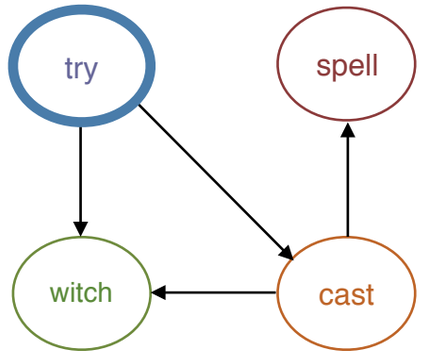
Has type [S]

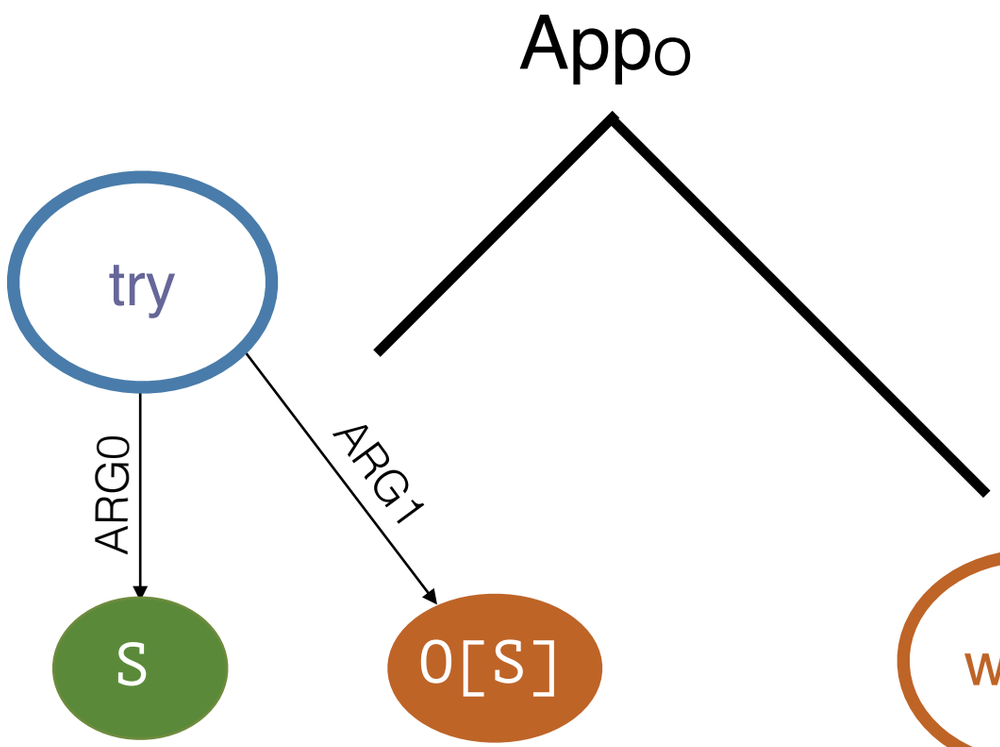
=



**cast to sleep*

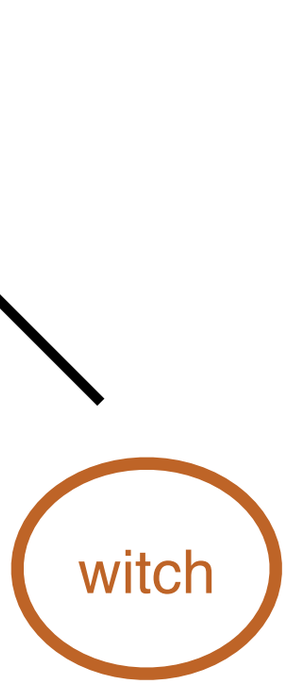
Types control reentrancies



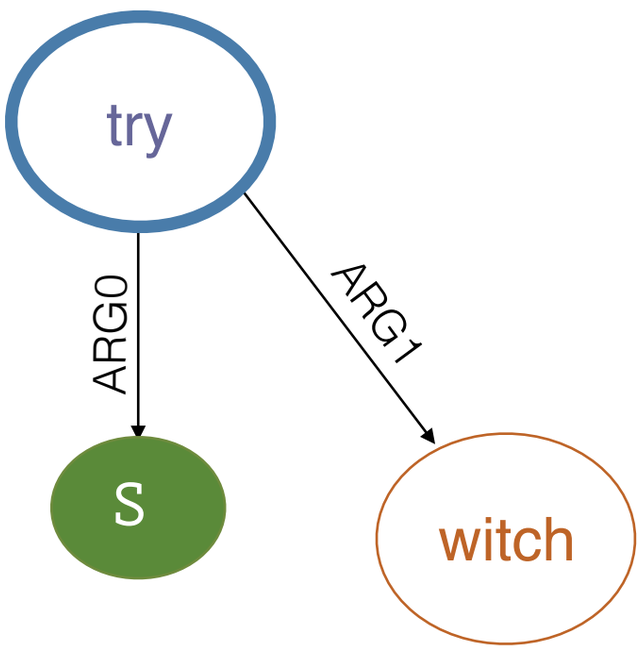


Object must have type [S]

=



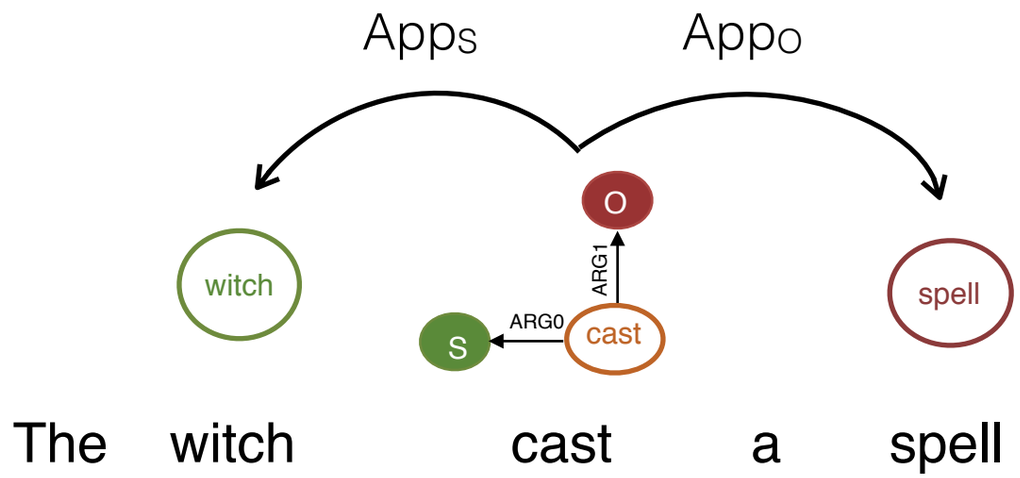
Has type [ ]



**tried to witch*

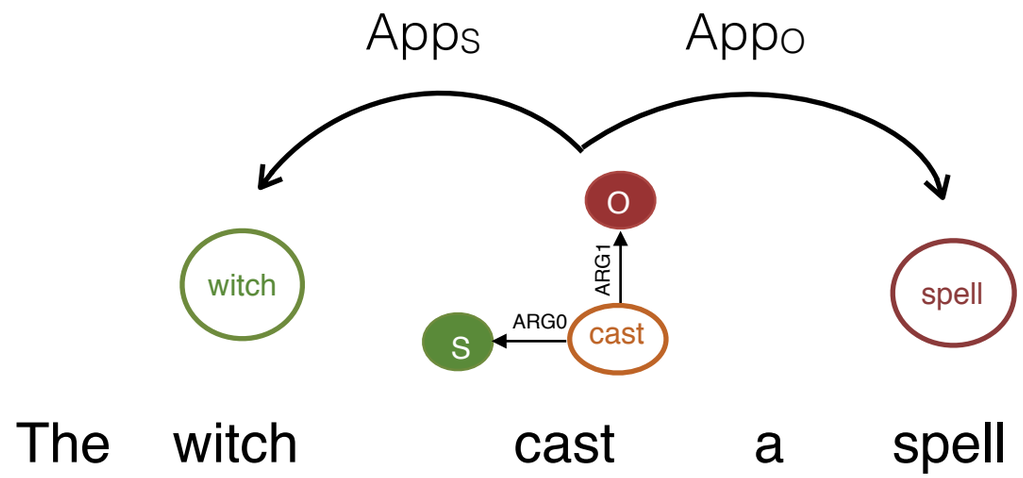
AM Dependency Trees

dependencies define operations, but not their order



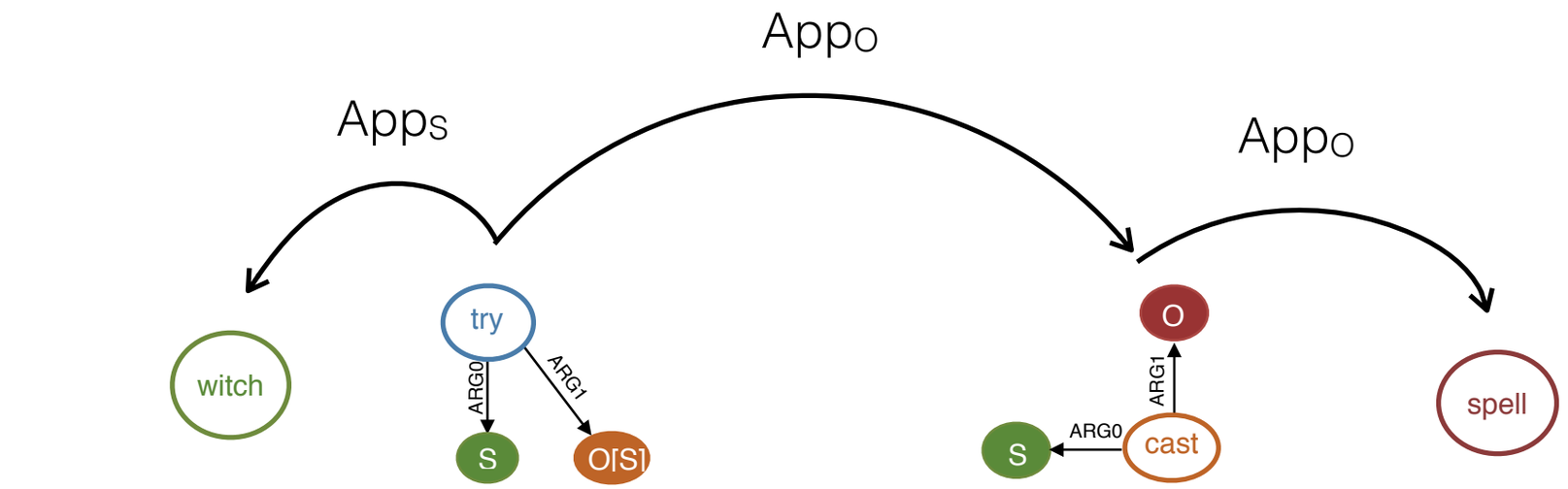
AM Dependency Trees

dependencies define operations, but not their order



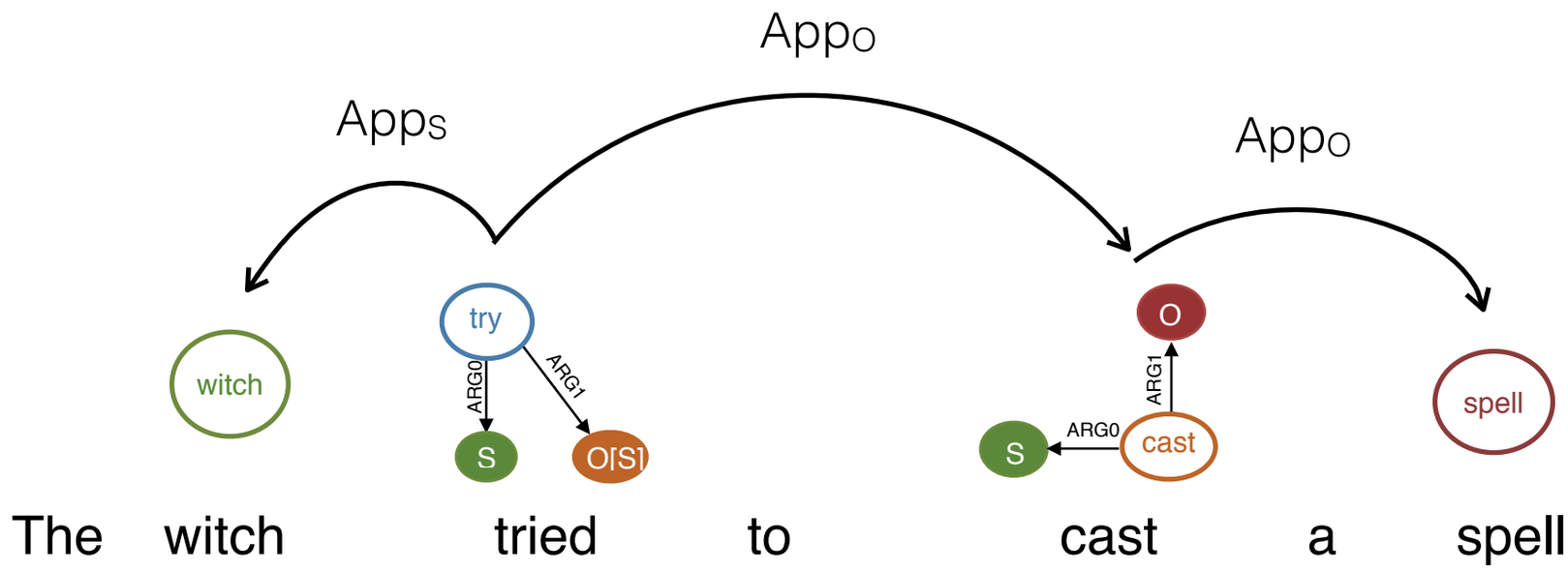
here: order does not matter

AM Dependency Trees



here: need APP_O before APP_S to get reentrancies

AM Dependency Trees

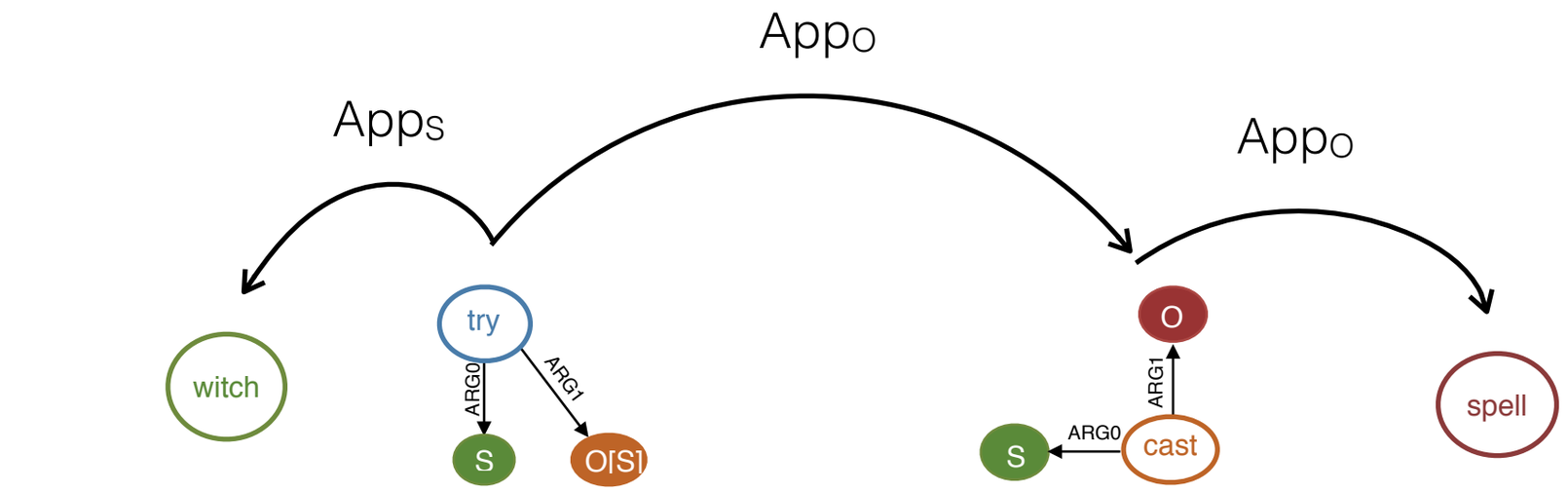


here: need APP_O before APP_S to get reentrancies

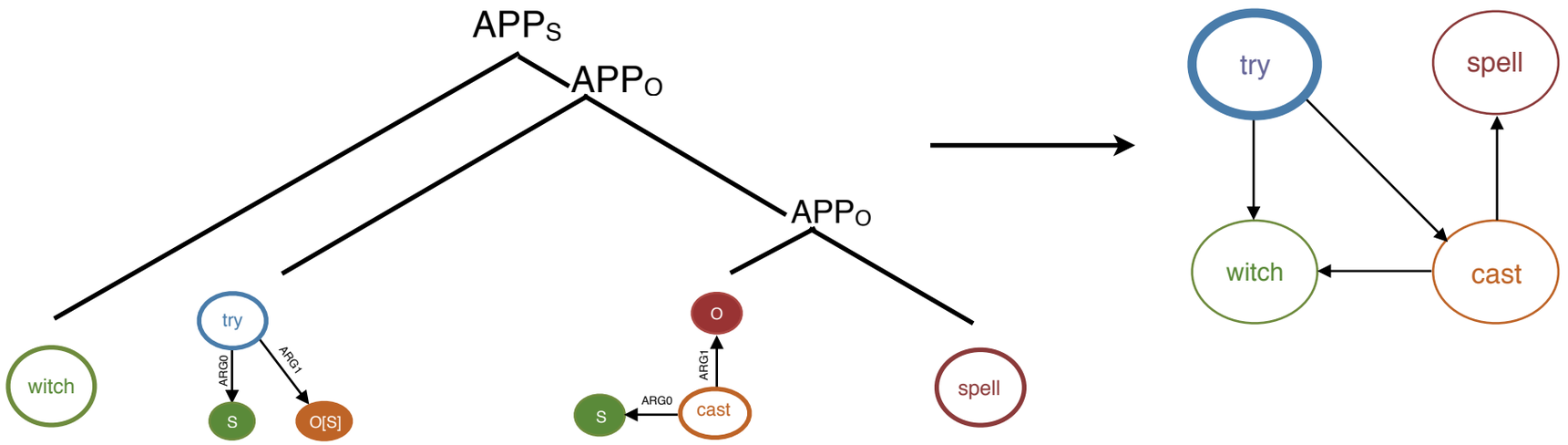
- Always need to resolve reentrancies first
- Types encode reentrencies

➡ use type system to determine operation order

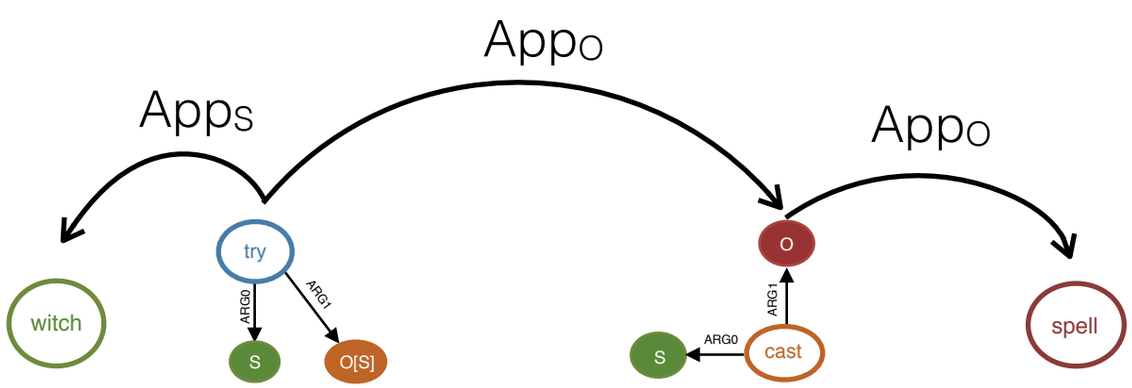
AM Dependency Trees



Building instructions for an AMR that we know how to predict



equivalent



easy (easier)

The witch tried to cast a spell

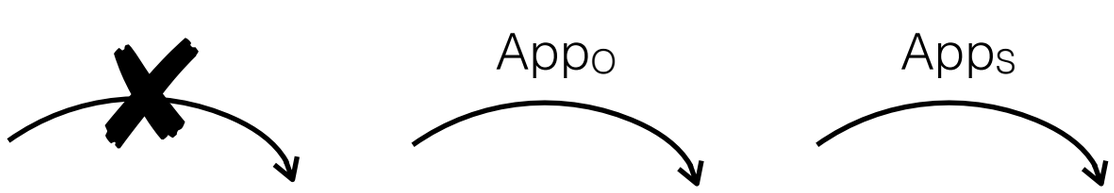
Part 1

Part 2

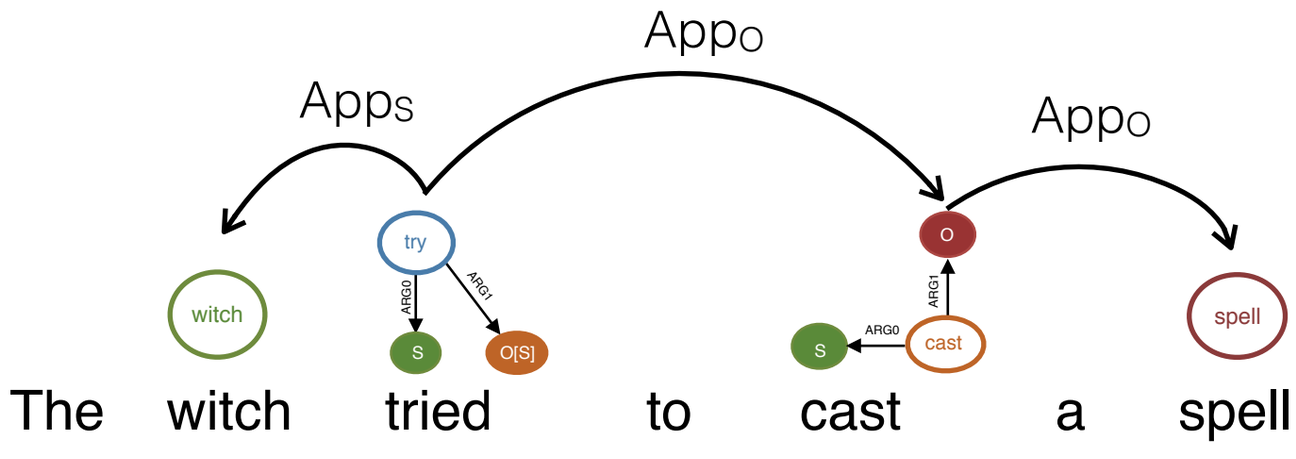
Model

1. **Supertagging:** score graph fragments for each word

2. **Dependency model:** score operations

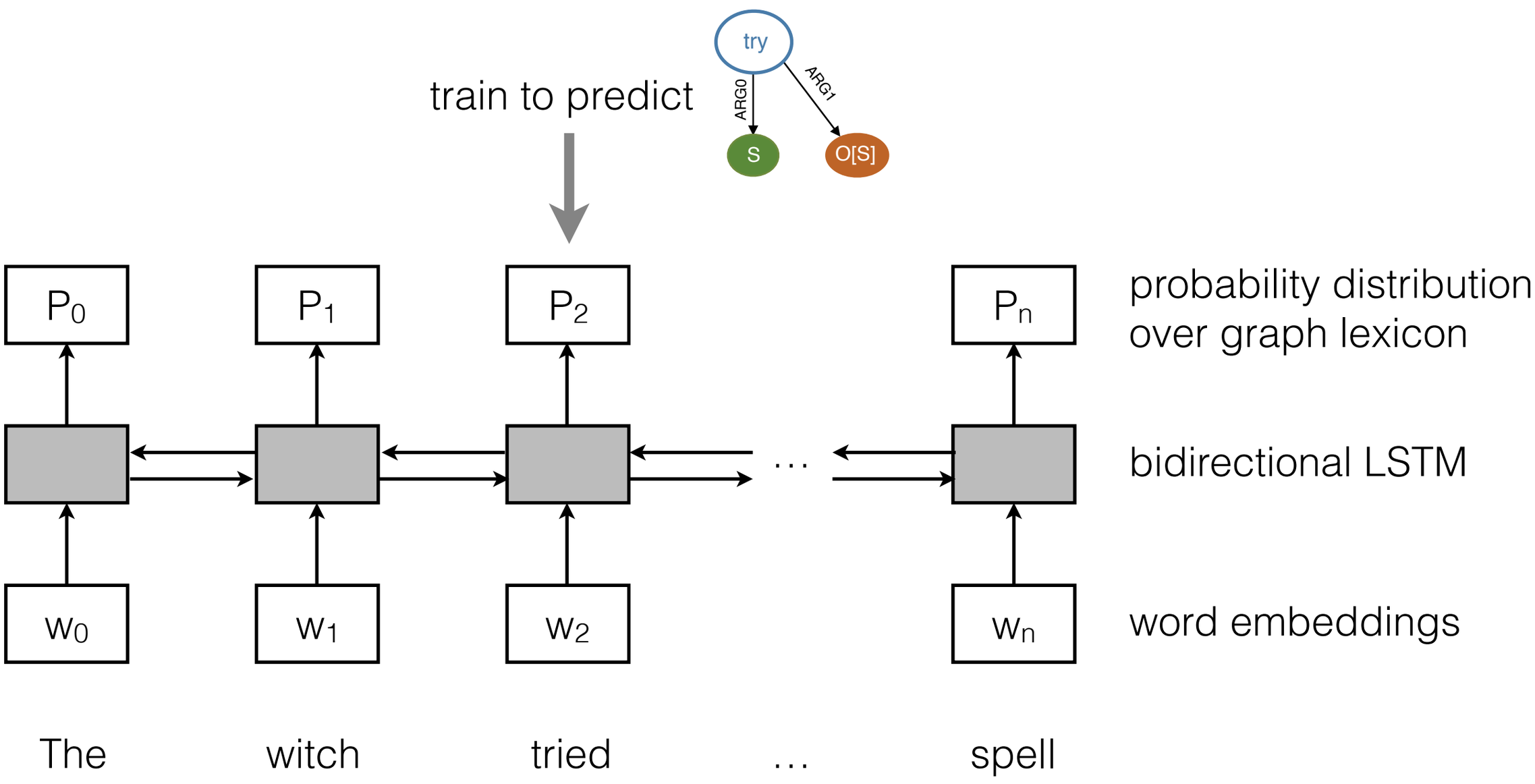


3. **Decoding**: find highest-scoring well-typed tree



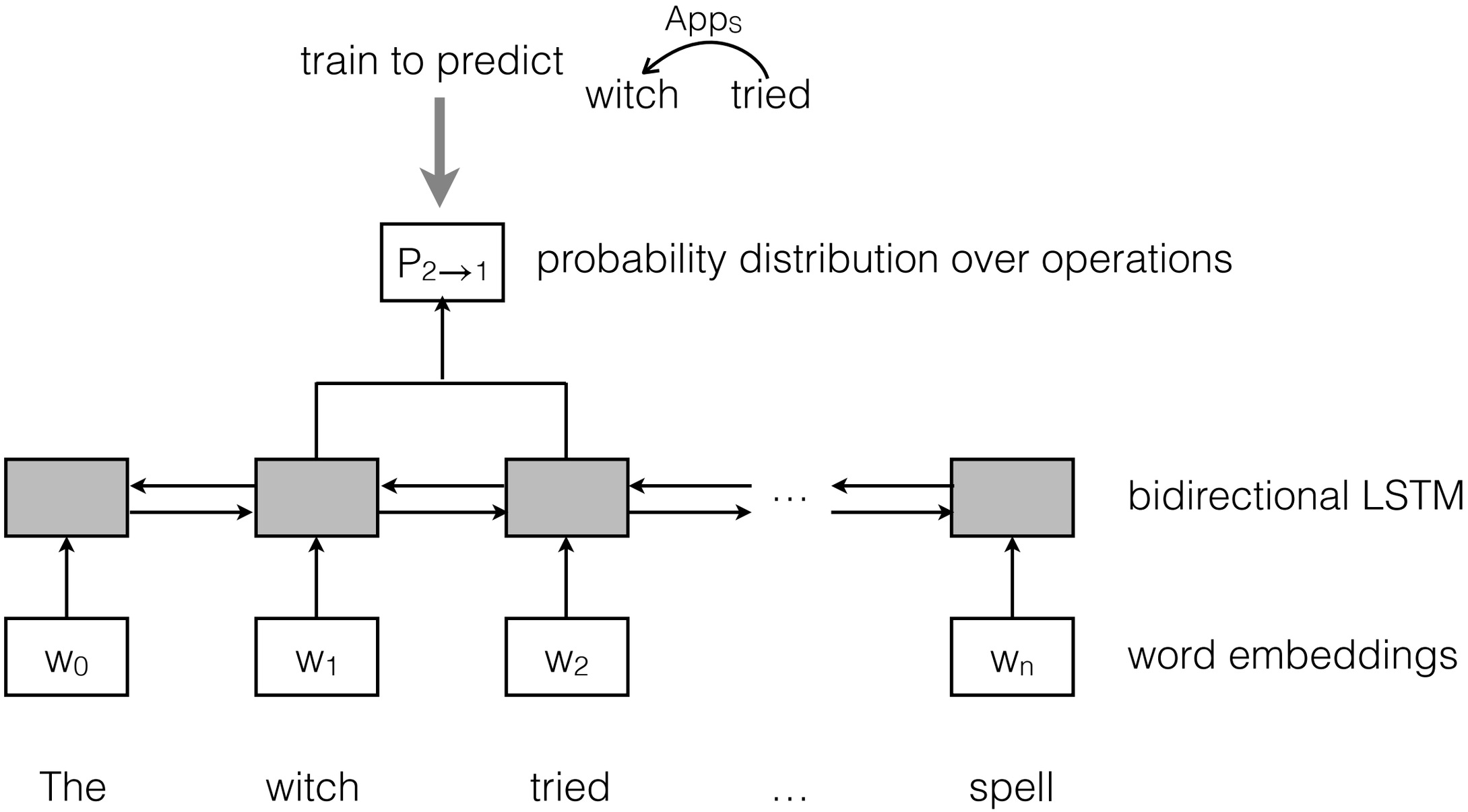
1. Supertagging

E.g. Lewis et al. (2014) for CCG

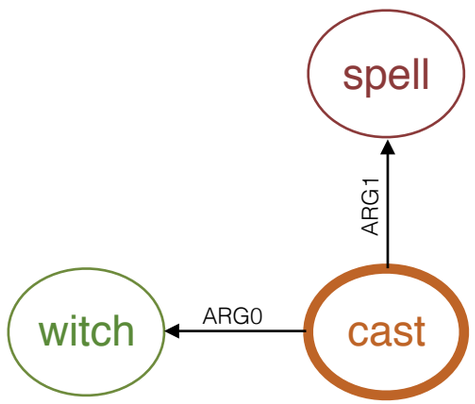


2. Dependency Model

Kiperwasser & Goldberg (2016) for syntactic dependencies

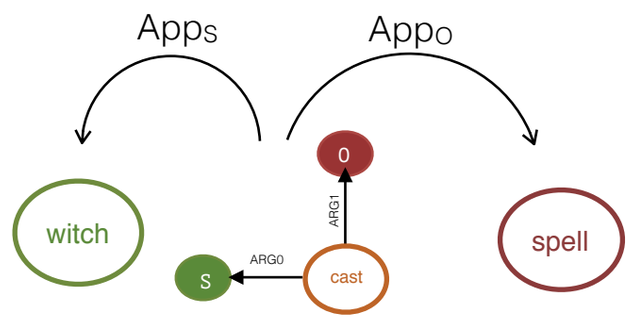


AMR Corpus



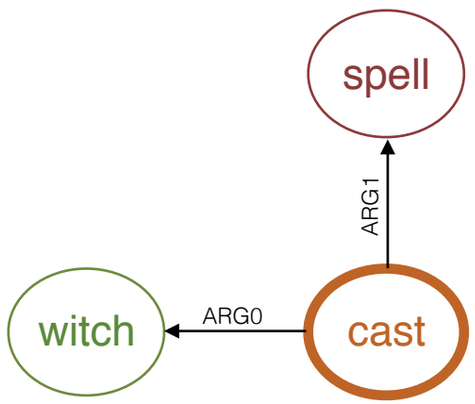
The witch cast a spell

Required training data



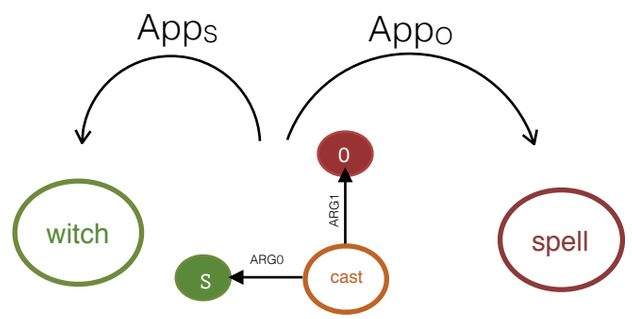
The witch cast a spell

AMR Corpus



The witch cast a spell

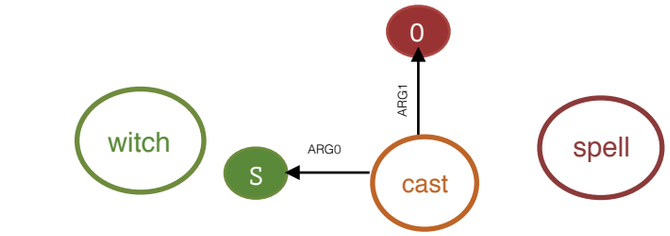
Required training data



The witch cast a spell

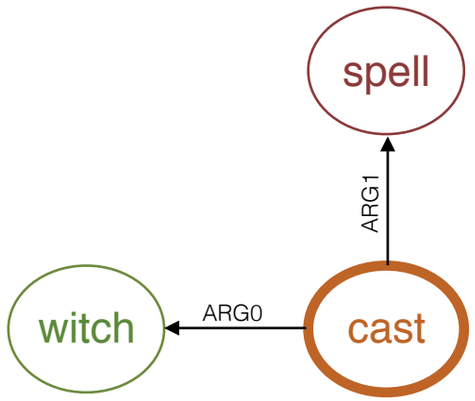
Heuristics

- Alignments
- Attaching edges
- Source names
- Source annotations



The witch cast a spell

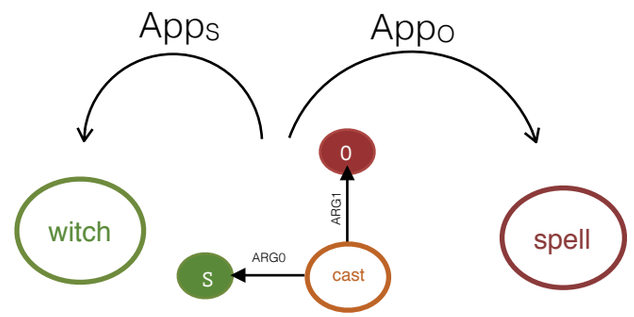
AMR Corpus



The witch cast a spell

determine

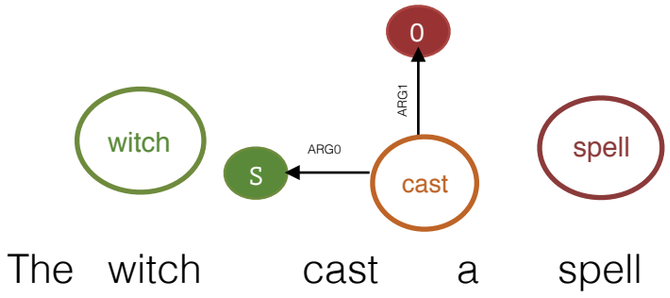
Required training data



The witch cast a spell

Heuristics

- Alignments
- Attaching edges
- Source names
- Source annotations



The witch cast a spell

3. Decoding

Find the **best well-typed** dependency tree

3. Decoding

Find the **best well-typed** dependency tree

- ill-typed trees do not evaluate to AMRs
- ill-typed trees to not match our linguistic intuitions

3. Decoding

Find the **best well-typed** dependency tree

- ill-typed trees do not evaluate to AMRs
- ill-typed trees to not match our linguistic intuitions
- Exact typed decoding is NP-hard
- Untyped decoding: 74% of trees are ill-typed

3. Decoding

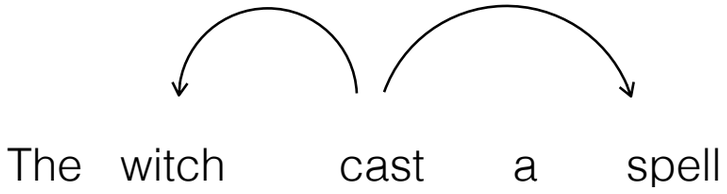
Find the **best well-typed** dependency tree

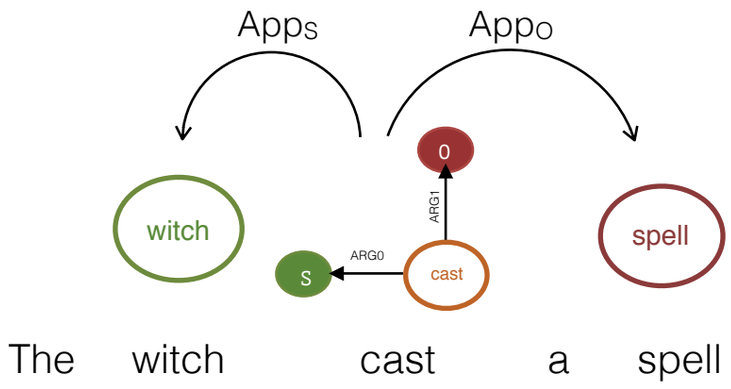
- ill-typed trees do not evaluate to AMRs
- ill-typed trees to not match our linguistic intuitions
- Exact typed decoding is NP-hard
- Untyped decoding: 74% of trees are ill-typed

➡ Approximate decoders

Approximate decoders

A: Fixed tree

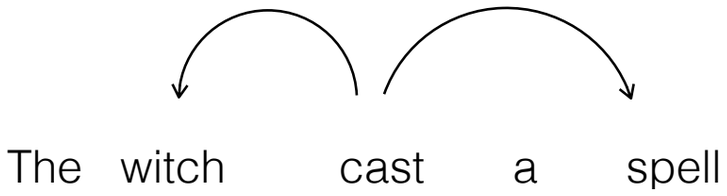




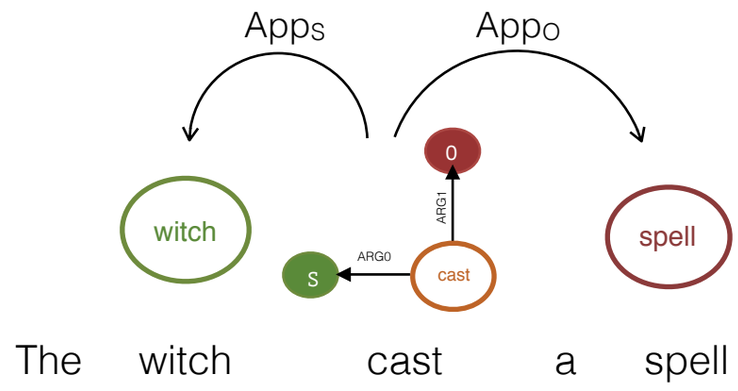
Approximate decoders

A: Fixed tree

1. Fix unlabeled tree



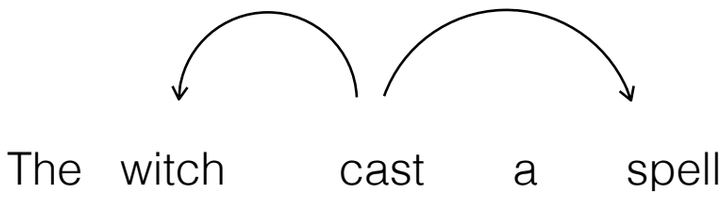
2. Label tree, with type checking



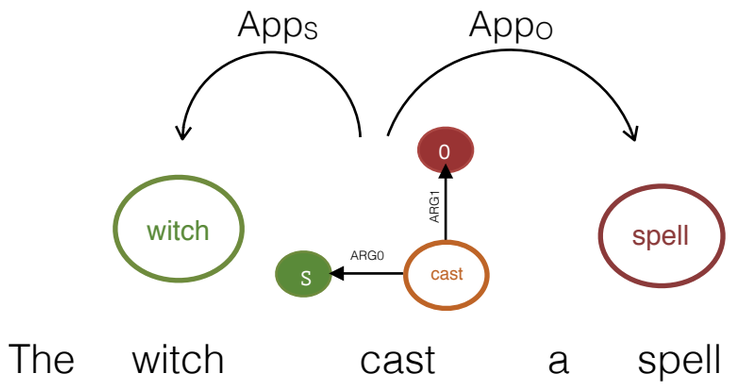
Approximate decoders

A: Fixed tree

1. Fix unlabeled tree



2. Label tree, with type checking



B: Projective: can only combine adjacent constituents

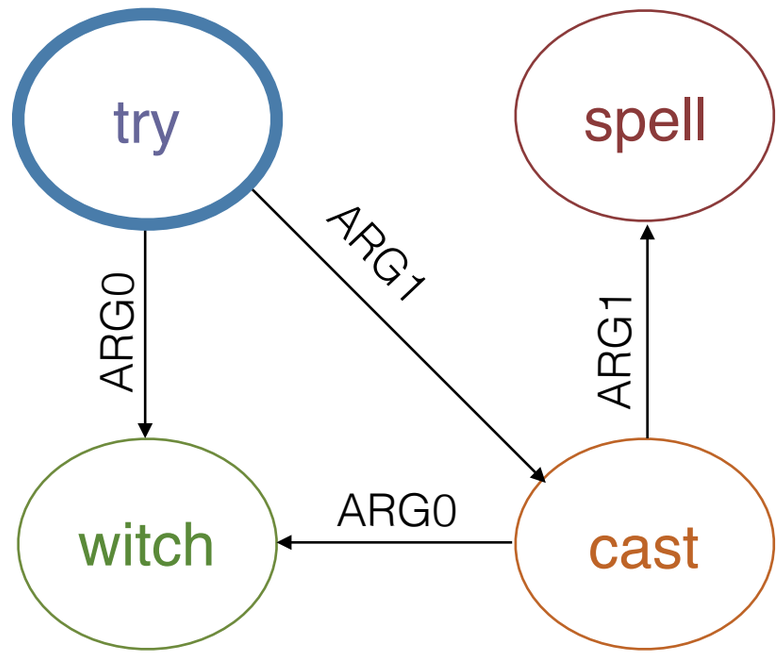
"CKY parsing with types as nonterminals"

Results

Classic AMR parser (*graph decoder*)

Step 1: Predict nodes

Step 2: Predict edges



Results

Model	Method	Smatch score
JAMR (Flanigan et al. 2016)	graph decoder	67
Foland & Martin 2017	graph decoder	70.7
van Noord & Bos 2017	neural seq2seq	68.5
Lyu & Titov (ACL 2018)	graph decoder	**73.7**
Our baseline	graph decoder	66.1
Our projective decoder		70.2
Our fixed tree decoder		70.2

Dataset: LDC2015E86

Conclusion

- We built a competitive compositional AMR parser
- Clear avenues to improvement
  - Update to recent advancements in training regimen (e.g. Lyu & Tivov 2018)
  - Look into specific phenomena, e.g.
    - anaphora
    - ellipsis
- Future work: extend method to other formalisms

Conclusion

- We built a competitive compositional AMR parser
- Clear avenues to improvement
  - Update to recent advancements in training regimen (e.g. Lyu & Tivov 2018)
  - Look into specific phenomena, e.g.
    - anaphora
    - ellipsis
- Future work: extend method to other formalisms