# Type Shifting and Overriding in Double R Grammar

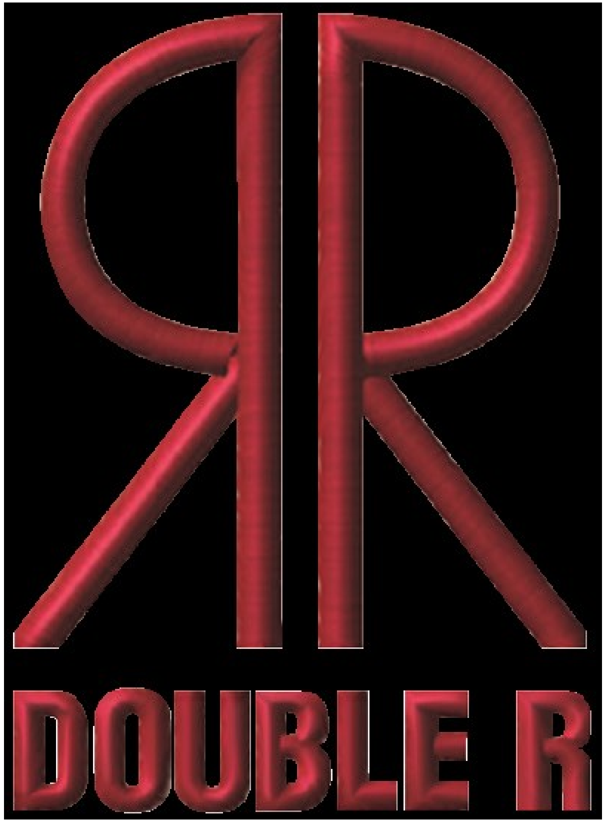
## Two Key Mechanisms of Context Accommodation

Jerry T. Ball & Stuart M. Rodgers[a]

[a]Institute for Defense Analyses

ACT-R Workshop 2023

# Double R Grammar



A Computational Cognitive Grammar of the Grammatical Analysis of Written English

The Grammatical Encoding of **Referential** and **Relational** Meaning

**Cognitively** and **Linguistically** Motivated

**Incremental** and **Interactive** Language Analysis

**Pseudo-Deterministic** Language Analysis

**Construction Driven** Language Analysis

**Large-scale** and **Functional**

Focus on meaning determination

# Double R Grammar

- Descendant of Propositional Model (PM) – 1985 to 1991
  - Implemented in Prolog (Programming in Logic) which proved to be inadequate
    - No probabilistic mechanisms and no type hierarchy
    - Unification based pattern matching to logic clauses, but no matching to higher level types
    - Strictly serial execution of matching logic clauses based on file order
    - Not suitable for preference semantics – couldn’t handle ambiguity of natural language
- Ported to ACT-R 5 in 2003
  - Logic clauses → productions and facts → chunks
  - ACT-R 5 has probabilistic mechanisms and type hierarchy
  - Chunks organized into a type hierarchy
    - Parallel chunk retrieval uses base level and spreading activation across matching types
    - Single chunk retrieved
  - Productions matched in parallel using type hierarchy
    - Single matching production with highest utility serially executed
  - Used in development of a **synthetic teammate** capable of analyzing text chat from human teammates and piloting a simulated UAV (drone)

# Synthetic Teammate Project

- Simulated UAV reconnaissance task
- Synthetic Pilot
- Two Human Teammates
  - Navigator
  - Photographer
- Communicate via text messaging
- Implemented in ACT-R 5
  - Reimplemented GUI in Lisp!

    to support interaction of synthetic teammate
- Empirically evaluated
  - Overall performance of teams

    with a synthetic pilot was **not significantly different** from all human teams

# Double R Grammar

- Ported to Java ACT-R in 2015 for follow on to synthetic teammate project
- Advantages of Java ACT-R
  - Compatible with ACT-R 6
  - Lots of good Java programmers and more attractive than Lisp to newcomers
  - GUI integration easier in Java vs. Lisp
  - Borrowed Java based code for spelling correction using edit distance
  - Used Java based regular expressions for perceptual bypass – tokenization
  - Java based tree diagram code implemented by Stuart Rodgers
- Disadvantages of Java ACT-R
  - Limited support for non-standard implementation of ACT-R
  - Most of ACT-R community uses standard Lisp version
  - Java based functionality not within ACT-R cognitive architecture

# Current Capabilities

- Written word recognition
  - ~100,000 words and multi-word units in mental lexicon
    - In line with estimates of size of human mental lexicon
    - Created with a combination of automated and manual techniques
  - ~70 parts of speech organized into a multiple inheritance hierarchy
  - ~50 prefixes, ~120 suffixes and ~40 morphological analysis productions
  - ~35 lexical retrieval and perceptual bypass productions
  - Perceptual bypass uses Java based regular expressions for tokenization
  - Spelling correction uses Java code based on edit distance algorithm
  - Single and double shot learning of new lexical chunks for unknown words
- Recently achieved **98.5%** part of speech tagging accuracy rate
  - Competitive with state of the art machine learning and deep learning systems
  - Results reported at Virtual ICCM 2023

# Current Capabilities

- Grammatical analysis
  - ~150 grammatical chunk types
  - ~2000 grammatical analysis productions
    - Manually created and tuned – linguistically motivated
  - Covers most of the common grammatical constructions of English
    - Declarative clause, wh & yes-no question, imperative clause, existential *there,* locative focus, relative clause, wh clause, *that* complement clause
    - Intransitive, transitive, ditransitive and situation complement predicates
    - Active vs. passive alternation, indirect object vs. recipient alternation
    - Nominal, possessive nominal, conjunction, punctuation
  - Some more specialized constructions also handled
    - Comparative, focus, subject extraposition
  - Approaching breadth and accuracy of leading machine learning systems

# Type Shifting and Overriding

- Chunk types made more flexible in ACT-R 6
  - Able to dynamically change the type of a chunk
  - Able to dynamically add or remove slots from a chunk
- These changes made a type shifting mechanism possible
- Previously only able to override a chunk with an alternative chunk
  - But chunk may be integrated into higher level chunk
    - Need to adjust pointers from higher level chunks to alternative chunk
  - But instantiated slot values may be lost
    - Need to copy all appropriate instantiated slot values from overridden chunk to alternative chunk
- Why do we need type shifting and overriding?

# Incremental & Interactive Processing

- Incremental processing means no parallel access to right context
- Interactive processing means using all available information to make the best choice given current input and current context
- Locally best choice may not be globally preferred
- Need to accommodate evolving context
  - Context accommodation is non-monotonic – representation may be altered
- Overall, processing is pseudo-deterministic
  - Make best choice given current input and context
  - Assume choice is correct and proceed incrementally forward
  - Accommodate evolving context

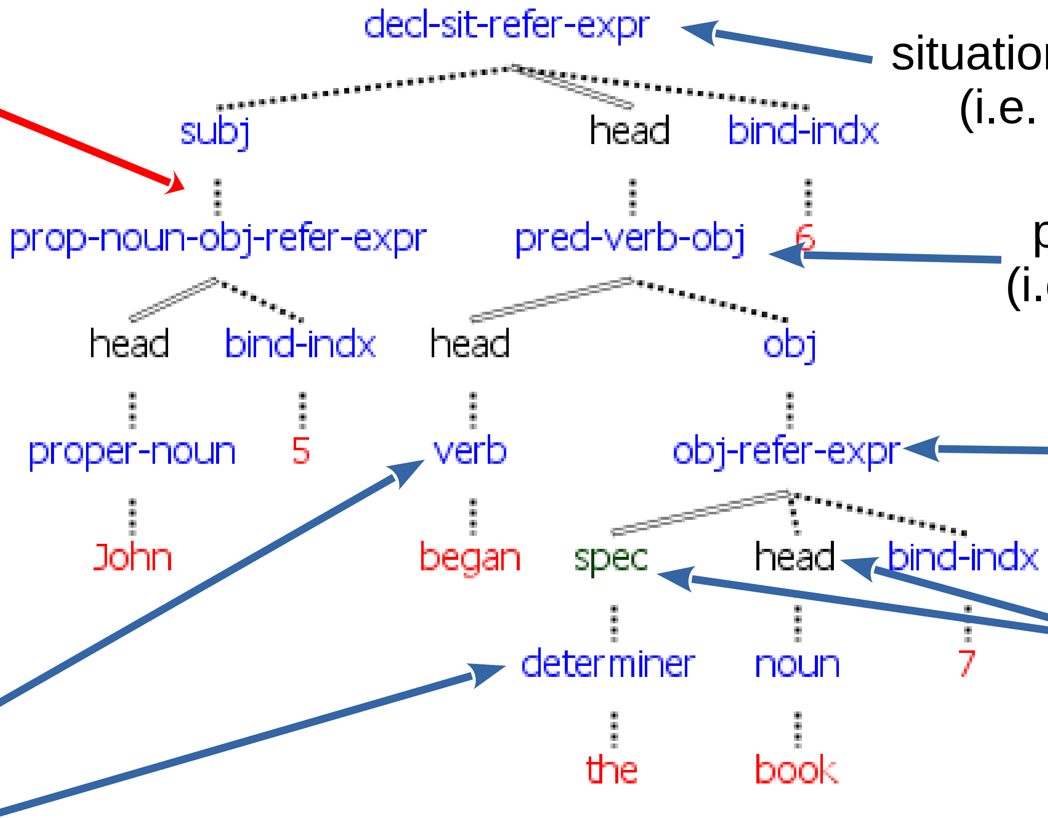
# Context Accommodation

- Type Shifting
  - Change the type of a chunk without replacing it
    - Instantiated slot values remain
    - Higher level pointers to chunk remain
  - Preferred to overriding when only minor adjustment is needed
  - Predicate intransitive verb construction → Predicate transitive verb construction
    - Add an object argument slot
- Overriding
  - Replace a chunk of one type with an alternative chunk of a different type
    - Instantiated slot values must be copied to alternative chunk or they will be lost
    - Higher level pointers must be shifted to alternative chunk or they will be incorrect
  - Preferred to type shifting when types are significantly different
    - Lexical noun → Object referring expression (i.e. nominal) construction
    - Lexical verb → Verbal predicate construction

# Type Shifting

- Change the type of an existing chunk
- *John began the book*
  - *Began* in mental lexicon as a verb that prefers an object argument
  - Access a predicate verb object (transitive predicate) construction when *began* is incrementally processed
  - Integrate *the book* as object argument when incrementally processed
- *John began reading*
  - *Began* can also occur with a situation complement with progressive (*reading*) or infinitive (*to read*) verb form
  - When *reading* is incrementally processed, type shift predicate verb object (transitive predicate) to predicate verb *ing* situation complement
  - Integrate *reading* as *ing* situation complement

# Representation for *John began the book*



pointer

declarative
situation referring expression
(i.e. declarative clause)

predicate verb object
(i.e. transitive predicate)

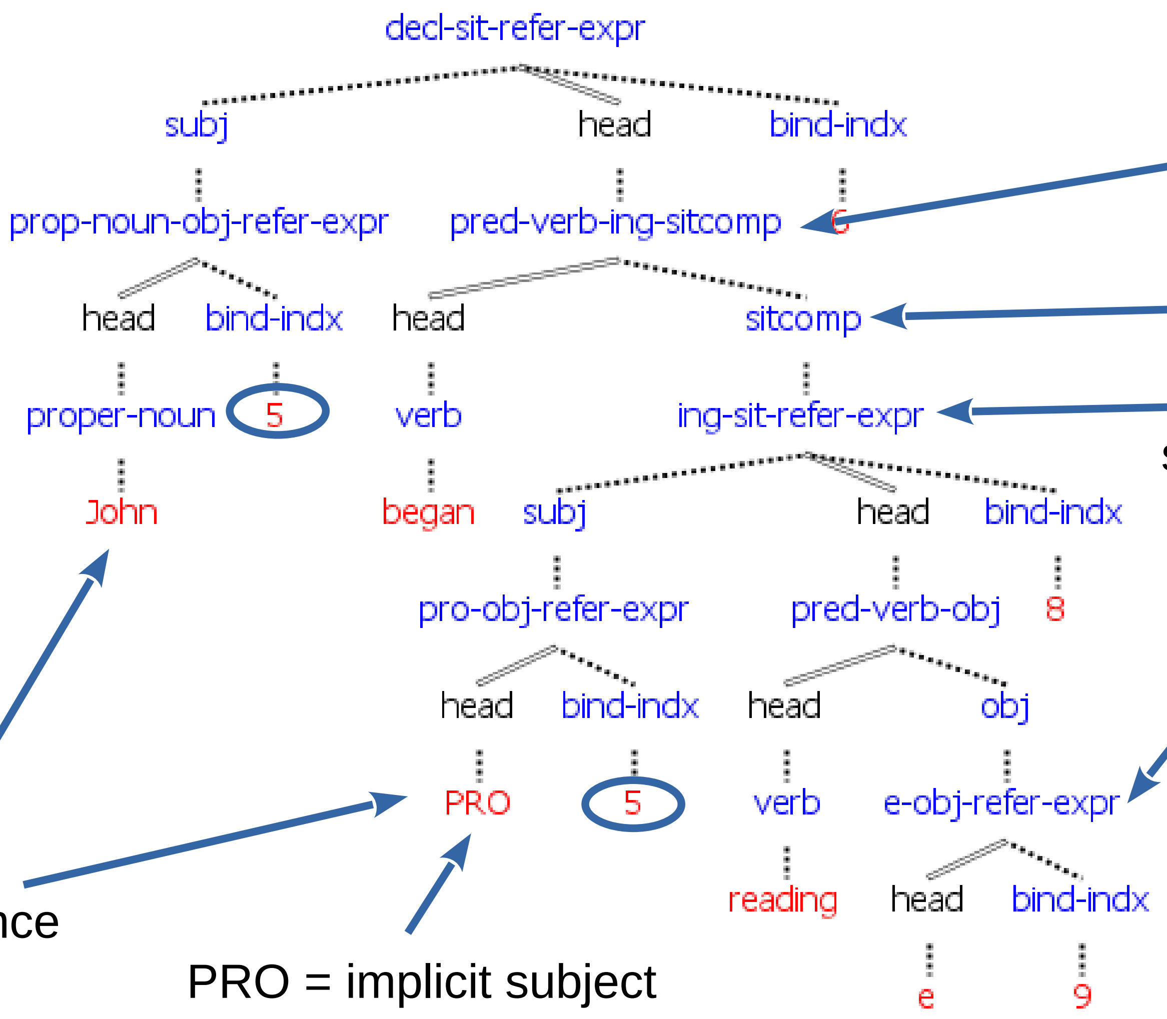
object referring
expression
(i.e. nominal)

grammatical
functions
(GFs)

Java based tree diagrams integrate multiple chunks for display purposes, but are really pointers in ACT-R

chunks must be accessible for type shifting
– buffers preferred over retrieval

# Representation for *John began reading*



type shift pred-verb-obj to pred-verb-ing-sitcomp

new grammatical function

integrate *ing* situation referring expression (i.e. progressive clause) as situation complement

integrate empty (e) implicit object of transitive verb *reading*

subject co-reference

PRO = implicit subject

# Type Shifting

- ’s = possessive marker or auxiliary
  - *John*’s *book* – ’s = possessive marker
  - *John*’s *going* – ’s = auxiliary verb *is*
- With type shifting, we can choose one and accommodate the alternative
- Alternatively, we could have a composite part of speech category
  - But meaning of possessive marker is different from meaning of auxiliary verb *is*

# Type Shifting

- *’s* = auxiliary verb *is* or *has*
- *John’s gone*
  - Default preference is auxiliary verb *is*
  - But cliticized auxiliary use is ambiguous
  - Need fine-grained meaning to disambiguate
- *John is gone* – intransitive inactive (from *is*)
  - Inactive voice is intransitive equivalent to transitive passive
- *John has gone* – intransitive active (from *has*)

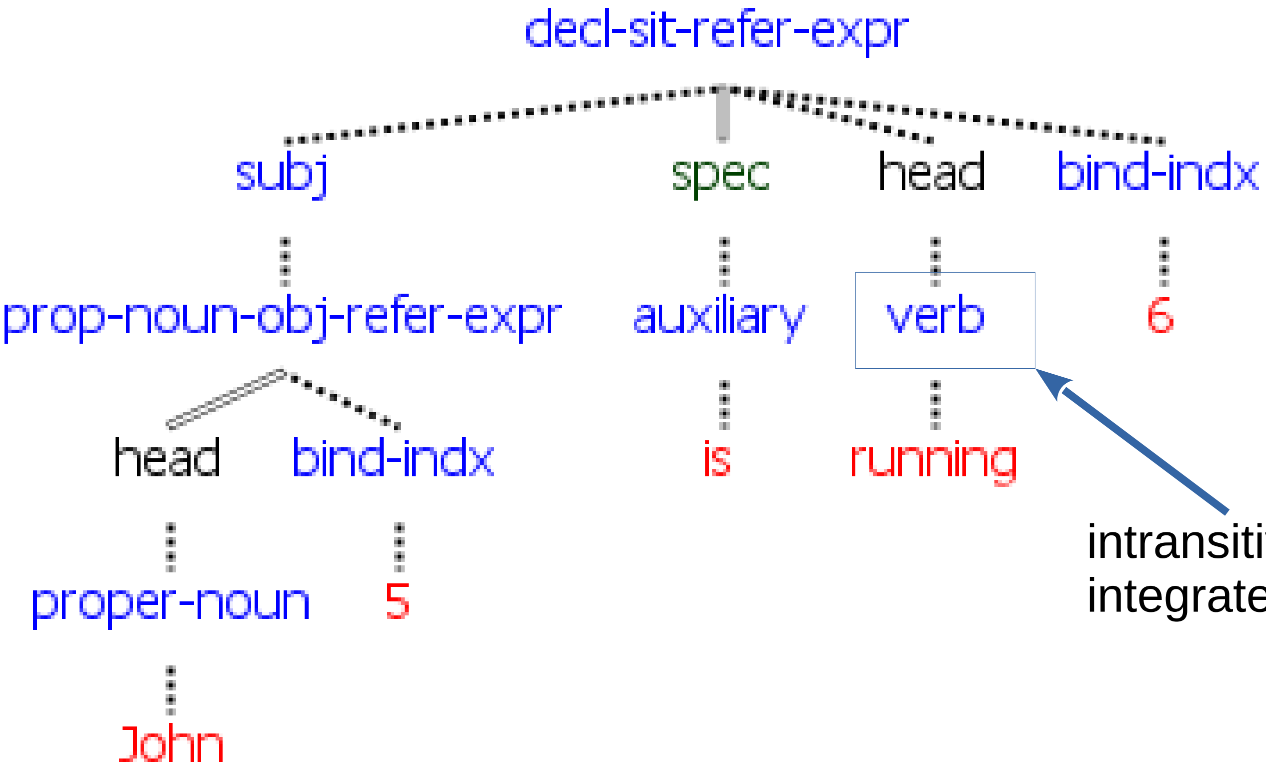
# Type Shifting

- *The taxi’s waiting he’s blowing his horn – ’s = is*
  - Default preference for *’s* is auxiliary verb *is*
  - Default preference is reinforced by subsequent progressive participle
    - *he is blowing* vs. **he has blowing*
- *The taxi’s waiting he’s blown his horn – ’s = has*
  - In context of perfect participle *blown*, type shift lexical chunk to *has*
  - Is *has* preferred over *is* when followed by perfect participle?
  - Preference may be verb specific and context dependent
    - Passive: *John’s kicked the ball by Bill* = *is*
    - Active: *John’s kicked the ball to Bill* = *has*

# Overriding

- Override an existing chunk with an alternative chunk
  - *John is running*
    - Intransitive verb *running* integrated as predicate head GF without accessing a predicate intransitive verb construction – build minimal structure needed
  - *John is running fast*
    - Use accessible predicate intransitive verb construction so that the adverb *fast* can be integrated as a predicate modifier
    - Override lexical chunk (*running*) with grammatical chunk (predicate intransitive verb)
    - Shift lexical chunk (*running*) from clausal head GF to predicate head
    - Overriding + function shifting ~ adjunction in Tree Adjoining Grammar (TAG)
- Since lexical chunk already integrated, need to adjust pointer from higher level chunk to point to predicate intransitive verb instead — higher level chunk must be accessible

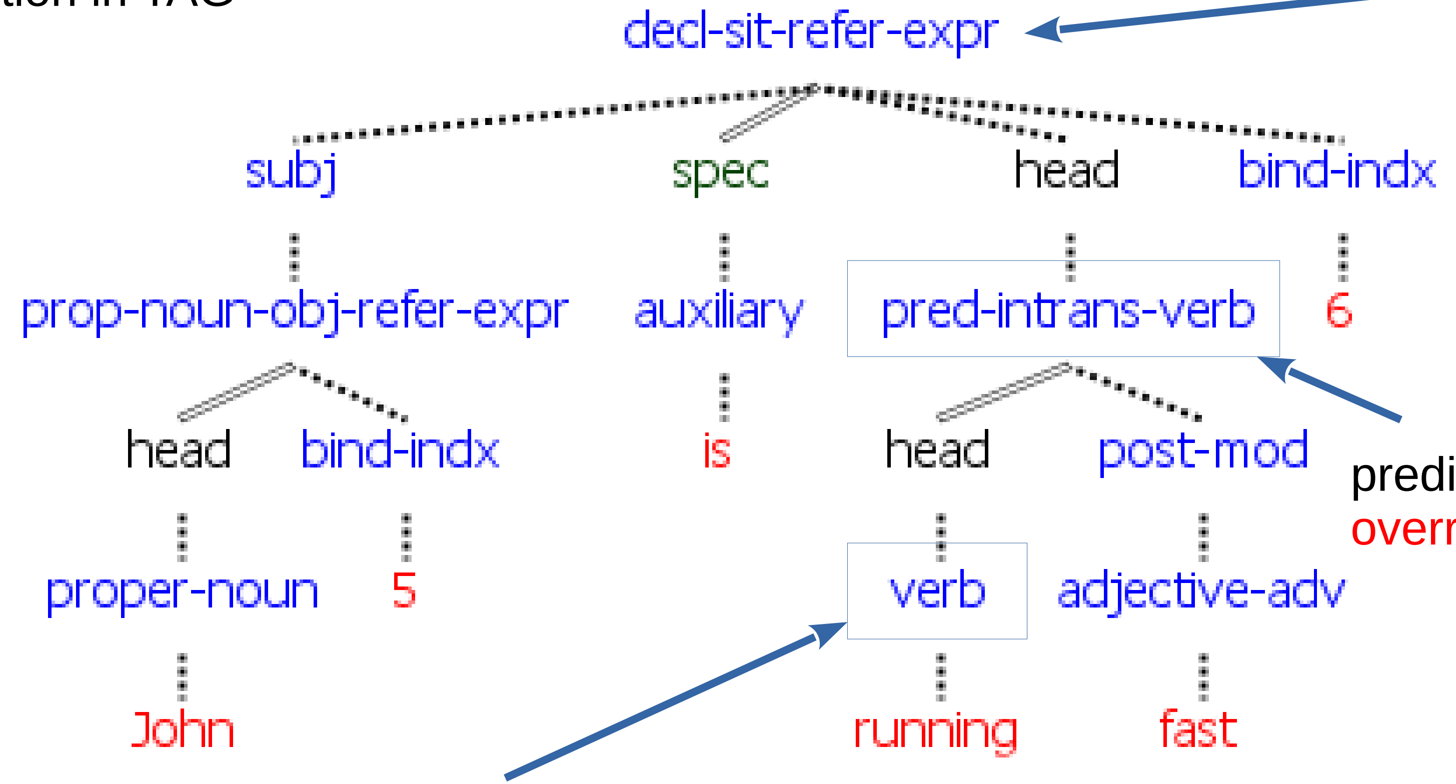
# Overriding



intransitive verb *running*
integrated as clausal head GF

# Overriding

overriding + function shifting ~ adjunction in TAG



higher level chunk must be accessible

predicate intransitive verb overrides verb as head

intransitive verb *running* shifted from clausal head to predicate head (function shifting)

# Summary

- Context accommodation needed given well-established cognitive constraints on Human Language Processing
  - Incremental and interactive processing
- ACT-R 6 (Java ACT-R) provides architectural support for type shifting and overriding
- Prefer type shifting for minor changes to chunk
- Prefer overriding for major changes which motivate creation of an alternative chunk
- Need for context accommodation is very common
  - Many intransitive verbs can be used transitively
    - *John smiled a big smile*
  - Many transitive verbs can be used ditransitively
    - *John kicked Mary the ball*
  - Construction driven accommodation (e.g. caused motion construction)
    - *Mary sneezed the napkin off the table*

Jerry Ball

Stu Rodgers

AIR FORCE RESEARCH LABORATORY

Cognitive Engineering Research Institute
for Collaborative Innovation

SANDIA RESEARCH CORPORATION

ASU

Georgia Tech

**Synthetic Teammate Kickoff Meeting**
**Arizona State University**
**December 7-8, 2017**

# References

Ball, J. & Rodgers, S. (2023). Cognitively and Linguistically Motivated Part of Speech Tagging: Quantitative Assessment of a Near Human-Scale Computational Cognitive Model. Virtual ICCM.

Ball, J. (2023, in preparation). Double R Grammar, the Grammatical Encoding of Referential and Relational Meaning. Preprint available at https://www.researchgate.net/profile/Jerry-Ball/research

Ball, J. (2011). A Pseudo-Deterministic Model of Human Language Processing. In L. Carlson, C. Hölscher, & T. Shipley (Eds.), *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, 495-500. Austin, TX: Cognitive Science Society.

Ball, J. (2007). A Bi-Polar Theory of Nominal and Clause Structure and Function. *Annual Review of Cognitive Linguistics*, 27-54. Amsterdam: John Benjamins.

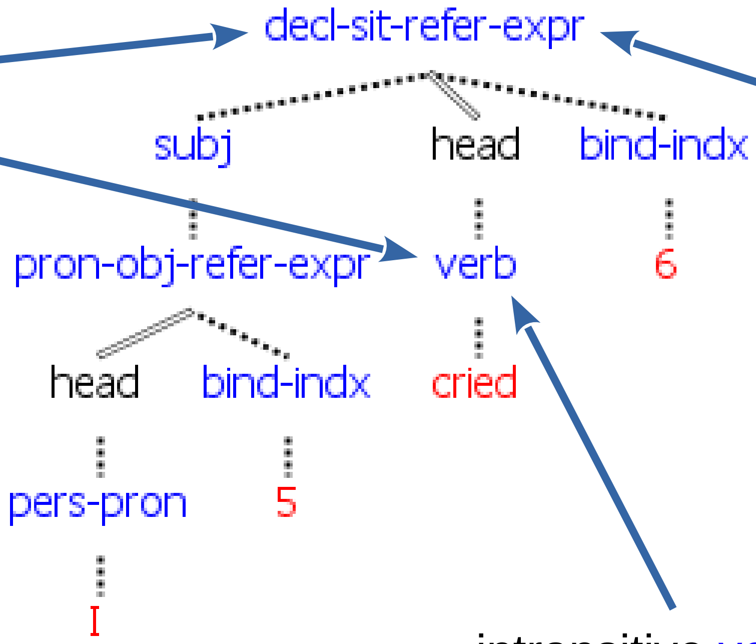
# Additional Examples

# Type Shifting, Overriding & Function Shifting

- I *cried a river over you*
  - *Cried* only in mental lexicon as an intransitive verb
  - Type shift intransitive predicate associated with *cried* to transitive when *a river* is incrementally processed
    - Intransitive predicate chunk available in parallel in buffer to avoid retrieval or projection which requires extra production – brings Double R in to closer alignment with human reading rates
  - Override lexical chunk with transitive predicate chunk
  - Integrate *a river* as the object of transitive predicate chunk

# Incremental Representation for *I cried...*

Java based tree diagrams integrate multiple chunks for display purposes



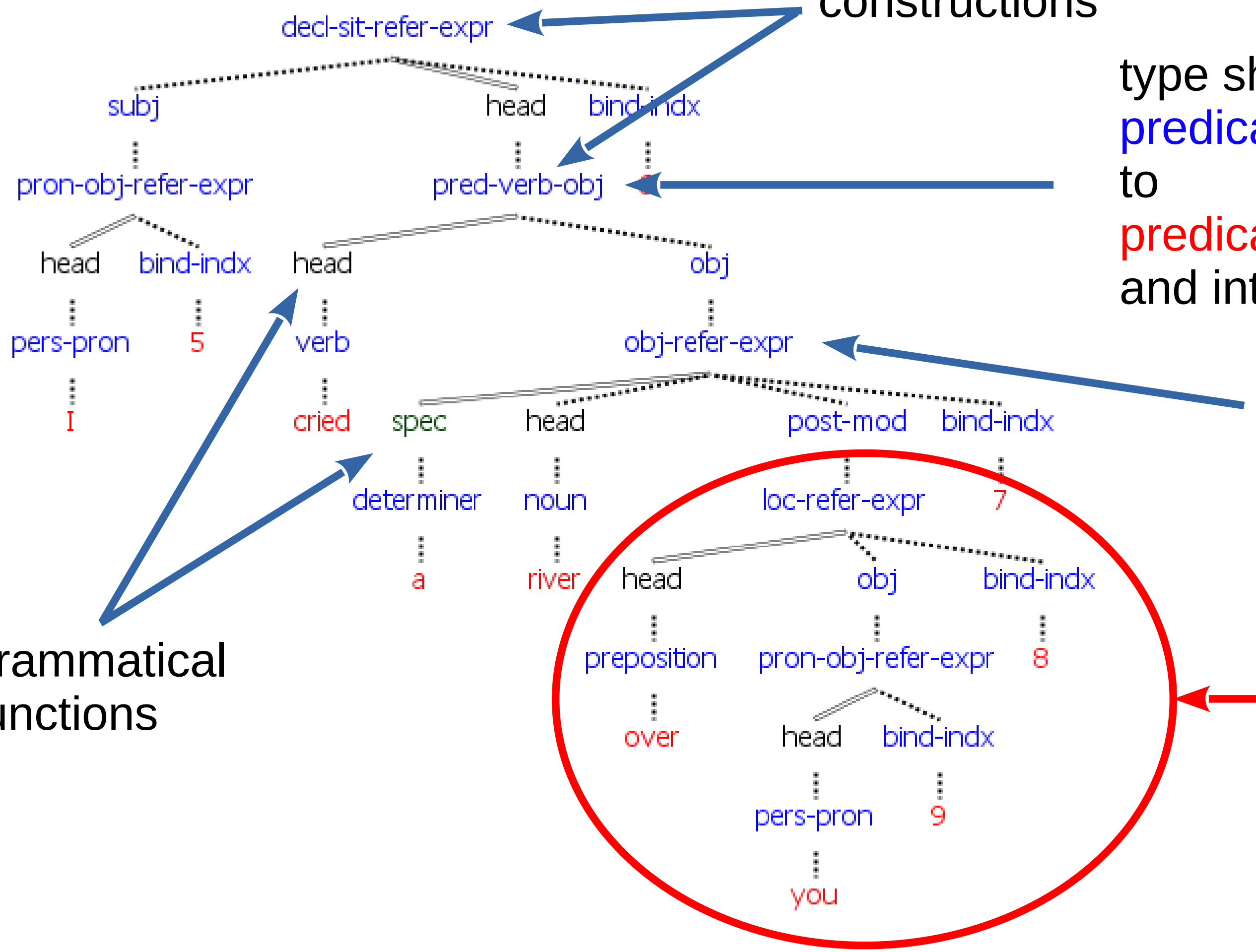
declarative
situation referring expression
(i.e. clause)

chunk integration is via pointers – chunk not directly integrated

intransitive verb integrated as head

chunks must be in buffers to be accessible without a retrieval

# Representation for *I cried a river over you*



constructions

type shift
predicate intransitive verb
to
predicate verb object = transitive
and integrate as head

integrate
object referring expression
as object

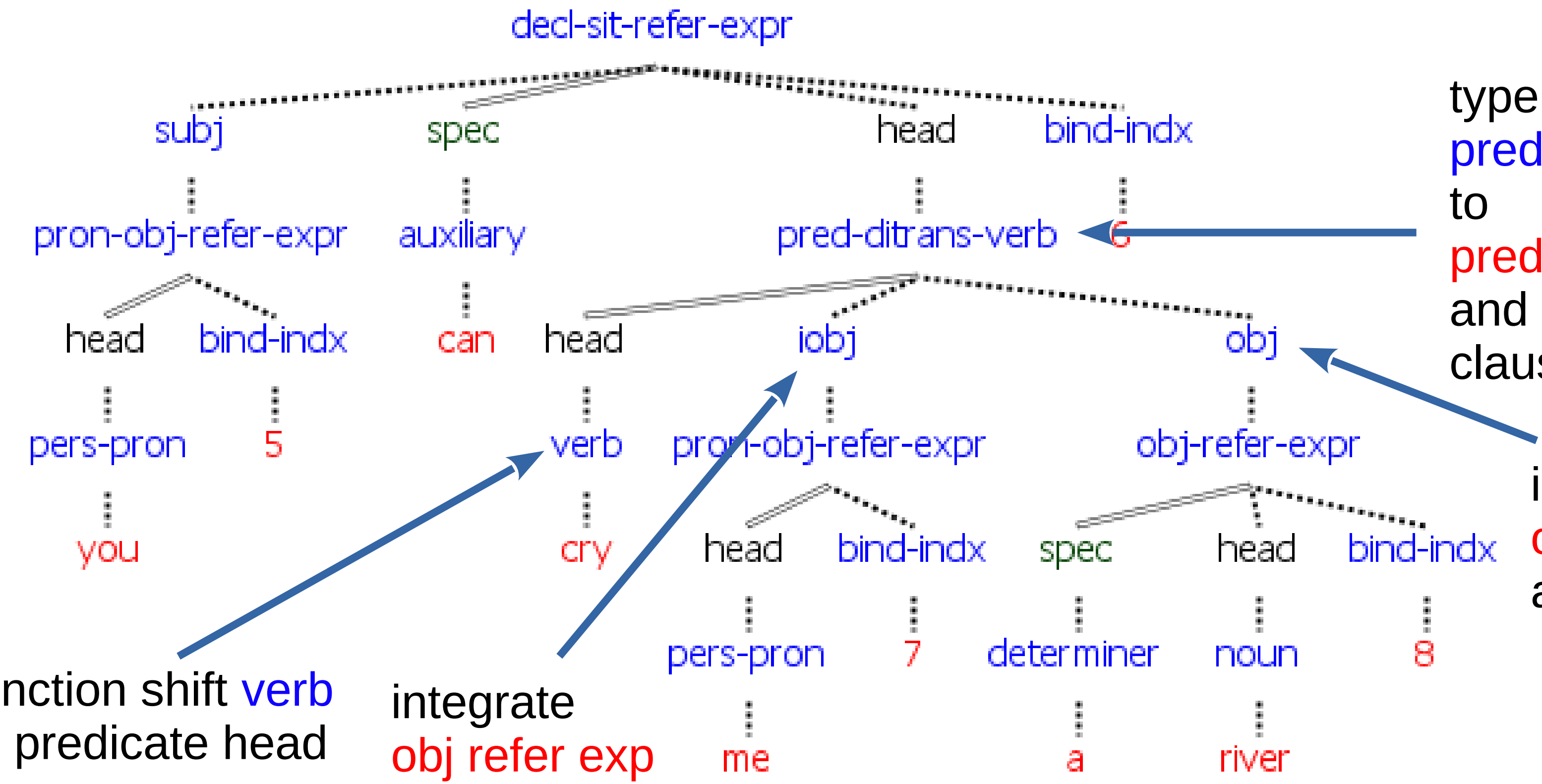
grammatical
functions

*over you* should be
a modifier of *cried*
not *a river*
(prepositional phrase
attachment ambiguity)

# Type Shifting, Overriding & Function Shifting

- *You can cry me a river*
  - Type shift intransitive predicate associated with *cried* to ditransitive
    - Intransitive predicate available in parallel in buffer
  - Override intransitive verb with predicate ditransitive verb construction
  - Shift intransitive verb from clausal head GF to predicate head
  - Integrate *me* as the indirect object
  - Integrate *a river* as the object

# Type Shifting, Overriding & Function Shifting



type shift
predicate intransitive verb
to
predicate ditransitive verb
and override verb as
clausal head GF

integrate
obj referring expression
as object

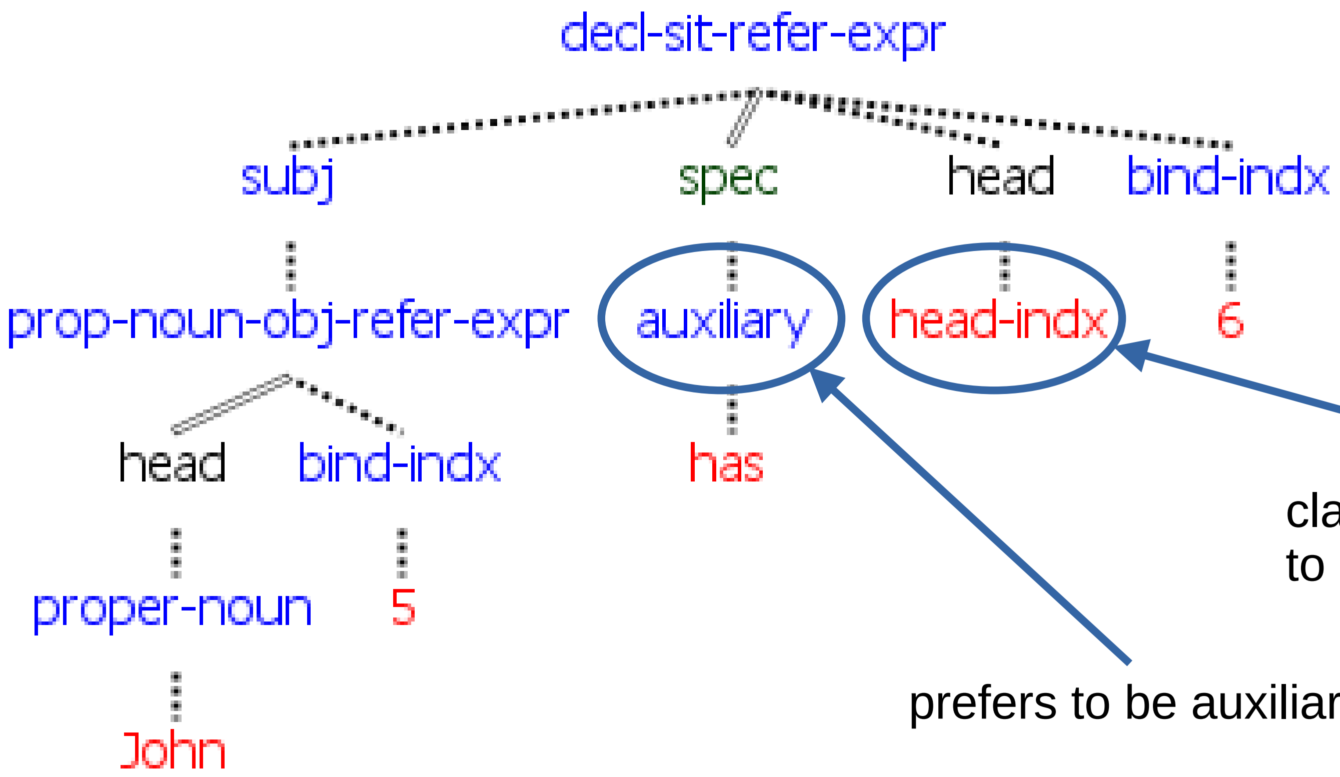
function shift verb
to predicate head

integrate
obj refer exp
as indirect
object

# Type Shifting, Overriding & Function Shifting

- *John has...*
  - *Has* prefers to be an auxiliary verb
  - Incrementally integrate *has* as clausal specifier GF of situation referring expression (i.e. clause)

# Incremental Representation for *John* ***has***...



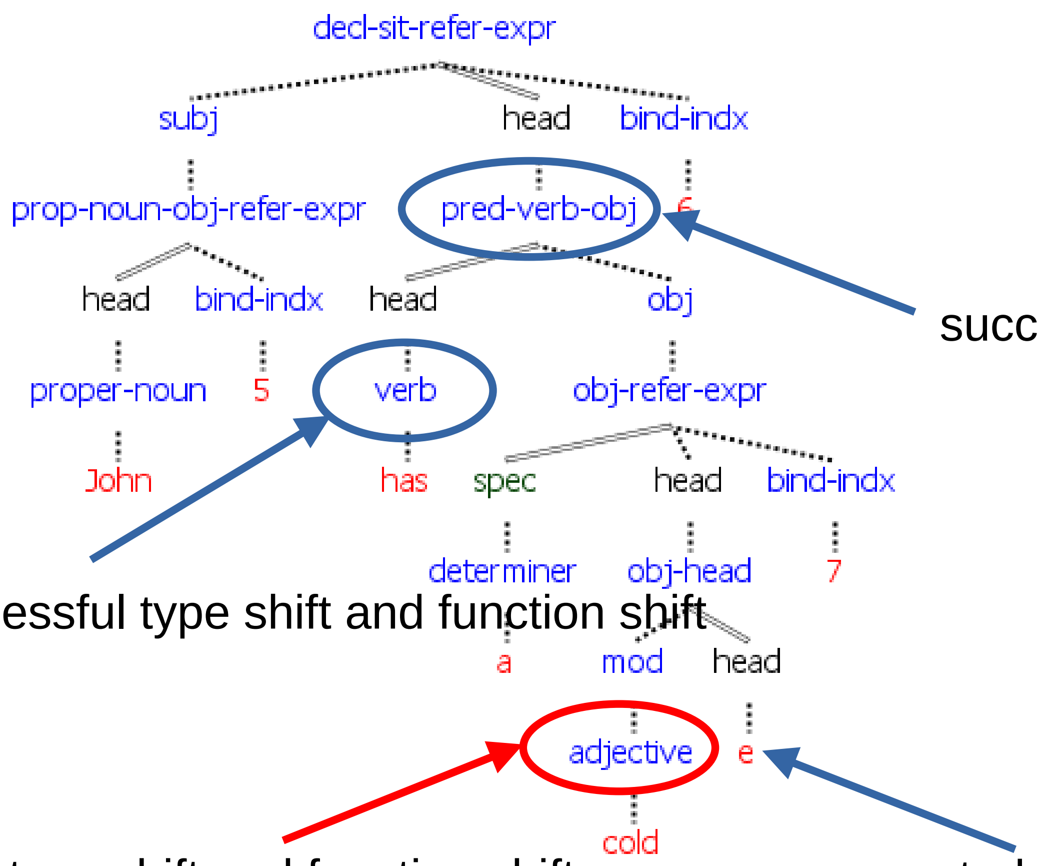
clausal head is predicted to occur

prefers to be auxiliary verb

# Type Shifting, Overriding & Function Shifting

- *John has a cold*
  - In context of object referring expression (i.e. nominal) *a cold*, type shift *has* from auxiliary verb to regular verb
  - Access predicate transitive verb construction and integrate as clausal head
  - Shift *has* from clausal specifier to predicate head GF
  - *Cold* prefers to be an adjective (e.g. *John has a cold beer*)
  - Should also type shift *cold* from adjective to noun in absence of a head noun since *cold* can also be a noun
    - Generic wrap up production treats head as empty (e)
    - Need alternative production that type shifts *cold* from adjective to noun and function shifts *cold* from modifier to head when adjective can also be a noun

# Type Shifting, Overriding & Function Shifting



successful override

successful type shift and function shift

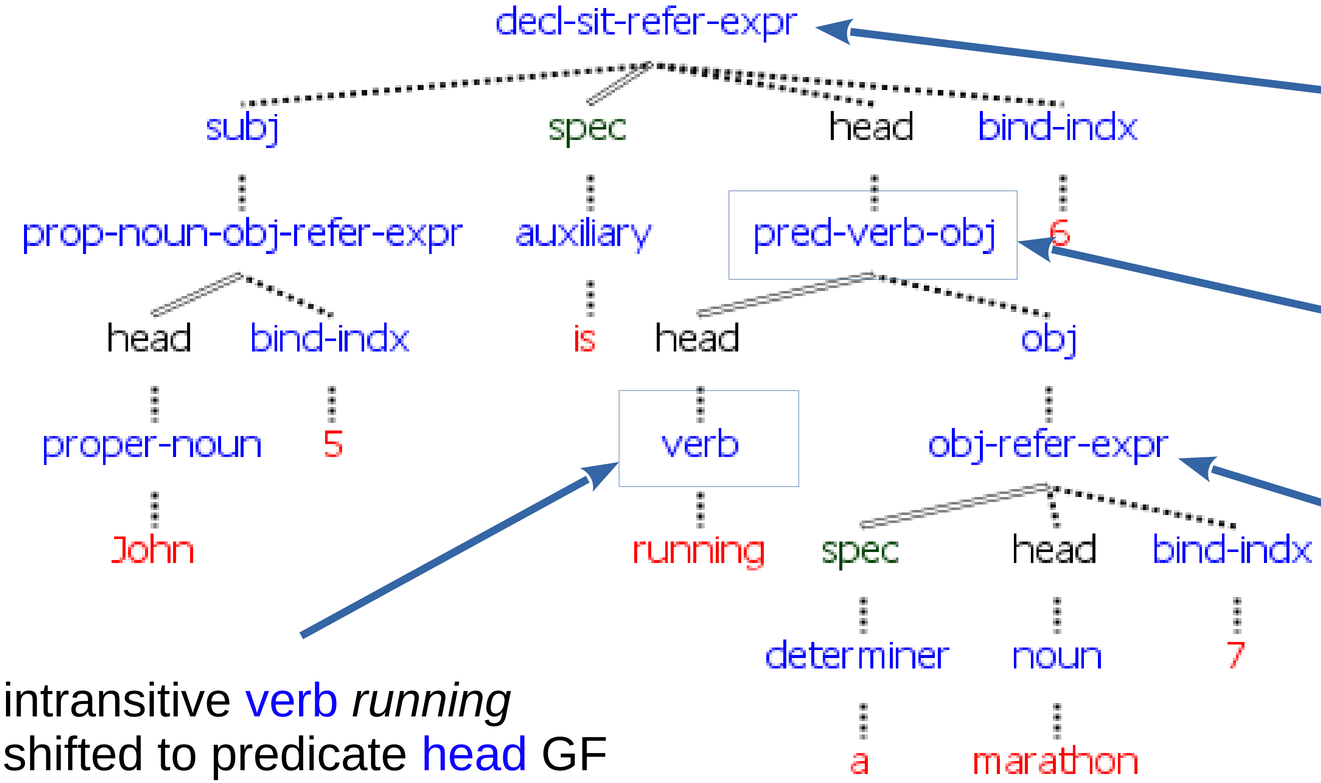
missed type shift and function shift

expected head is empty (e)

# Overriding + Function Shifting ~ Adjunction

- Override an existing chunk with an alternative chunk
  - *John is running a marathon*
    - Access a predicate verb object = transitive construction so that the nominal *a marathon* can be integrated as the object of *running*
    - Override lexical chunk (*running*) with grammatical chunk (predicate verb object)
    - Shift lexical chunk (*running*) to predicate head GF
- Since lexical chunk is already integrated, need to adjust pointer from higher level chunk to point to predicate verb object chunk — higher level chunk must be accessible
  - Use of buffers for accessibility preferred over retrieval

# Overriding + Function Shifting ~ Adjunction



higher level chunk
must be accessible

predicate verb object
overrides verb as head

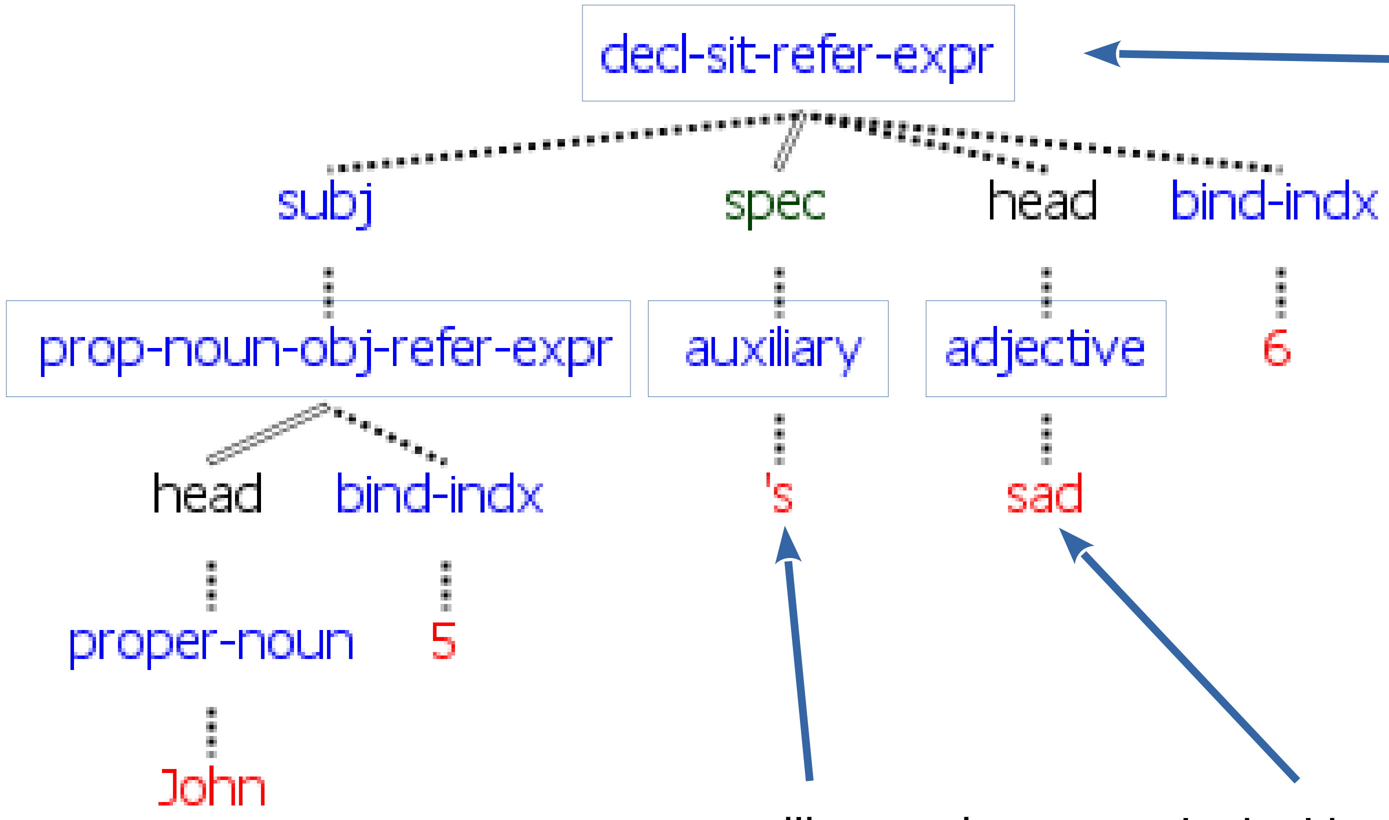
object referring expression
*a marathon* integrated
as object GF

intransitive verb *running*
shifted to predicate head GF

# Extreme Type Shifting, Overriding & Function Shifting

- *John’s sad*
  - Lexical chunk for adjective *sad* integrated as lexical head of situation referring expression (i.e. clause)
  - *’s* is the cliticized **auxiliary verb** *is* that functions as specifier

# Representation for *John's sad*



situation referring expression (i.e. clause)

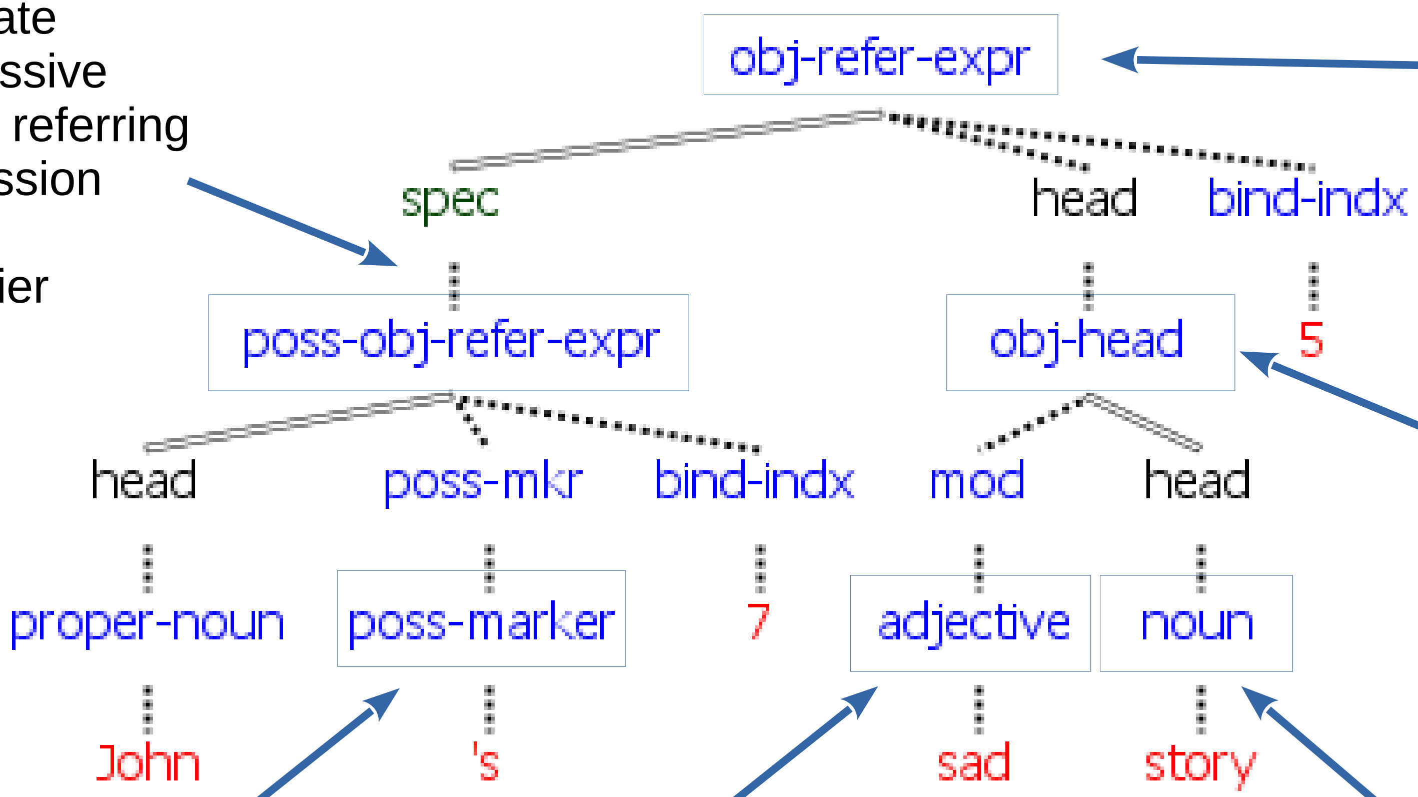
auxiliary verb

lexical head

# Extreme Type Shifting, Overriding & Function Shifting

- *John's sad story*
  - Adjective *sad* functions as modifier of noun *story* within object referring expression
  - Object head construction accessed
    - Noun *story* integrated as head GF, adjective *sad* integrated as modifier GF
  - Object referring expression (i.e. nominal) construction accessed
    - Object head integrated as head
  - Second object referring expression construction accessed and type shifted to possessive object referring expression
  - *'s* = auxiliary type shifted to possessive marker and integrated as poss-marker GF
  - Possessive object referring expression integrated as specifier GF of higher level object referring expression
  - Higher level object referring expression overrides situation referring expression

# Representation for *John's sad story*



integrate possessive object referring expression as specifier

object referring expression (i.e. nominal)

integrate object head as clause head GF

type shift

shift adjective to mod GF

integrate noun as object head GF

# Extreme Type Shifting, Overriding & Function Shifting

- No obvious processing difficulty for humans in making these extensive adjustments
  - No garden path effect – e.g. *The horse raced past the barn fell*
- But what if expression is already integrated?
  - *I think John's sad story is coming to an end*
- Need to adjust higher level pointers if chunk is overridden
- Need to copy appropriate instantiated values from overridden chunk and remove inappropriate values
  - Risk of losing previously instantiated values
  - Risk of having inappropriate values