

# Grammar Matrix (incl morphotactics)

## AGGREGATION

### Test suites

---

Ling 567

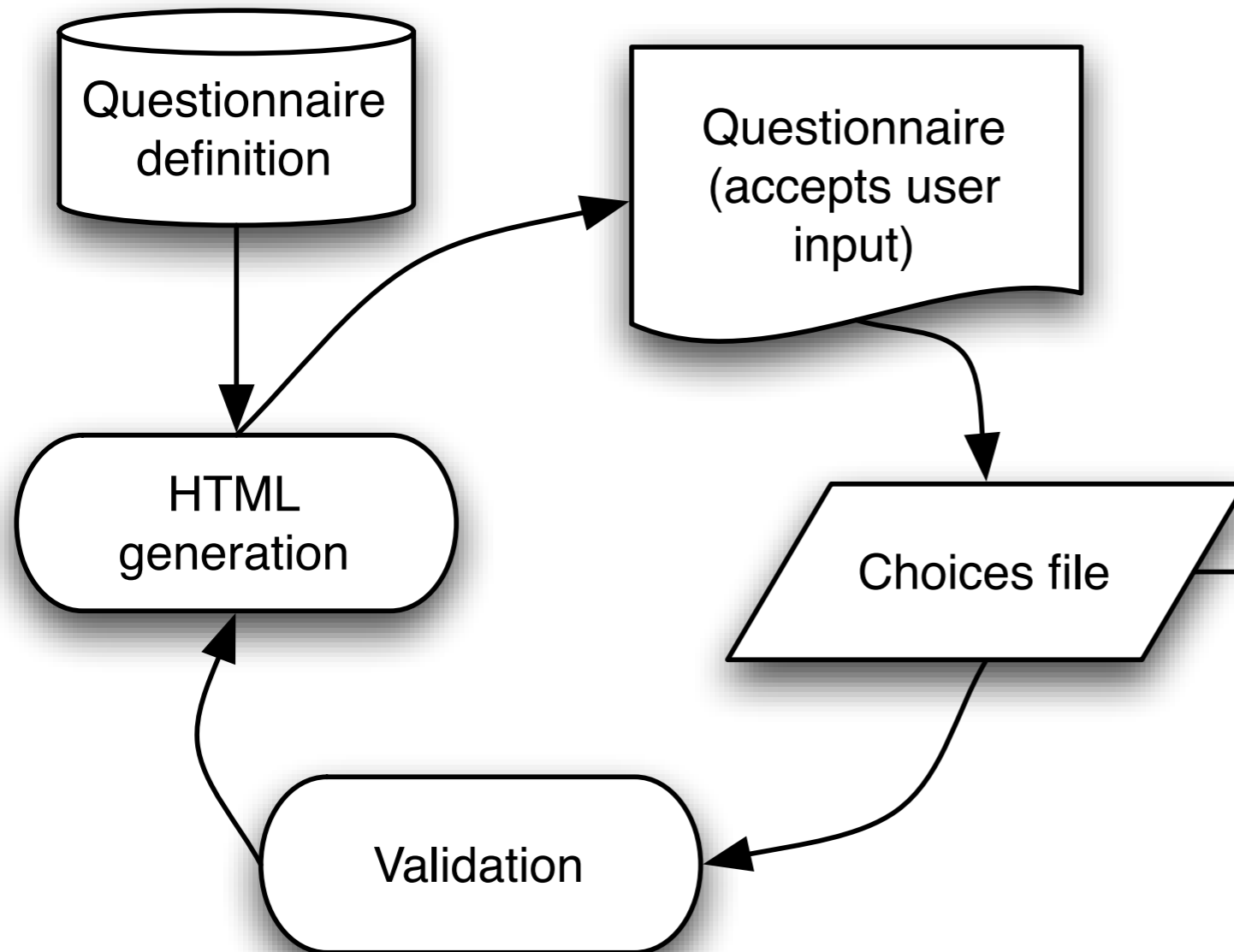
Jan 10, 2023

# Overview

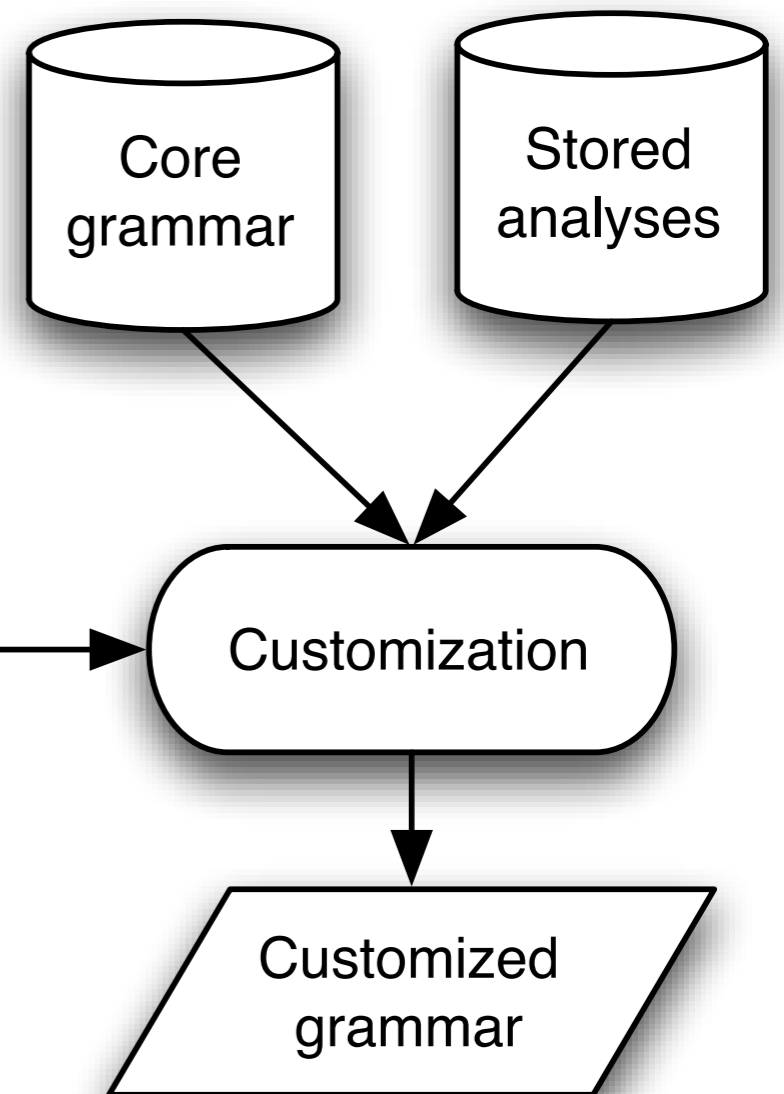
---

- Grammar Matrix customization system
- Questions from Lab 1
- AGGREGATION
- Testsuites & [incr tsdb()]
- Morphotactics in the Grammar Matrix

## Elicitation of typological information



## Grammar creation



# Creating a library for the customization system

---

- Choose phenomenon
- Review typological on phenomenon
- Refine definition of phenomenon
- Conceptualize range of variation within phenomenon
- Review HPSG (& broader syntactic) literature on phenomenon
- Pin down target MRSs
- Develop HPSG analyses for each variant
- Implement analyses in tdl
- Develop questionnaire & customization logic
- Run regression tests
- Test with pseudo-languages
- Test with illustrative languages
- Test with held-out languages
- Add tests to regression tests
- Add to MatrixDoc pages

# Overview

---

- Grammar Matrix customization system
- Questions from Lab 1
- AGGREGATION
- Testsuites & [incr tsdb()]
- Morphotactics in the Grammar Matrix

# Lab 1 questions

---

- I tested non-parsing sentences such as "The dog chased a cats" but couldn't find the specific grammar or lexical rule that causes the NP "a cats" to fail to parse, or I was trying the incorrect unification. Which rule is it?
- How do I best trace why a unification fails? What is the algorithm for determining what rules conflict upon finding a unification failure?
- Why are some rules in 567\_English.tdl and some rules in Matrix.tdl? What's the relationship between matrix.tdl and English\_567.tdl? Some rules are present in one but not the other.

# Lab 1 questions

---

- Is it okay to refer to things like the Head Complement Rule and Head Specifier Rule as 'rules'? They are technically not defined in any of the .tdl files as rule, rather they are defined as phrases.
- I am confused where are the lexical rules encoded. I have only found lexical rule type hierarchy, but I think I am looking for the arrow notation in Ling566.
- What is basic-head-2nd-comp-phrase used for?
- How are grammar principles enforced? Are they baked into each rule? Inherited? Is there a separate rule that applies?

# Lab 1 questions

---

- I've noticed several identities in matrix.tdl that have very similar names, what causes these to arise? (e.g. binary-headed-phrase and basic-binary-headed-phrase or const-lex-rule and constant-lex-rule)
- What is the HOOK feature that is referenced frequently? The comments in matrix.tdl say it contains "externally visible attributes of a sign" and the FAQ says it contains "information available for further semantic composition" but I'm curious about how its used and why is see it so often.
- What is a "sign?" It seems to be a top-level generic lexical type (like lexeme in 566), but the name is throwing me off.



# Lab 1 questions

---

- Orienting myself in the information, folders, and abbreviations is a bit overwhelming at the moment. Is there an easier or more intuitive way to organize all this information while we're still learning?
- I don't really understand the new aspects of feature geometry. Things like LIGHT, MODIFIED, WH and more confuse me. Other than the FAQ, is there an easy or simple set that covers most of our needs? Or perhaps a set we can safely overlook for now?



# Lab 1 questions

---

- Is there any way to present the type hierarchy inheritance while showing exactly (or just roughly) what is contributing what aspect in an abbreviated manner? Even better, is there a way to do so while excluding the less-pertinent inheritances?

# Lab 1 questions

---

- How is logical ‘or’ indicated in feature structures (both in the visual output of the LKB GUI and in the source files)? In 566 we used a pipe (`|`), but I do not seem to have encountered anything equivalent in the LKB or the tdl files that I have happened to look at so far.
- Do periods within lists just separate distinct elements so that `< #synsem . #comps >` represents a list with two elements on it?
- I am confused by the meaning of `cons null` and how it represents the unification error.
- With our notation is it true that an identity like `#comps` can be an empty list but `< #comps >` must be non-empty, or is it possible for both to be empty lists?

# Lab 1 questions

---

- The Feature Geometry FAQ on Github says “NB we don’t do binding theory.” Is this still true or is that information outdated? It seems odd that the LKB would not do binding theory at all.
- What is the AGR feature for? I thought it would be involved in the unification failure of a sentence such as "a cat chase the dog", but a feature such as PNG seems to mostly occur with HOOK features.
- Is there a convention that guides where certain identities are defined?
-

# Overview

---

- Grammar Matrix customization system
- Questions from Lab 1
- AGGREGATION
- Testsuites & `[incr tsdb()]`
- Morphotactics in the Grammar Matrix

# AGGREGATION: Research goals

---

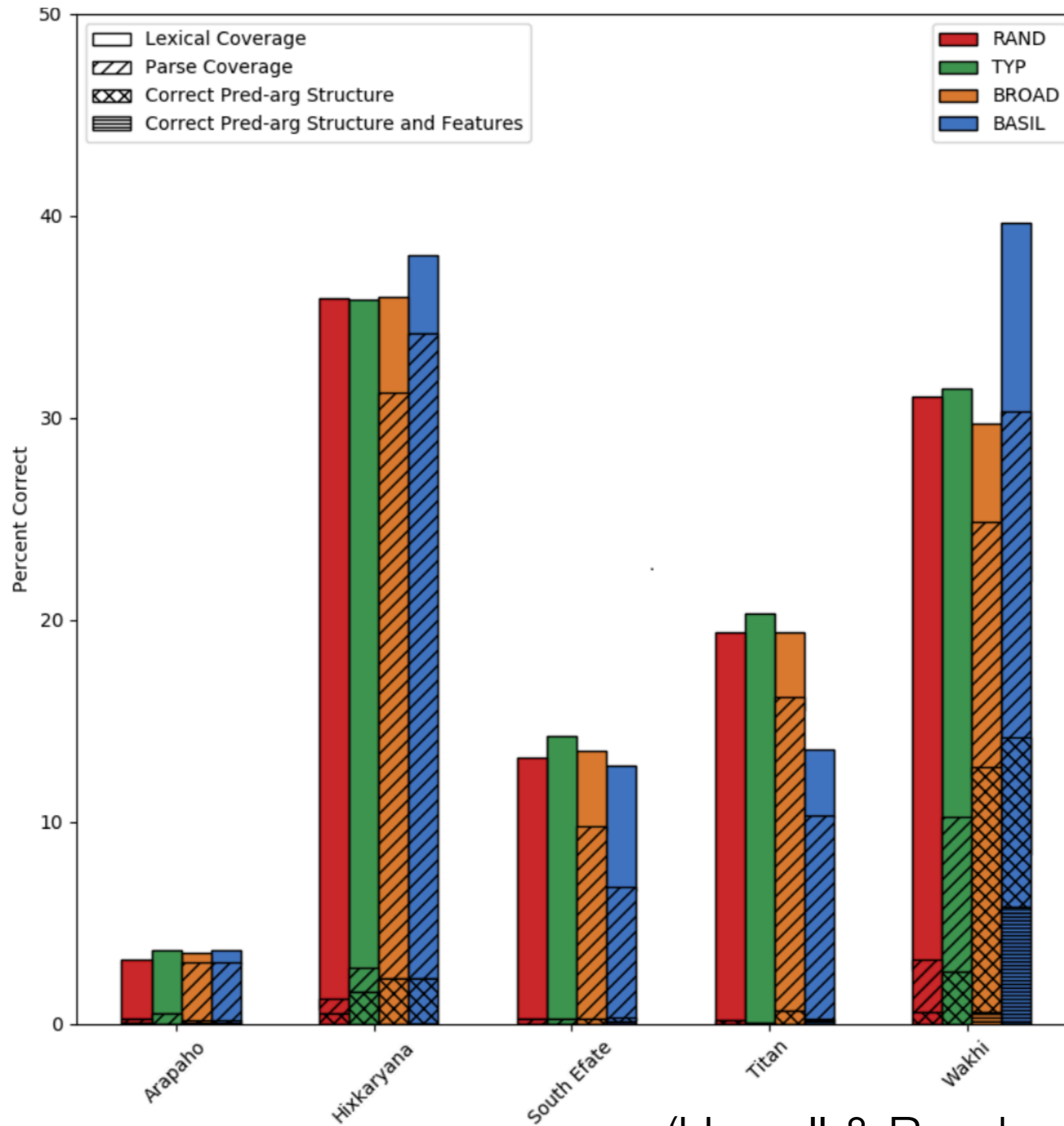
- Precision implemented grammars are a kind of structured annotation over linguistic data (cf. Good 2004, Bender et al 2012).
- They map surface strings to semantic representations and vice-versa.
- They can be used in the development of *grammar checkers* and *treebanks*, making them useful for language documentation and revitalization (Bender et al 2012)
- But they are expensive to build.
- The AGGREGATION project asks whether existing products of documentary linguistic research (IGT collections) can be used to bootstrap the development of precision implemented grammars.

# AGGREGATION: Recent developments

---

- See LIFT 2019 slides
- Plus: Conrad 2021, Dodds 2022, Lin forthcoming
- Howell & Bender 2022





(Howell & Bender 2022: 26)

# Overview

---

- Grammar Matrix customization system
- Questions from Lab 1
- AGGREGATION
- Testsuites & [incr tsdb()]
- Morphotactics in the Grammar Matrix

# Evaluation and Computational Linguistics

---

- Why is evaluation so prominent in computational linguistics?
- Why is it not so prominent in other subfields of linguistics?
- What about CS?

# Intrinsic v. extrinsic evaluation

---

- Intrinsic: How well does this system perform its own task, including generalizing to new data?
- Extrinsic: To what extent does this system contribute to the solution of some problem?
- Examples of intrinsic and extrinsic evaluation of parsers?

# Test data

---

- Test suites
  - Hand constructed examples
  - Positive and negative examples
  - Controlled vocabulary
  - Controlled ambiguity
  - Careful grammatical coverage

# Test data

---

- Test corpora
  - Naturally occurring
  - More open vocabulary
  - Haphazard ungrammatical examples
  - Application-focused

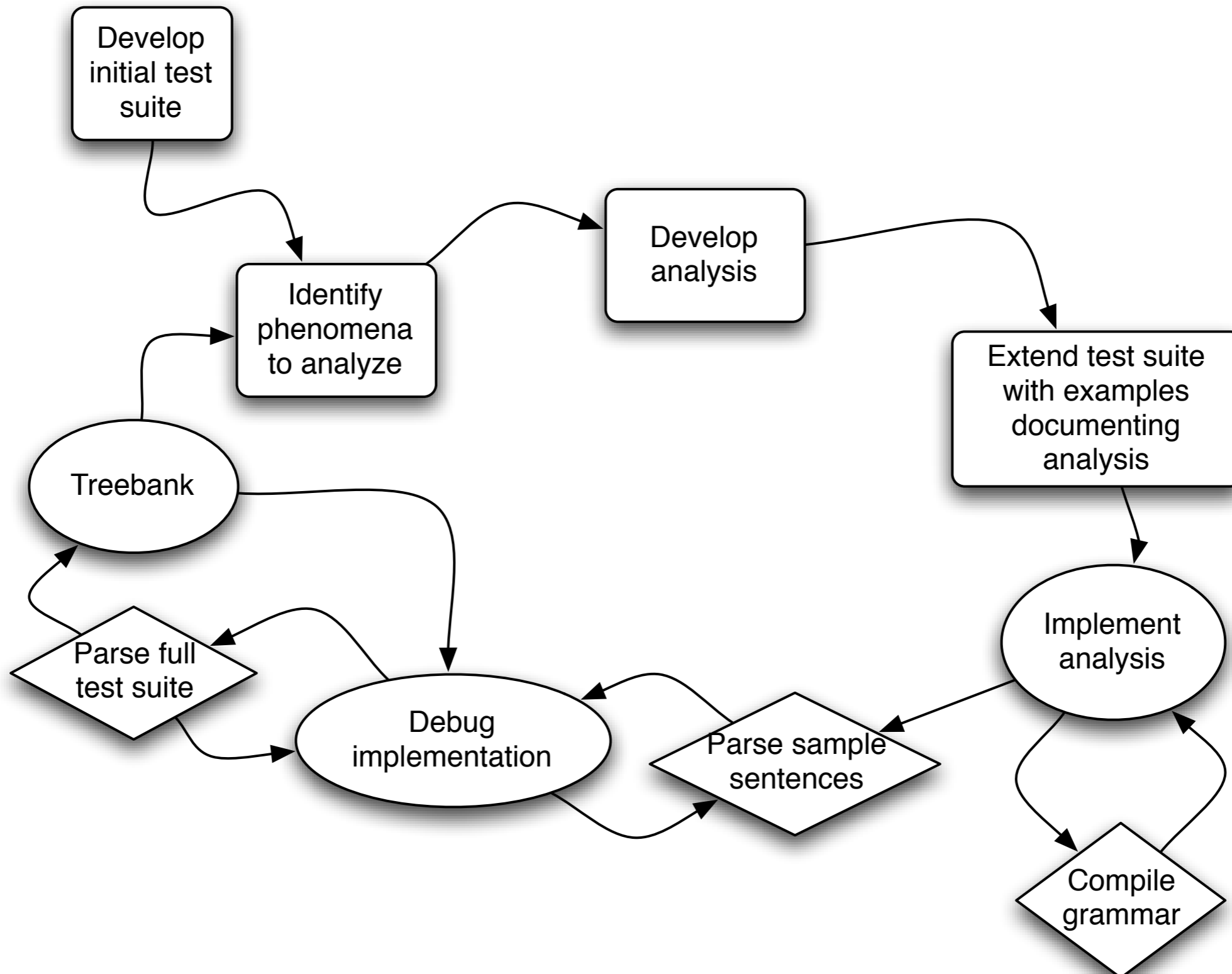
# Uses of test data

---

- How far do I have left to go?
  - Internal metric
  - Objective comparison of different systems
- Where have I been?
  - Regression testing
  - Documentation

# Grammar engineering workflow

---





# Evaluating precision grammars

- Coverage over some corpus
  - Which corpus?
  - Challenges of lexical acquisition
- Coverage of phenomena
  - How does one choose phenomena?
- Comparison across languages

# Levels of adequacy

- grammaticality
- “right” structure
- “right” dependencies
- “right” full semantics
- only legit parses (how can you tell?)
- some set of parses including the preferred one
- preferred parse only/within first N

# Typical 567 test suites

- Map out territory we hope to cover
- Include both positive and negative examples
- Serve as an exercise in understanding the description of the language
  - IGT format
  - Creating examples where necessary

# On the importance of simple examples

- Why keep examples simple?
- How simple is too simple?
- What kinds of things make an example not simple enough?

# On the importance of simple examples

- Awtuw [awt] (Feldman 1986:67)

(70) Yowmen Yawur du-k-puy-ey  
Yomen Yawur DUR-IMPF-hit-IMPF  
'Yowmen and Yawur are hitting (someone).' [awt]

- Basque [eus] (adapted from Joppen and Wunderlich 1995:129)

(112) Zuek lagun-ei opari polit-ak ema-ten dizkiezue.  
you.PL.ERG friend-PL.DAT present nice-PL.ABS give-IMPF 3A.have.PLA.3PLD.2PLE  
'You(pl) always give nice presents to your friends.' [eus]

# On the importance of simple examples

- Russian [rus] (Bender 2013:92)

a. Человек            укусил            собаку.  
Chelovek            ukusi-l            sobak-u.  
man.NOM.SG.M bite-PAST.PFV.SG.M dog-ACC.SG.F  
'The man bit the dog.' [rus]

# But this year we have test corpora!

- Might include both elicited and naturally occurring examples
- Lots more data to play with (yay!)
- Will be messy: Spoken language, lots of interacting phenomena, possibility of inconsistent transcription & glossing
- But more satisfying because it's way more authentic
- Possibly too large: Okay to cut down or break into smaller chunks if processing is slow
  - Possibly consider using ace & art for batch processing

# [incr tsdb()] basics

- [incr tsdb()] stores test suite profiles as (plain text) relational databases: Each is a directory with a fixed set of files in it.
- Most files are empty.
- A profile that has not been processed has only two non-empty files: item (the items to be processed) and relations (always the same)
- Once the profile has been processed, the result of the processing is stored in some of the other files (in particular, parse and result)



# [incr tsdb()] basics

- A test suite *skeleton* consists of just the item and relations files and can be used to create new test suite profiles
- [incr tsdb()] allows the user to compare two profiles to see how they differ
- It can also produce graphs plotting summary data from many profiles to visualize grammar evolution over time
- -> If time: Demo

# Overview

---

- Grammar Matrix customization system
- Questions from Lab 1
- AGGREGATION
- Testsuites & [incr tsdb()]
- Morphotactics in the Grammar Matrix

# Morphology: Basics

---

- Morpheme: The smallest meaningful unit of language/smallest pairing of “form” and “meaning”
- But:
  - “form” can be lots of things, including empty but also messy changes to word form
  - “meaning” can be just syntactic features
- Morphotactics: Which morphemes can combine, in what order
- Morphophonology: Relationship between underlying word forms and surface forms
- Morphosyntax: Relationship between morphemes and syntactic and semantic features

<b>2</b>	<b>Morphology: Introduction</b> . . . . .	<b>11</b>
	#7 Morphemes are the smallest meaningful units of language, usually consisting of a sequence of phones paired with concrete meaning. . . . .	11
	#8 The phones making up a morpheme don't have to be contiguous. . . . .	11
	#9 The form of a morpheme doesn't have to consist of phones. . . . .	13
	#10 The form of a morpheme can be null. . . . .	13
	#11 Root morphemes convey core lexical meaning. . . . .	14
	#12 Derivational affixes can change lexical meaning. . . . .	16
	#13 Root+derivational affix combinations can have idiosyncratic meanings. . . . .	17
	#14 Inflectional affixes add syntactically or semantically relevant features. . . . .	18
	#15 Morphemes can be ambiguous and/or underspecified in their meaning. . . . .	19
	#16 The notion 'word' can be contentious in many languages. . . . .	20
	#17 Constraints on order operate differently between words than they do between morphemes. . . . .	21
	#18 The distinction between words and morphemes is blurred by processes of language change. . . . .	22

	#19 A clitic is a linguistic element which is syntactically independent but phonologically dependent. ....	23
	#20 Languages vary in how many morphemes they have per word (on average and maximally).....	24
	#21 Languages vary in whether they are primarily prefixing or suffixing in their morphology. ....	25
	#22 Languages vary in how easy it is to find the boundaries between morphemes within a word. ....	26
<b>3</b>	<b>Morphophonology</b> .....	<b>29</b>
	#23 The morphophonology of a language describes the way in which surface forms are related to underlying, abstract sequences of morphemes. ....	29
	#24 The form of a morpheme (root or affix) can be sensitive to its phonological context. ....	29
	#25 The form of a morpheme (root or affix) can be sensitive to its morphological context. ....	31
	#26 Suppletive forms replace a stem+affix combination with a wholly different word. ....	32
	#27 Alphabetic and syllabic writing systems tend to reflect some but not all phonological processes. ....	33
<b>4</b>	<b>Morphosyntax</b> .....	<b>35</b>
	#28 The morphosyntax of a language describes how the morphemes in a word	

# Morphology: Example

---

slolmáyaye

slol-ma-ya-yÁ

know-1SG.PAT-2SG.AGT-know

‘you know/knew me’ [lkt]

- Infixation, vowel harmony: Morphophonology
- Relative order of PAT and AGT marker, optionality of same: Morphotactics
- Mapping to constraints that the patient argument be 1sg and the agent 1pl: Morphosyntax
- Actually parsing the string: priceless!

What morphophonology can the LKB & the customization system handle?

	LKB	Customization System
polite concatenative morphology	✓	✓
zero morphemes	✓	✓
morphologically conditioned allomorphy	✓	✓
phon. changes at morpheme boundary	✓	
ablaut		
infixation		
vowel harmony		
suppletion		

# Assume a morphophonological analyzer...

---

- Morphophonological analyzers map surface forms to underlying strings of morphemes
- FSTs are up to the task (except for open-class reduplication)
  - XFST (Beesley & Karttunen 2003) is a very linguist-friendly set up; FOMA (Holden & Algeria 2010) is an open-source package with similar functionality
- But you don't need to build one for this class!
- Use the morpheme segmented line of your IGT to represent what it would map to, and then (if you have any interesting morphophonology) have that line be the target for your grammar.



# Morphophonology/morphosyntax boundary: Where to draw the line?

---

- Underlying morphemes can be represented as a sequence of phonemes or as symbols representing morphological features.
- A canonical XFST-derived analyzer will also include POS tags as a morphological feature in the underlying form.
- From the point of view of the LKB:
  - The POS tag adds nothing
  - Spelling the morphemes as morphological features adds nothing: we still need a lexical rule that maps those strings to constraints on avms

# Morphophonology/morphosyntax boundary: Where to draw the line?

---

- On the other hand: for XFST/FOMA, the POS tags (and maybe features) can be useful intermediate stages in processing
- The features can make it easier to create gloss lines automatically.
- On the third hand: using sequences of morphemes might make LKB input/output comprehensible to speakers
- So what should the upper tape have?

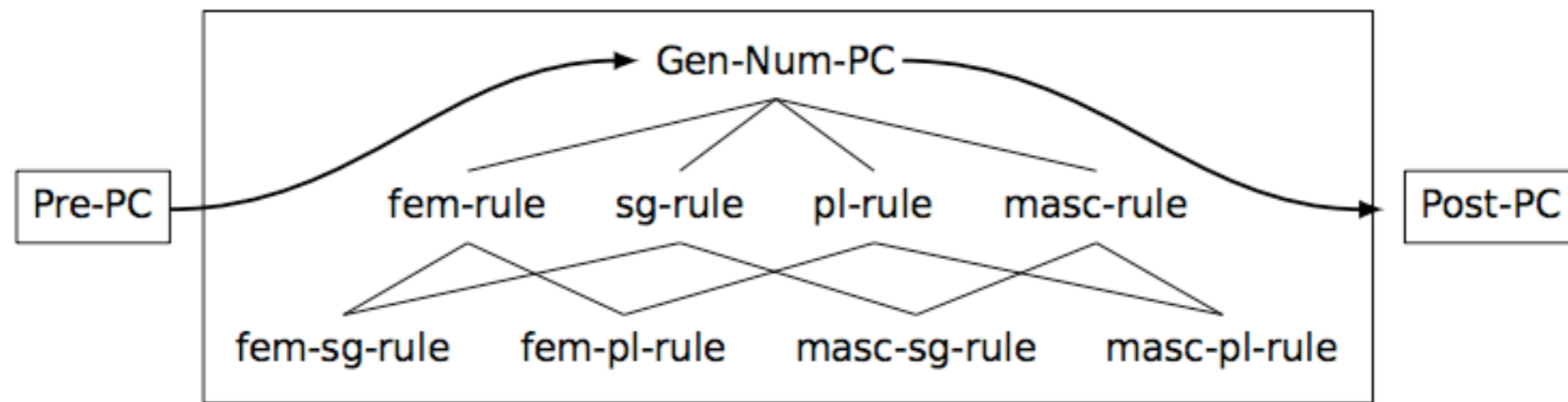
# Basic concepts

---

- Position class: A supertype to lexical rules which fit in the same slot
- Lexical rule type: *lex-rule* and its subtypes, all have DTR feature
- Lexical rule instance: A grammar entity (manipulatable by the LKB) which inherits from a lexical rule type and specifies a spelling change (including no change).
- Forbids constraint: A specification in the customization system stating that a stem lexical rule type (including a position class) cannot co-occur with another lexical rule type, instance, pc or stem.
- Requires constraint: A specification in the customization system stating that a stem lexical rule type (including a position class) must co-occur with another lexical rule type, instance, pc or stem.

# Position classes, inputs and lexical rule hierarchies

---



**Figure 9:** Example lexical rule type hierarchy in a position class

(Goodman 2013)

# To define a position class

---

- Required:
  - Whether or not it is obligatory
  - Possible inputs and prefix/suffix
    - = position in the string
- Optional:
  - Requires/forbids constraints

# To define a lex rule type

---

- Required
  - Nothing (though defaults fill in)
- Optional
  - Name
  - Supertype (if it doesn't inherit directly from its position class)
  - Feature/value pairs (optional, but this is usually the point!)
  - Requires/forbids constraints

# To define a lex rule instance

---

- Required
  - Affix v. no affix
  - Spelling for affix
- Optional
  - Nothing

# tdl files

---

- matrix.tdl: Supertypes for lex-rules, which handle the copying up of everything you're not changing
- my\_language.tdl: Position classes and lex rule types defined through the customization system; features for inside INFLECTED
- lrules.tdl: Instances for non-spelling-changing lex rules (zero morphemes)
- irules.tdl: Instances for spelling-changing lex rules



# Handling of morphotactics

---

- Rule order handled through super types and typing the DTR feature
- Requires/forbids through the INFLECTED feature

```
case-lex-rule-super := representative-rule-dtr &
                    add-only-no-ccont-rule &
                    noun-telic-rule-dtr &
[ INFLECTED [ CASE-FLAG +,
              INNER-NEGATION-FLAG #inner-negation,
              NUMBERED-FLAG #numbered ],
  DTR case-rule-dtr &
  [ INFLECTED [ INNER-NEGATION-FLAG
                #inner-negation,
                NUMBERED-FLAG #numbered ] ] ].
```

# Lab 2 notes

---

- Post questions by Tuesday night
- Hover over red asterisks
- `[incr tsdb()]` woes:
  - Do format `Index.lisp` precisely as shown
  - Don't put `tsdb/` in a shared folder
  - Don't let Windows corrupt your item files
  - `LD_LIBRARY_PATH`—see Canvas

# Thursday = demo day

---

- Send me questions by noon on Thursday; all should include:
  - Question
  - Choices file
  - Data:
    - Testsuite profile
    - IGT that should parse if we can just fix the thing
    - ... or should stop parsing, if we can just fix the thing, in the case of ungrammatical examples