

Learned Conflation Policies

From Lexical Transformation to Reproducible Retrieval

A formal, empirical and historical synthesis of the
RADIXOR research programme and its EGOTHOR lineage

Leo Galambos

Independent research

2026

Learned Conflation Policies: From Lexical Transformation to Reproducible Retrieval.

A synthesis of the RADIXOR research programme.

Copyright © 2026 Leo Galambos. All rights reserved.

The RADIXOR software described in this volume is distributed under the BSD-3-Clause licence. Project documentation, protocol definitions, benchmark identities, result tables and reproduction material are published at <https://leogalambos.github.io/Radixor/>.

This volume synthesises nine research manuscripts listed in Appendix D. Their availability as preprints does not imply journal or conference peer review. Experimental results are restated with the population, resource, implementation and interpretation boundary of the source study; cross-study synthesis introduced by this volume is identified separately.

Provenance. The technical line begins with the EGOTHOR work published in 2001 and is continued here as independent research. The volume was prepared in a personal capacity and contains no third-party confidential material.

Typeset by the author in L^AT_EX with Palatino, TikZ/PGF and algorithm2e.

Abstract

Stemming is usually compared as if the treatment were identified by a program name. That convention is adequate only when the program denotes a fixed mapping. A trainable or adaptable stemmer exposes an additional experimental variable: the lexical policy that determines which forms are permitted to share retrieval evidence. Once that policy is compiled, its observable effect also depends on the ranker, the query population, the optimisation criterion and the rank depth at which retrieval is measured. This monograph develops the consequence of that observation: an adaptable stemmer should be reported as a *realised conflation policy*, not merely as a software label.

The argument is built from nine constituent studies in the Radixor research programme and from the earlier Egothor line on which Radixor is based. The technical object is a lexical transformation system. A finite relation between surface forms and selected targets is converted into patch commands; a trie selects a command from the input context; reduction compiles the decision structure; and the selected command executes on the input itself. The model is therefore trained from a lexicon without degenerating into closed-set lookup. This distinction motivates a sequence of questions that the nine studies answer at different layers: what compilation preserves, which induction choices change the learned object, how lexical evidence shapes the command vocabulary, whether transformations transfer beyond training forms, how competing conflation errors should be compared, what runtime cost follows the compiled structure, whether an alternative reference policy can be learned, how a standard model can be adapted and verified in both directions, and finally how the realised policy interacts with retrieval.

Two contributions are independent of the Radixor implementation. First, conflation evaluation is formulated directly over pair relations, so reference memberships may overlap rather than being forced into a partition. Second, multi-objective comparison is represented by recursive Pareto strata together with exact support intervals for an explicit false-split/false-merge exchange rate. These constructions require only a reference relation and the relation realised by each candidate system. They are therefore evaluation methods, not instrumentation specific to one stemmer.

The system-specific results establish a narrower empirical record. The compiler semantics separate signature-based canonicalisation, which preserves the declared observable, from accepting-leaf contraction, which deliberately extends an already uniform command decision to unseen continuations. Edit costs act as structural hyperparameters, although many nominal settings collapse to exactly the same induced behaviour. A multilingual family-holdout campaign shows that learned commands are applied successfully to forms absent from the training surface set. A resource-consistency audit shows that lexical evidence and learned command vocabulary are connected through support structure rather than by a one-row/one-command correspondence. Runtime measurements associate cost

more strongly with compiled structure than with the number of training rows in the measured census. A controlled German experiment shows that the same mechanism can realise a different conflation objective when the training relation changes. A two-sided adaptation workflow then adds both positive assignments and negative membership boundaries and verifies the realised policy over a declared inventory.

The downstream English MIRACL study provides the final link but also the strongest validity boundary. At a common BM25 setting, a task-adapted Radixor policy improves all five recorded mean metrics over the standard Radixor policy and over PyStemmer in that development experiment. The change is not a scalar increase in stemming strength: the final index has fewer output terms and more term–document incidences. Separate scorer experiments show that the preferred BM25 coordinates and objective can move with the lexical representation. The result demonstrates that policy and ranker form a coupled experimental condition; because policy selection and reporting use the same development judgments, it does not establish out-of-sample superiority.

The monograph consequently distinguishes three kinds of claim. Formal properties are proved for the specified model and compiler. Empirical properties are reported only for the frozen populations in the constituent studies. Field-level conclusions are stated as testable methodological proposals: same-resource ancestor and comparator studies, independent lexical references, matched-budget adaptation comparisons and out-of-sample multilingual retrieval remain the decisive external validation. The resulting research programme is therefore not a claim that one stemmer settles the field; it is a framework for making adaptable conflation systems scientifically identifiable, comparable and reproducible.

Keywords: stemming, conflation, lexical policy, patch commands, transformation learning, trie reduction, compiler semantics, understemming, overstemming, Pareto stratification, model adaptation, information retrieval, BM25, EGOthOR, RADIXOR.

Preface

Why this book exists

The nine papers behind this volume were written as bounded experiments. That was useful for scientific control but poor for showing the architecture of the research programme. A reader of one paper sees a compiler theorem, a multilingual experiment, a resource audit or a retrieval result. The common question only becomes visible when the studies are placed in dependency order: *what exactly is the object that a trainable stemmer learns, how can that object be measured, and what changes when the object itself becomes a tunable component of retrieval?*

This book is a new synthesis around that question. It does not reproduce the papers chapter by chapter, and the paper numbers do not define the reading order. The programme is reconstructed from the technical dependencies among the studies. Compiler semantics, induction sensitivity and lexical support structure establish the object. Generalisation, relation-based evaluation, Pareto analysis and runtime characterise its behaviour. Policy-transfer, two-sided adaptation and retrieval then show what becomes possible once the object is treated as a policy rather than as an immutable language package.

A second reason is practical and more recent. The research line was reopened while considering EGOTHOR v4 as a deterministic, configurable lexical layer for high-assurance, highly regulated and security-sensitive information systems. Such a component must be able to encode language- and domain-specific terminology, protected exceptions and explicit policy versions while remaining reproducible under a frozen release. This is not an alternative to semantic AI. Modern search systems increasingly combine lexical and vector retrieval; Chapter 1 therefore begins by asking what role a controlled conflation policy can play inside that hybrid architecture.

Central proposition. For an adaptable stemmer, the executable conflation function is part of the experimental treatment. Reporting only the program name hides a variable that can alter both intrinsic conflation behaviour and downstream retrieval. A reproducible experiment therefore identifies the policy specification, its compiled realisation, and—for retrieval—the ranker, objective, evaluation depth and selection/evaluation relationship.

Method and evidence are deliberately separated

The volume contains two kinds of contribution. Chapters 10 and 11 define methods that do not depend on Radixor. Their inputs are reference memberships and predicted memberships; another stemmer family can be inserted without changing the definitions. By contrast, the results about patch induction, compiler reduction, runtime structure, resource consistency, policy transfer and adaptation belong to the Egothor–Radixor lineage. The book keeps those levels distinct because confusing them would convert a well-supported system result into an unsupported field claim.

This distinction also governs the language used throughout. A theorem is not presented as empirical evidence. A same-resource intrinsic result is not called external linguistic validation. A development-set retrieval gain is not called transfer. A production deployment is evidence of technical continuity, not of comparative effectiveness. These boundaries are part of the argument, not disclaimers added after it.

How the nine studies are used

Each constituent study supplies one link in the chain rather than one chapter of a serial narrative. The formal compiler study defines what reduction preserves. The edit-cost and resource-consistency studies show how choices made before compilation change the learned command system. The held-out-family study tests whether the transformation mechanism reaches beyond memorised surfaces. The Pareto study supplies a comparison language for the two dominant conflation errors. The runtime study asks what the compiled representation costs to execute. The German transfer study tests whether a different reference relation changes the learned policy while the mechanism is held fixed. The adaptation study turns that capability into a deployment workflow with positive and negative constraints. The MIRACL study then asks whether a changed realised policy can change the retrieval result attributed to the engine.

Appendix D is an evidence ledger for these sources. It records the research question, principal result, DOI and the point in this monograph where each result is used. Chapters cite the constituent study when they draw on its frozen experiment; synthesis-level arguments that are new to this volume are identified as such.

Historical scope

The historical chapter has been written to avoid a common visual and conceptual error: chronology is not genealogy. Dates establish priority of publication; mechanistic similarity establishes neither descent nor influence. The revised figure therefore separates verified dated milestones into methodological swimlanes and uses descent arrows only where a continuation is documented. In particular, the Galambos/Egothor publication of 2001, the later Stempel presence in Apache Lucene, the separate Stempelator/Morfologik line, and the much later XSTEM publication are shown without forcing them into one ancestry tree.

How to read the volume

Part I defines the experimental object and places it historically. Part II follows the object from lexical relation through induction, structural choices, resource support and compilation. Part III develops the measurement language and then asks whether the realised policy generalises, where it lies in the under/over-conflation plane, and what it costs to execute. Part IV changes the policy deliberately and follows the consequence into retrieval. Part V separates what the programme now establishes from what still requires external validation.

Readers interested primarily in methodology can begin with Chapters 1, 10, 11, 17 and 18. Readers interested in the executable mechanism should follow Chapters 5, 6, 9 and 15. Readers interested in information retrieval should read Chapter 16 together with its explicit selection boundary rather than as a standalone leaderboard result.

Scope of the conclusion

The book does not claim that Radixor is universally the fastest or most accurate stemmer, that lexical transformation dominates rule-based stemming, or that the reported English retrieval improvement transfers to unseen queries. It claims something more precise and, in my view, more durable: once conflation is adaptable, the realised policy becomes an experimental variable that must be specified, measured and verified. The nine studies show that this variable is structurally meaningful and operationally consequential in one lineage. The next stage is to apply the same experimental discipline across lineages.

Leo Galambos
2026

Contents

Abstract	v
Preface	vii
List of Figures	xvii
List of Tables	xix
Notation and Conventions	xxiii
I Conflation as a Policy	1
1 Why Revisit Stemming? From Legacy Preprocessing to a Controlled Retrieval Policy	3
1.1 Why revisit stemming in the age of semantic retrieval?	3
1.2 The hidden variable in stemming experiments	6
1.3 What must be true before policy can be treated as a variable	6
1.4 The Egothor–Radixor line as an experimental instrument	7
1.5 Nine studies, one dependency chain	8
1.6 Two levels of contribution	8
1.7 Boundaries that are part of the result	9
1.8 Reading order	9
2 Conflation Policy as the Treatment Variable	11
2.1 The retrieval object is an equivalence decision	11
2.2 Stemming, lemmatisation and morphological analysis are not synonyms . .	12
2.3 A policy is a declared reference relation	12
2.4 The error surface has two axes	12
2.5 Why overlapping groups matter	13
2.6 Three registers of evidence	14
2.7 Why policy, not algorithm name, is the unit of comparison	15
3 Lineages, Parallels, and Verified Chronology	17
3.1 What the diagram is allowed to claim	17
3.2 The IR rule tradition	17
3.3 The documented Egothor–Stempel–Radixor line	18
3.4 Stempelator and Morfologik: a neighbouring line, not a predecessor	18

3.5	Parallel learned-transformation traditions	19
3.6	Chronology is not a novelty verdict	19
3.7	A mechanism matrix instead of a single family tree	19
4	From Egothor to Radixor: What Was Inherited and What Changed	23
4.1	The 2001 design decision	23
4.2	Patch commands as the inherited instruction language	24
4.3	The first study already treated configuration as a variable	25
4.4	Stempel as continuity evidence	25
4.5	What Radixor changes	26
4.6	A historical line becomes an experimental platform	27
II	Learning and Compiling the Policy	29
5	The Executable Policy Object	31
5.1	Lexical evidence	31
5.2	Patch commands are programs, not stored answers	31
5.3	The command-selection structure	32
5.4	Selection and realised output are different functions	33
5.5	Reduction mode is part of the model condition	33
5.6	Object ledger	34
6	From Lexical Pairs to Reusable Transformations	35
6.1	Construction pipeline	35
6.2	Minimum-cost alignment	36
6.3	Normalising an edit path into a command	36
6.4	Indexing commands by context	37
6.5	Determinism and row order	38
6.6	What induction guarantees—and what it cannot	38
7	When Alignment Parameters Become Model Structure	39
7.1	The experimental object	39
7.2	Exact behavioural equivalence	40
7.3	Why a continuous-looking parameter space collapses	40
7.4	Sensitivity of quality inside the sampled grid	41
7.5	Structural slack is larger than quality sensitivity	41
7.6	Consequences for the rest of the programme	42
8	Resource Consistency as a Compiler Input	43
8.1	The row-to-command map is many-to-one	43
8.2	Audit protocol	43
8.3	Observed structural effect	44
8.4	The informative cases are the non-linear ones	45
8.5	What the reduction means—and what it does not mean	46
8.6	Why this chapter precedes compiler semantics	47

9	Two Compiler Operations, Two Semantic Contracts	49
9.1	Canonicalisation: quotienting without changing selection	49
9.2	Accepting-leaf contraction: generalising a decision	51
9.3	The generalised object is a command selection, not a stored stem	51
9.4	Implementation verification accompanies the proof	52
9.5	What P7 contributes to the programme	53
III	Measuring the Realised Policy	55
10	Evaluating Conflation as a Relation	57
10.1	Reference covers can overlap	57
10.2	Predicted relation and pairwise counts	58
10.3	Metrics are views of the same finite counts	58
10.4	What the finite multilingual census shows about metric choice	58
10.5	Candidate-aware outputs change the relation, not merely the score	59
10.6	Output policy must be declared	60
10.7	Comparison rules before statistics	60
10.8	Inference follows the experimental unit	60
10.9	Why the scalar is postponed	61
11	Pareto Geometry of Conflation Errors	63
11.1	Dominance is the comparison that needs no exchange rate	63
11.2	The measured candidate set has four strata	63
11.3	Radixor shifts the within-reference frontier	64
11.4	Frontier position has different meanings	65
11.5	Making the exchange rate explicit	66
11.6	Candidate-set dependence is measured, not hidden	66
11.7	Why an external floor is still needed	67
12	Generalisation Beyond Enumerated Lexical Families	69
12.1	Protocol: family-level holdout	69
12.2	Three evaluation scopes	70
12.3	Result: measurable within-resource transfer	70
12.4	What the study does not establish	71
12.5	Why the result matters to the programme	71
13	Runtime Cost of the Compiled Policy	73
13.1	Measured execution envelope	73
13.2	Direct Snowball comparison	73
13.3	Training size and online structure are different quantities	74
13.4	Runtime is not a proxy for lexical quality	75
13.5	Why runtime closes the measurement part	75

IV	Adapting the Policy to Retrieval	77
14	A Controlled Test of Policy Learnability	79
14.1	A nested policy contrast	79
14.2	The architecture is held fixed	79
14.3	Result: small, directional policy response	80
14.4	Why the result is different from ordinary tuning	80
14.5	From learned policy to deployable policy	81
15	From Policy Edits to a Verified Deployment	83
15.1	A standard model is a substrate, not a ceiling	83
15.2	Positive assignments are only half of the policy	83
15.3	The measured reason closure is necessary	84
15.4	Representation, learning and deployment fidelity	84
15.5	Comparison with later configurable mechanisms	85
15.6	The bridge to retrieval	86
16	Where Retrieval Quality Goes	87
16.1	Experimental setting	87
16.2	Adaptation improves the standard model in the reported case	87
16.3	The aggregate gain is concentrated on a minority of queries	88
16.4	Vocabulary reduction and query work do not explain the gain	89
16.5	The representation change is not one-dimensional	89
16.6	The ranker optimum is endogenous to the policy	90
16.7	The existing recall means locate where judged relevance moves	91
16.8	The preferred fixed policy depends on the declared objective	91
16.9	Different depths answer different product questions	92
16.10	The decisive next retrieval experiment	92
V	What the Programme Establishes	93
17	What the Nine Studies Establish Together	95
17.1	The programme as a chain of identifiability	95
17.2	Three levels of claim	96
17.3	The reusable methodological contribution	96
17.4	A coupled view of lexical normalisation and ranking	97
17.5	What the programme does not yet establish	97
17.6	The revised thesis statement	98
17.7	What would promote the thesis from framework to field result	98
18	External Validation Programme	99
18.1	C4: turn the framework outward	99
18.2	C1: independent lexical evidence	99
18.3	C2: held-out multilingual retrieval	100
18.4	C3: establish an absolute intrinsic floor	101
18.5	C5 and C6: workload and inference	101
18.6	C7: fixed-test learning curves	101

18.7	How the nine studies should be counted	101
18.8	Prioritised programme	102
18.9	The programme is open by design	102
19	Conclusions	103
19.1	A policy can be made explicit	103
19.2	A policy can be learned into an executable object	103
19.3	A learned policy can be changed and checked	104
19.4	A verified policy can change retrieval	104
19.5	What is transferable beyond Radixor	104
19.6	What remains to earn a field-level empirical claim	105
19.7	Final statement	105
VI	Appendices	107
A	Notation, with Cross-References	109
B	Metric Definitions and Applicability	111
B.1	Confusion-derived metrics	111
B.2	Applicability by output policy	112
B.3	Metrics deliberately not computed	113
B.4	Aggregation rules	113
C	Protocol Identities and Reproducibility	115
C.1	Provenance discipline	115
C.2	Declared seeds	115
C.3	Generalisation campaign scale	115
C.4	Reproduction entry points	116
C.5	Edit-cost campaign	116
C.6	Quality campaign	117
C.7	Where the material lives	117
D	Evidence Ledger: The Nine Constituent Studies	119
	How the studies combine	122
	Bibliography	123

List of Figures

1.1	Construction, adaptation and verification	7
1.2	Logical spine of the nine constituent studies	8
2.1	The two-axis conflation error surface	13
3.1	Verified chronology of relevant stemming and transformation lines	21
4.1	Patch execution with a preserved interior	24
5.1	Patch-command trie indexed by reversed form	33
6.1	Alignment path to reusable command	37
7.1	Nominal parameter space versus induced behaviour	40
8.1	Ordered audit evidence path	44
8.2	Command vocabulary retained after resource audit	46
9.1	Semantic subtree canonicalisation	51
9.2	Accepting-leaf contraction extends a command decision	52
10.1	Overlapping lexical groups induce a pair relation	57
11.1	Recursive Pareto strata in the two-error plane	64
12.1	Construction of the primary generalisation population	70
13.1	Observed Radixor execution envelope	73
14.1	Controlled policy-transfer design	80
15.1	Two-sided policy adaptation	84
16.1	Index statistics exclude a scalar “stemming strength” ordering	90

List of Tables

1.1	Operational motivation for a configurable lexical layer	5
1.2	The nine constituent studies	10
1.3	Contribution levels	10
2.1	Three registers of evidence	14
3.1	Mechanism comparison across relevant lines	20
4.1	Patch command primitives	24
4.2	Continuity and extension from Egothor to Radixor	26
5.1	Objects used throughout the monograph	34
7.1	Scale of the edit-cost campaign	39
7.2	What the edit-cost study establishes	41
8.1	Resource-consistency audit outcomes	45
8.2	Mechanistic cases from the P6 resource audit	45
9.1	Formal compiler results and their interpretation	53
10.1	Relation-level metrics	59
10.2	Metric choice changes the observed best comparator set	59
11.1	Nearest-comparator dominance margins	64
11.2	Interpretive regimes on the Pareto frontier	65
11.3	German stratum-2 pair-cost support	65
12.1	Generalisation evaluation scopes	70
13.1	Runtime campaign summary	74
15.1	Native transformation and deployment distinctions	85
16.1	Reported development-set retrieval changes	88
16.2	Final full-corpus MIRACL development means	88
16.3	Paired query diagnostics for sealed Radixor vs PyStemmer	89
16.4	Vocabulary and deterministic query-work census	89
16.5	Depth-band decomposition of reported recall means	91

17.1	Three levels of contribution	96
18.1	Roles of external comparators	100
18.2	Prioritised external-validation programme	102
B.2	Metric applicability by output policy	112
C.1	Generalisation campaign artefacts	116

List of Algorithms

5.1	$\text{Exec}(c, w)$ – execute a patch command	32
5.2	$H(w)$ – runtime lookup and command execution	34
6.1	$\text{BUILD}(\mathcal{L}, \gamma, \mu)$ – induce and compile a model	35
6.2	$\text{ALIGN}(w, r, \gamma)$ – minimum-cost edit sequence	36
6.3	$\text{INSERT}(\mathcal{T}, \overleftarrow{w}, c)$ – add command evidence	37
9.1	$\text{CANONICALISE}(\mathcal{T}, \mu)$ – share equal semantic subtries	50
9.2	$\text{CONTRACT}(\mathcal{T})$ – replace uniform represented regions	52
11.1	$\text{STRATIFY}(S)$ – dominance depth and linear-cost support	66
12.1	Family-holdout evaluation used by P1	69
15.1	Two-sided adaptation with observed-vocabulary verification	84

Notation and Conventions

The table below fixes the symbols used throughout. Appendix A repeats it with cross-references to the definitions in which each symbol is introduced.

Symbol	Meaning
Σ	finite alphabet of the language resource
Σ^*	set of finite strings over Σ
w, u, v	surface forms (word types), elements of Σ^*
$\overleftarrow{w}$	the reversal of w ; suffix-oriented models are indexed on $\overleftarrow{w}$
$\mathcal{L}$	the lexical training relation, a set of (form, root) pairs grouped into rows
$r(w)$	the root recorded for w in $\mathcal{L}$
$G(w)$	the set of gold groups of which w is a member
$\mathcal{C}$	the learned patch-command vocabulary
c	a single patch command, a sequence of edit operations
γ	the edit-cost vector $(\gamma_{\text{ins}}, \gamma_{\text{del}}, \gamma_{\text{rep}}, \dots)$
$\mathcal{T}$	the patch-command trie
$\mathcal{N}(\mathcal{T})$	node set of $\mathcal{T}$; $ \mathcal{N}(\mathcal{T}) $ is the internal node count
$\mathcal{F}(\mathcal{T})$	accepting leaves of $\mathcal{T}$
$\text{sig}(n)$	the computable reduction signature of node n
$\text{val}(n)$	the value (selected command) of a locally uniform region rooted at n
$\text{dom}(n)$	the set of continuations reaching n
$C(w)$	the command-selection function; returns $c \in \mathcal{C}$
$\text{Exec}(c, w)$	execution of command c on form w
$H(w)$	the realised stemming function, $H(w) = \text{Exec}(C(w), w)$
π	a lexical policy: the pair (training relation, adaptation set) realised by a model
TP, FP, FN, TN	pairwise confusion counts over the gold pair relation
UI	understemming index, $FN / (TP + FN)$
OI	overstemming index, $FP / (TN + FP)$
BA	balanced accuracy, $\frac{1}{2}(\text{recall} + \text{specificity})$
C_ρ	raw pair cost, $C_\rho = \rho FN + FP$
ρ	the caller's exchange rate between false splits and false merges
nDCG@ k	normalised discounted cumulative gain at rank k
$R@k$	recall at rank k
$AP@k$	average precision to depth k

Conventions. Counts are reported as integers taken from the frozen campaign artefacts; percentages are derived quantities and are shown to the precision at which they were published. A dash or the marker n/a denotes an undefined denominator and never the value zero. Language names refer to the specific model identity used, not to the language in

general: default Polish means the pl-pl-unimorph model and does not include results obtained with pl-pl-polimorf. Where a reduction mode affects a claim, the mode is named. The word *stem* denotes the realised output of H , which is not required to be a well-formed word of the language; the word *root* denotes the canonical form recorded in the lexical resource.

Part I

Conflation as a Policy

Why Revisit Stemming? From Legacy Pre-processing to a Controlled Retrieval Policy

1.1 Why revisit stemming in the age of semantic retrieval?

A practical systems problem reopened this research line. The design target for *EGOTHOR v4* is a lexical analysis layer for environments in which the same input under the same released policy version must produce the same output, where changes to that policy must be reviewable, and where language- and domain-specific behaviour must be introduced without replacing the execution engine. The motivating deployment class includes highly regulated, security-sensitive and high-assurance information systems, but the requirements are more general: deterministic replay, explicit versioning, bounded configuration, local execution, and the ability to protect or introduce terminology deliberately.

That target does not imply that modern search should return to a purely lexical architecture. Contemporary production search systems increasingly make the opposite point. Azure AI Search defines hybrid search as full-text and vector queries executed together and fused into one result set; Elasticsearch exposes hybrid full-text/vector retrieval as a first-class workflow; and OpenSearch likewise combines keyword and semantic queries in one search pipeline (Elastic, 2026; Microsoft, 2026; OpenSearch Project, 2026). Azure's own architecture guidance is especially relevant to the present problem: exact identifiers, product codes, specialised terminology, dates and names are among the cases in which lexical matching retains an advantage even when semantic retrieval is available (Microsoft, 2026).

The modern question is therefore not whether semantic search has made lexical analysis obsolete. It is how a controlled lexical channel should coexist with semantic retrieval. A hybrid system can use lexical evidence for exact and morphologically normalised matching, a vector channel for conceptual and paraphrastic similarity, and a fusion or reranking stage to combine the two. The lexical component remains consequential because it controls an inverted index, term statistics, filters and exact matches that the semantic component does not simply subsume. Chapter 16 provides one concrete instance of the coupling: changing the realised conflation policy changes the term space seen by BM25 and can move the ranker's preferred operating point.

1.1.1 A different optimisation target in high-assurance systems

In an ordinary consumer application, a small relevance gain can justify an opaque model update if the operational risk is low. The optimisation target is different when a system has to support formal change control, forensic review, restricted data movement or repeatable acceptance testing. NIST’s AI Risk Management Framework does not prescribe stemming, and it does not state that a component must be deterministic. It does, however, identify validity and reliability, security and resilience, accountability, transparency, explainability and documentation as trustworthiness concerns; the accompanying Playbook recommends instrumentation, histories and audit records that allow operators to review sources of error and vulnerability (National Institute of Standards and Technology, 2023; Tabassi, 2023). A deterministic lexical analyser is one engineering means of reducing the verification surface for one part of a larger retrieval system.

The distinction matters. Determinism is not compliance, and a deterministic component can still implement a bad policy. Its advantage is narrower: given a frozen implementation, policy package and input, behaviour can be replayed exactly. A policy change can be diffed at the lexical-relation level, compiled again, and subjected to the same acceptance tests. That makes it possible to place the lexical layer under conventional software and configuration management even when later stages use statistical or generative models.

Interpretation boundary. A deterministic stemmer is not a substitute for governance of the complete search or AI system. The claim is architectural: isolating a reproducible and inspectable lexical channel can make one consequential part of a hybrid retrieval pipeline easier to validate, version and audit.

1.1.2 Configuration is the reason to learn a policy rather than ship one

A fixed language stemmer answers one question: which conflations did its author choose for a generic language? The EGOTHOR v4 design problem is broader. A single execution engine should be able to carry different policy packages for languages, organisational terminology, specialist vocabularies, slang and controlled exceptions. In one deployment an abbreviation may have to remain isolated; in another it may be required to join a domain expression. A security product may need to protect identifiers and command names from an otherwise productive suffix rule. A legal or medical corpus may need explicit terminological equivalences that are irrelevant to general-language search.

This requirement changes the meaning of “stemming”. The useful object is no longer a language-wide suffix heuristic but a versioned conflation policy. A compact operational description of such a deployment can be written as

$$\Pi = (L, D, E^+, E^-, \theta, v),$$

where L is the base lexical resource, D the domain extension, E^+ required memberships, E^- protected separations, θ the frozen construction/compiler configuration, and v the policy version. The runtime contract is then a deterministic function $h_\Pi(w)$ for a token w . The notation is introduced here as a systems abstraction rather than a new formal claim: the individual elements are developed by the constituent studies.

In particular, P4 shows that the mechanism can move toward an alternative reference relation while the engine is fixed; P7 defines the compiler semantics needed to reason about

the realised executable object; and P9 shows why positive edits alone are insufficient and adds negative class boundaries plus closure verification. These results make language and domain customisation a property that can be tested rather than an informal promise that “a dictionary can be edited”.

Table 1.1: Why a controlled lexical transformation layer remains useful in a modern hybrid retrieval system. These are design goals and verification advantages, not a claim that stemming by itself satisfies regulatory requirements.

Requirement	Operational reason	Consequence for the lexical layer
Deterministic replay	Acceptance tests and incident reconstruction should reproduce the same lexical decisions	Freeze engine, compiler configuration and policy version
Language/domain packages	General-language morphology is not identical to organisational jargon, slang or specialist terminology	Treat lexical policy as deployable data rather than hard-coded engine behaviour
Protected exceptions	Identifiers, names, abbreviations or regulated terms may be damaged by otherwise productive conflation	Support explicit negative boundaries in addition to positive assignments
Auditable change	Reviewers need to know what changed and which observed forms are affected	Compare realised relations before/after compilation and verify closure over a declared inventory
Local operation	Some security domains restrict external inference services or data movement	Keep the lexical stage executable without a mandatory remote model dependency
Hybrid coexistence	Semantic retrieval improves conceptual recall while exact lexical evidence remains valuable	Preserve a separately measurable lexical channel for fusion or reranking

1.1.3 Why not replace this layer with contextual language understanding?

Contextual lemmatisation and neural semantic retrieval can solve problems that a context-free stemmer cannot. That is not an argument against them; it is a reason to keep task contracts separate. A contextual model consumes a larger information budget, introduces a model version and inference environment into the decision path, and may be unnecessary when the required operation is simply “apply the same declared lexical relation at indexing and query time”. A small deterministic analyser is attractive precisely when the narrower contract is sufficient.

The complementary architecture is therefore asymmetric. The lexical channel should be deliberately boring at runtime: fast, deterministic, inspectable and stable under a frozen policy. Adaptation belongs in an explicit construction and verification process. The semantic channel is free to solve the harder problem of conceptual similarity. Hybrid retrieval then combines signals rather than forcing one representation to serve every purpose.

The motivation for revisiting stemming is not nostalgia for suffix stripping. It is the need for a *controlled lexical equivalence layer*: one that can be configured for language and domain, can express exceptions, can be rebuilt and verified deterministically, and can coexist with vector and generative AI rather than pretending to replace them.

1.2 The hidden variable in stemming experiments

The question “does stemming help retrieval?” has never had one stable answer. English experiments have reported small or collection-dependent changes (Harman, 1991; Hull, 1996); dictionary-constrained stemming has shown benefits under different conditions (Krovetz, 1993); and multilingual work has repeatedly found that morphology, collection and query formulation matter (Braschler and Ripplinger, 2004; Kettunen and Airio, 2006). Those results need not be contradictory. They are estimates of interventions that differ in language, corpus, query set, retrieval model and evaluation measure.

This monograph isolates another source of variation that becomes unavoidable once the stemmer itself is trainable or adaptable. A software package can realise more than one relation over word forms. Changing that relation changes which postings are pooled, which term frequencies accumulate and which document frequencies enter the ranker. In such a system, the program name is an architecture label; it is not a complete description of the experimental treatment.

The distinction is easiest to state in three layers. A *policy specification* says which lexical identities or exclusions are intended. A *compiled realisation* is the executable function produced from that specification and from the compiler’s generalisation rules. A *retrieval condition* places the realised function inside an index and a ranker. The first two can differ while the software name remains unchanged; the third can move even when the lexical function is held fixed.

Reproducibility proposition. For an adaptable conflation mechanism, reporting the program name is insufficient. Intrinsic evaluation should identify the policy specification and the realised conflation function. Retrieval evaluation should additionally identify the ranker, its parameters, the optimisation objective, evaluation depth and whether policy/ranker selection is separated from final reporting.

This proposition is methodological. The nine studies behind the book provide one detailed empirical and formal test bed for it; they do not, by themselves, show how large the same effect is for every stemmer family.

1.3 What must be true before policy can be treated as a variable

Calling a stemming configuration a policy is useful only if the term carries technical content. Four conditions are needed.

The realised object must be defined. A learned representation can share states, choose among competing outputs and terminate before all input characters are consumed. It is therefore not enough to point at a training lexicon. The executable mapping has to be derived from an explicit selection and execution semantics. The formal framework in Chapter 5 and the compiler analysis in Chapter 9 provide that semantics.

The object must be distinguishable from its training data. A system that only memorises observed input–output rows is a compressed dictionary, not a learned transformation mechanism. The held-out-family experiment of P1 (Galambos, 2026a) tests this distinction directly: after excluding training-surface overlap, the compiled commands still transform previously unseen forms with substantial exact-target agreement.

The policy must be capable of changing while the engine remains fixed. P4 (Galambos, 2026d) supplies the controlled evidence. Two German reference relations encode different conflation objectives; holding the mechanism fixed while changing the relation causes the learned system to move in the direction of the changed policy. The effect is small and one-dimensional, but it establishes the relevant capability without appealing to retrieval.

The changed policy must be auditable after compilation. P9 (Galambos, 2026i) separates representational capability, learning capability and deployment fidelity. Positive assignments alone do not certify the realised classes: in the shared English case, the positive-only model admits 11,291 unintended surfaces to protected outputs. Negative membership boundaries and an observed-vocabulary closure check are therefore part of the experimental definition, not implementation decoration.

These four conditions turn “policy” from a metaphor into an object that can be manipulated and reported.

1.4 The Egothor–Radixor line as an experimental instrument

The system used to study those conditions descends from the Egothor work published in 2001 (Galambos, 2001). Its central move is to learn *transformations* rather than to store only target strings. A lexical relation supplies surface/target pairs. Alignment converts a pair into a patch command. Context indexes commands in a trie. Compilation reduces the trie. At runtime, the input selects a command and the command executes on that input.

That last step is the reason the model is lexicon-trained without being lexicon-bound. The compiled structure chooses an operation; it does not merely retrieve a stored answer. Chapter 12 measures the consequence, and Chapter 9 shows where compilation itself can extend a command decision beyond the exact observed continuation.

Figure 1.1 distinguishes initial model construction from later policy adaptation. Initial construction induces transformations from the lexical relation and compiles them into the standard model. Adaptation starts from that compiled model: policy statements are added, the modified structure is recompiled, and only then is the realised policy verified over a declared inventory before deployment.

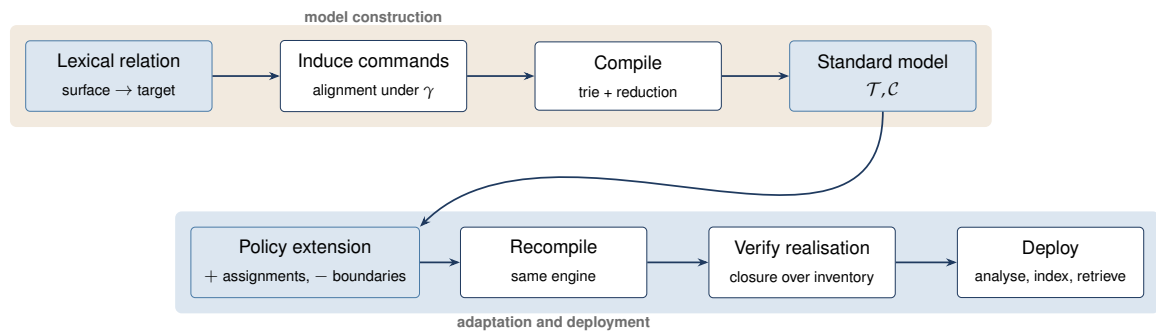


Figure 1.1: Two distinct workflows share the same compilation engine. Initial construction learns a standard model from a lexical relation. Adaptation reopens that model, adds positive and negative policy statements, recompiles, verifies the realised classes over the declared inventory and only then deploys the model. Verification is a post-compilation acceptance test rather than part of lexical induction.

1.5 Nine studies, one dependency chain

The research papers were numbered by release order, not by logical dependency. Reading them as P1 through P9 therefore hides the programme’s structure. A more informative order begins with the executable object, moves through measurement, and ends in retrieval.

1. **P7** (Galambos, 2026g) asks what trie compilation preserves and where it generalises. This supplies the semantics of the executable object.
2. **P2** (Galambos, 2026b) shows that alignment costs are structural hyperparameters and that many nominal settings nevertheless induce exactly the same command sequence.
3. **P6** (Galambos, 2026f) shows how lexical-resource consistency propagates through command support rather than through a one-row/one-command mapping.
4. **P1** (Galambos, 2026a) tests whether the induced transformations reach forms that did not participate in training.
5. **P3** (Galambos, 2026c) supplies the pairwise error space, Pareto strata and cost-support analysis needed to compare realised relations without hiding a false-split/false-merge exchange rate.
6. **P5** (Galambos, 2026e) asks what the compiled object costs at runtime and relates timing more closely to representation structure than to raw training row count in the measured census.
7. **P4** (Galambos, 2026d) changes the reference policy while holding the mechanism fixed and observes a policy-aligned change in output relation.
8. **P9** (Galambos, 2026i) turns policy change into a two-sided deployment operation and introduces verification of protected classes.
9. **P8** (Galambos, 2026h) places the adapted policy in a full retrieval pipeline and measures the interaction with BM25, objective choice and rank depth.

Figure 1.2 renders that dependency chain. It is intentionally not a chronology: its arrows mean “this question becomes interpretable after that one”, not “this paper was published later”.

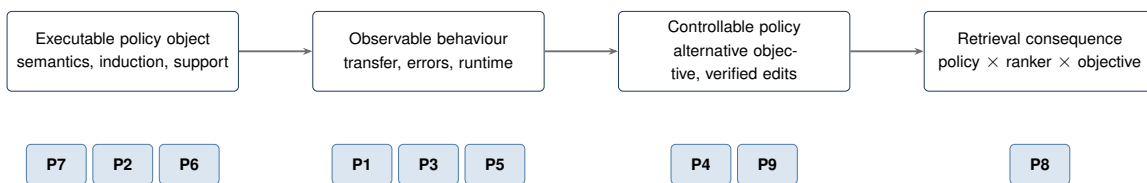


Figure 1.2: The nine constituent studies organised by evidential dependency rather than publication number. The programme first makes the executable policy precise, then characterises its behaviour, then changes and verifies it, and finally measures its retrieval consequence.

Table 1.2 keeps the compact paper-to-chapter index for readers who approach the volume from an individual manuscript.

1.6 Two levels of contribution

A framework can be general in definition while its measured effect remains local. The monograph therefore uses the distinction in Table 1.3 whenever a result is promoted from a

study to an argument about methodology.

The distinction is especially important for Chapters 10 and 11. Their definitions are portable by construction. The fact that they were motivated and exercised on Radixor does not make the measured Pareto positions or effect sizes portable.

1.7 Boundaries that are part of the result

Several statements would make the narrative simpler but the evidence does not support them.

- The intrinsic campaigns are not independent linguistic validation when the resource being scored also supplies training evidence. They measure same-resource agreement under controlled withholding or transformation.
- The runtime campaign does not prove universal speed superiority. It measures a specific reference-changing workload and a limited comparator set.
- The German policy-transfer result is evidence that policy is learnable in one controlled nested contrast; its approximately sparse switching effect is not a general effect size for adaptation.
- The English MIRACL result is development-set feasibility. The same judgements participate in policy selection and final scoring, so the reported gains are not an unbiased estimate on unseen queries.
- Accepting-leaf contraction has a formal command-selection semantics. A proof that it extends a uniform decision does not prove that every newly covered form is linguistically correct.

These are not concessions placed at the end of the argument. They determine which conclusions can be transported from one study to the next. Part V turns them into an experimental agenda rather than allowing them to remain generic “future work”.

1.8 Reading order

Part I treats conflation as an experimental policy and then reconstructs the historical lineages needed to locate the mechanism. Part II follows the policy from formal object through induction, edit-cost choice, lexical support and compilation. Part III introduces system-independent measurement before reporting Pareto structure, held-out-family transfer and runtime. Part IV asks whether the policy can be changed deliberately, verified, and coupled to a retrieval objective. Part V assembles the chain of claims and states the external validation that would move the argument from one technical lineage to the field as a whole.

The order is intentionally different from the paper numbers. The book is not an anthology. It is an argument reconstructed from nine experiments whose scientific value is clearer when their dependencies are made explicit.

Table 1.2: Constituent studies and their principal role in this synthesis. Full titles, DOIs and evidence boundaries are recorded in Appendix D.

Study	Question	Role in the programme	Ch.
P1	Do learned transformations act on genuinely unseen forms?	Controlled within-resource family holdout	12
P2	Do edit costs alter the learned model?	Structural hyperparameter sensitivity and exact behavioural collapse	7
P3	How should false splits and false merges be compared?	Pairwise relation evaluation, Pareto strata and support intervals	11
P4	Can one engine learn a different conflation objective?	Controlled policy-transfer evidence	14
P5	What determines online execution cost?	Compiled-structure/runtime characterisation	13
P6	How does lexical consistency affect learned complexity?	Support-aware resource audit	8
P7	What does the compiler preserve or generalise?	Formal compiler semantics	9
P8	Can a changed policy change the retrieval result?	Full-corpus MIRACL case and ranker/policy interaction	16
P9	Can policy edits be installed and verified in both directions?	Two-sided adaptation and closure checking	15

Table 1.3: What is established at the methodological and system-specific levels.

Level	Established here	External test still required
System-independent method	Pair-relation evaluation with overlapping gold memberships; recursive Pareto strata; exact linear-cost support intervals; reporting fields for adaptable policies.	Application across independent stemmer families, resources and tasks.
Egothor–Radixor evidence	Patch-command semantics; induction sensitivity; resource support structure; family-holdout transfer; runtime; policy transfer; two-sided adaptation; one English development-set retrieval case.	Matched ancestor/comparator campaigns, independent lexical references, unseen-query multilingual retrieval and appropriate inferential replication.

Chapter 2

Conflation Policy as the Treatment Variable

Stemming is often described operationally—remove an ending, look up a form, apply a learned rewrite—when the retrieval system ultimately observes a much simpler consequence: several surface forms may become indistinguishable after analysis. That equivalence structure is the level at which this monograph defines *policy*. The definition is deliberately independent of whether the implementation uses suffix rules, a dictionary, an automaton, a neural model, or a patch-command trie.

This move is more than terminology. If two systems implement different conflation relations, a comparison between their default outputs is jointly a comparison of mechanism and policy. If one mechanism can realise several relations, the policy becomes a treatment variable. The nine-study programme is organised around making that variable explicit, learnable, measurable and, finally, testable in retrieval.

2.1 The retrieval object is an equivalence decision

Let V denote the vocabulary presented to an analyser and let $h : V \rightarrow S$ map each surface form to the string used for indexing or query analysis. The literal strings in S are implementation outputs; the retrieval-relevant relation induced by them is

$$u \sim_h v \iff h(u) = h(v).$$

Two methods can emit different stem strings and nevertheless realise exactly the same relation on V . Conversely, two methods can emit superficially similar strings while disagreeing on which forms are grouped.

Definition 2.1 (Conflation function). A *conflation function* is a deterministic mapping $h : V \rightarrow S$. Its induced relation $\sim_h$ groups all forms receiving the same output. Unless stated otherwise, intrinsic comparisons in this volume concern the relation $\sim_h$, not the linguistic well-formedness of the output string itself.

That distinction is essential for a system such as Radixor. Its patch commands need not emit a dictionary lemma in every deployment; they must realise the desired grouping for the task. A canonical-looking string can be convenient for inspection, but the grouping is what changes postings and query matches.

2.2 Stemming, lemmatisation and morphological analysis are not synonyms

The literature routinely places several normalisation tasks next to one another because they manipulate word forms. They should nevertheless be separated by the evidence available at decision time and by the contract of the output.

A *stemmer* used in a search analyser normally sees a type in isolation and returns one indexing form. A *lemmatiser* aims at a lexical lemma and may use part-of-speech, morphosyntactic features or sentence context to resolve ambiguity. A *morphological analyser* may return several analyses rather than one conflation key. An unsupervised segmenter such as Morfessor solves yet another problem: discovering sub-word structure rather than selecting one search-equivalence class.

The relevant comparator set is defined by an information budget and an output contract, not by whether a tool is labelled “morphological”. A contextual lemmatiser can be an informative neighbouring method without being a fair same-task baseline for a context-free indexing stemmer.

This principle recurs in the validation agenda. Morfologik, Stempel, Snowball, KStem and other type-level analysers can be compared directly once resource and output semantics are aligned. Contextual lemma-rule systems require a separate matched-budget experiment rather than being silently treated as equivalent baselines.

2.3 A policy is a declared reference relation

The word *policy* is used here in a deliberately technical sense. Suppose a resource declares groups of forms that should be treated as related for an intended use. Those groups induce a reference relation R on the observed vocabulary. A deployed analyser induces its own relation P . The experimental question is then how P differs from R and, when the reference changes, whether the same learning mechanism can move P in the corresponding direction.

Definition 2.2 (Lexical policy). A *lexical conflation policy* over a finite vocabulary V is a symmetric, reflexive reference relation $R \subseteq V \times V$ whose positive pairs state which observed forms are intended to be conflated for the declared task. The relation may arise from disjoint classes or from overlapping lexical groups; the latter case need not be an equivalence relation.

The definition deliberately does not call R “linguistic truth”. UniMorph, CISTEM, a hand-curated domain dictionary and a retrieval-derived oracle can all encode different legitimate policies. What must be reported is which one was used, how it was constructed, and which conclusions depend on it.

2.4 The error surface has two axes

Given a reference relation R and a predicted relation P , every unordered pair of distinct forms falls into one of four cells. Pairs joined by R but not by P are missed conflations; pairs

separated by R but joined by P are false conflations. The former are the relation-level form of understemming and the latter the relation-level form of overstemming.

Definition 2.3 (Under- and over-conflation). For a fixed evaluation population, let FN denote reference-positive pairs not predicted positive and FP reference-negative pairs predicted positive. Let TP and TN have their usual pairwise meanings. We use

$$UI = \frac{FN}{TP + FN}, \quad OI = \frac{FP}{TN + FP},$$

with denominator-zero cases reported explicitly rather than silently imputed.

Neither axis can be reduced to the other. A method can reduce UI simply by joining more forms, but that may increase OI; another can protect lexical specificity and move in the opposite direction. The geometry is shown in Figure 2.1.

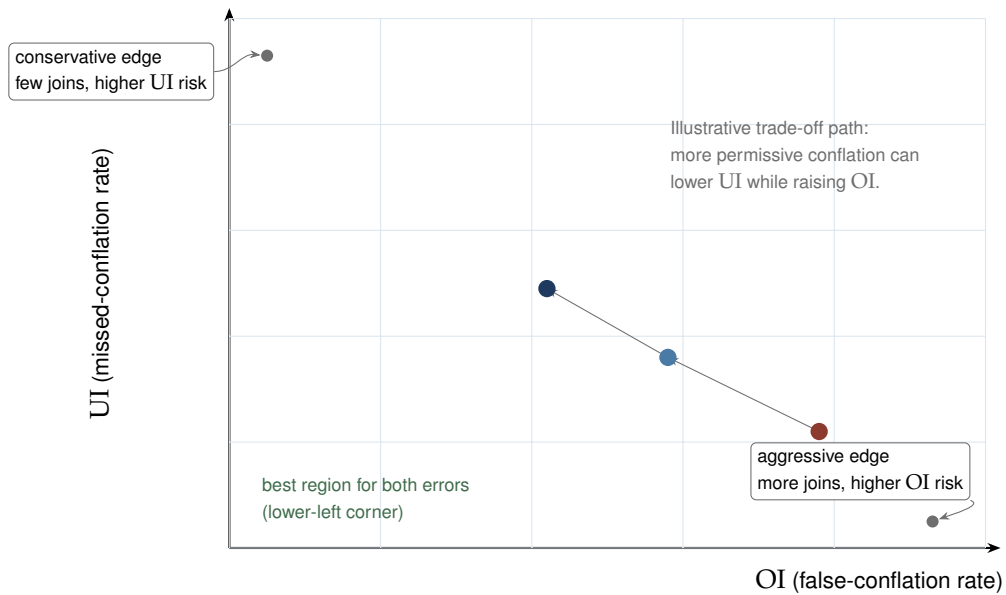


Figure 2.1: Intrinsic policy agreement is not naturally one-dimensional. Under-conflation and over-conflation are separate error rates; reducing one can increase the other. The corner callouts mark the two limiting regimes, while the three interior points show an illustrative trade-off path. Chapter 11 therefore compares candidates by dominance and support before introducing any scalar cost ratio.

This is why a single “stemming accuracy” cannot carry the whole argument. **P3** (Galambos, 2026c) treats the two error axes directly and introduces recursive Pareto strata. Scalars such as balanced accuracy remain useful for summaries, but only after the multi-objective structure is visible.

2.5 Why overlapping groups matter

P3 (Galambos, 2026c) sharpens an issue that is often hidden by toy examples: a lexical reference need not partition the vocabulary into mutually exclusive classes. The same observed surface form can occur in more than one lexical family, alternative canonical targets can coexist in the source, and a multi-output analyser may deliberately retain

several analyses before a later selection stage. In all of those cases, the scientifically stable object is not a single partition index but the set of positive and negative pairwise judgements that the study claims to evaluate.

This is why the programme adopts relation semantics rather than family IDs as its primary language for intrinsic comparison. A partition-based score can be perfectly meaningful when the source really is a partition, but it becomes hard to interpret when overlaps are present. The same numerical change can mix three different phenomena: genuine policy disagreement, ambiguity in the source resource, and a scorer’s forced collapse of overlapping evidence into one class assignment. By working directly with pairwise cover relations, P3 avoids building these assumptions into the metric itself.

The distinction matters especially for multilingual auditing and historical comparison. Some resources are conservative and nearly partition-like; others are noisier or intentionally retain competing analyses. Treating them all as if they provided the same type of gold structure would make the methodology look cleaner than the evidence really is. The present monograph therefore uses overlap-tolerant reference relations whenever the source warrants them and states explicitly when a simpler equivalence-class interpretation is safe.

The pairwise relation is not a stylistic reformulation of class membership. It is the minimal representation that continues to make sense when lexical groups overlap, when one surface belongs to several rows, or when a system exposes multiple candidate analyses before downstream selection.

2.6 Three registers of evidence

The research programme operates in three registers that must never be treated as interchangeable.

Table 2.1: The same analyser can be studied as a lexical relation, as a compiled software object, and as a component of a retrieval system. Conclusions move between registers only when an experiment provides the bridge.

Register	Question	Representative evidence
Lexical/policy	Which form pairs are joined or separated?	P1, P3, P4, P6
Compiled/system	Which transformation is selected, how is it represented, and what does it cost?	P2, P5, P7, P9
Retrieval	How does the realised policy change ranking under a declared scorer and objective?	P8 (with P9 as the deployment contract)

A lexical improvement is not automatically a retrieval improvement. A faster runtime is not evidence of a better relation. A retrieval gain on a development set does not prove that an intrinsic reference is universally correct. Much of the apparent disagreement in stemming research disappears once these registers are named explicitly.

2.7 Why policy, not algorithm name, is the unit of comparison

The conceptual claim of the monograph can now be stated without reference to Radixor. A stemmer name identifies an implementation family; it does not fully identify the relation realised on a corpus. Resource choice, construction parameters, adaptation data, output policy and ranker settings can all change the observable treatment.

For a fixed mechanism M , let π denote a lexical policy, θ the construction/deployment configuration, d a document/query population, and m an evaluation objective. The quantity observed in an experiment is better written as

$$Q(M, \pi, \theta, d, m)$$

than as $Q(M)$. The notation does not assert that all five arguments can be independently controlled in every experiment. It records what must be held fixed or reported before a comparison can be interpreted causally.

The remainder of the book makes those arguments concrete. Part II defines the compiled policy object and how it is learned. Part III asks how faithfully and efficiently it behaves. Part IV changes the policy while holding the engine fixed and then measures a retrieval consequence. The final part asks how much of that evidence transfers from one artefact to the field as a whole.

Chapter 3

Lineages, Parallels, and Verified Chronology

Historical diagrams are unusually easy to over-read. A horizontal position looks like a priority claim; an arrow looks like evidence of influence; two boxes placed in one column look as though they belong to one technical family. For a research programme whose mechanism has survived for more than two decades, those visual suggestions are not harmless. This chapter therefore uses a stricter convention than an ordinary related-work survey: *dates* come from identifiable publications or release records, and *arrows* are reserved for documented continuity. Similarity without evidence of descent is shown by adjacency, not by an arrow.

The result is less visually dramatic than a single genealogical tree, but more useful scientifically. It separates four questions that a conventional history often collapses: when a method was documented, which software line continued it, which later systems solved a related problem independently, and which mechanisms merely resemble one another when viewed at a sufficiently abstract level.

3.1 What the diagram is allowed to claim

Figure 3.1 is organised as a set of parallel lanes. The vertical direction is chronological within each lane, but the lanes are not claimed to form a common ancestor tree. A solid arrow means that the later item is explicitly documented as deriving from, incorporating, renaming, or continuing the earlier one. No dashed “similarity” arrows are used. This is intentional: a mechanistic comparison belongs in prose or in a table, where its scope can be stated, rather than in a visual grammar that readers naturally interpret genealogically.

3.2 The IR rule tradition

The conventional stemming chronology begins with hand-engineered reductions. Lovins’s one-pass suffix stripper established the practical pattern of reducing variants without performing full linguistic analysis (Lovins, 1968). Porter replaced the single list with a staged procedure whose conditions control when a suffix rule is permitted to fire (Porter, 1980). Paice and Husk contributed another iterative rule system, while Paice’s evaluation work became at least as durable as the algorithm itself because it made false splitting and false merging measurable (Paice, 1990, 1994). Krovetz then made lexical acceptability part of the decision by gating morphological operations with a dictionary (Krovetz, 1993).

These systems matter to the present programme chiefly as alternative ways of specifying a conflation policy. Their policy is encoded in program logic, rule tables and exception data. The Egothor/Radixor line makes a different engineering choice: it induces reusable edit programs from an external lexical relation and compiles the resulting decisions. The difference is not that one family is “linguistic” and the other is not; it is where the policy enters the system and how explicitly it can later be varied.

3.3 The documented Egothor–Stempel–Radixor line

The publication anchor for the line studied in this volume is Galambos’s SOFSEM paper of 2001 (Galambos, 2001). It describes a Java stemming/lemmatizing method that derives transformations from lexical evidence rather than requiring a manually authored language grammar. The associated Egothor code line contains the patch generator, patch executor, trie representation and reduction machinery that Chapters 5–9 formalise.

The later Apache Lucene Stempel module is not merely a similar implementation. Lucene’s own API documentation states that the core stemming algorithm and implementation are taken from the Egothor project (Apache Software Foundation, 2024). A conservative historical statement is therefore possible without reconstructing every intermediate repository revision: the Egothor-derived Stempel module is publicly documented in Lucene 3.1 (2011) (Apache Software Foundation, 2011), and the same line remains present in contemporary Lucene documentation. That continuity is important evidence of engineering viability. It is not, by itself, evidence that the model is superior on a particular lexical or retrieval benchmark.

RADIXOR is the modern continuation examined by the nine constituent studies. Its contribution is not the invention of the patch instruction language. That instruction language is inherited. What changes is the experimental and systems envelope around it: explicit formal semantics, large multilingual resource studies, policy-conditioned training, two-sided adaptation, verification of compiled memberships, Python packaging, and downstream retrieval experiments (Galambos, 2026a,b,d,g,h,i). Chapter 4 separates this continuity from the genuinely new work.

3.4 Stempelator and Morfologik: a neighbouring line, not a predecessor

The available publication and release record places this line after the 2001 Egothor publication. Weiss’s *Stempelator* technical report is dated 2005 and describes a hybrid Polish stemmer combining Lametyzator with Stempel (Weiss, 2005). The public Morfologik project record and project blog date the project to 2006 and document early dictionary and library releases (Miłkowski and Weiss, 2006). In May 2007 the project explicitly announced that the Stempelator code had been reorganised and renamed `morfologik-stemming` (Miłkowski and Weiss, 2007). On the evidence used here, there is therefore no basis for placing Morfologik before the 2001 Egothor publication.

Morfologik remains highly relevant as a comparator. Its compact automata and lexical-resource orientation probe a different design point from a compiled patch-command trie. The proper comparison is empirical—quality, coverage, runtime and memory under matched resources—rather than genealogical. This is why Morfologik appears in the validation programme of Chapter 18 without an ancestry arrow in Figure 3.1.

3.5 Parallel learned-transformation traditions

The idea of learning transformations from examples predates Egothor and occurs in several neighbouring literatures. Transformation-based learning (Brill, 1995), induced unknown-word rules (Mikheev, 1997), and memory-based morphological analysis (van den Bosch and Daelemans, 1999) established that linguistic normalisation need not be expressed only as hand-written suffix logic. Later token-level lemmatisers made edit or rewrite behaviour an explicit prediction target, including Morfette (Chrupała et al., 2008), CSTlemma (Jongejan and Dalianis, 2009), LemmaGen (Juršič et al., 2010), Lemming (Müller et al., 2015), and related neural systems (Bergmanis and Goldwater, 2018).

Those parallels are technically informative but not interchangeable. A contextual lemmatiser may consume part-of-speech or sentence evidence that a type-level search stemmer never sees. Conversely, a type-level compiled stemmer may operate at indexing throughput and with a fixed deployment model that is not the target of a contextual tagger. Mechanistic resemblance therefore does not create an equal-budget benchmark automatically.

XSTEM is closer to the search-stemming problem. Its first arXiv submission is from 2022 (Baker, 2022). As analysed in P9 (Galambos, 2026i), its terminal cut-and-append action can be embedded in the older Egothor patch executor, whereas an Egothor patch can preserve source characters between two edits and therefore expresses native actions outside a single terminal cut-and-append. The useful conclusion is one of technical scope and chronology; it does not imply that every function obtainable by one complete system is impossible to reproduce by another system through composition or additional machinery.

3.6 Chronology is not a novelty verdict

The chronology supports several statements and deliberately stops short of others. The patch-command stemming mechanism is documented in 2001 (Galambos, 2001). Stempelator follows in 2005 (Weiss, 2005); Morfologik’s public project record follows in 2006 (Miłkowski and Weiss, 2006); CSTlemma and LemmaGen are documented in 2009 and 2010 (Jongejan and Dalianis, 2009; Juršič et al., 2010); the Egothor-derived Stempel module is verifiably present in Lucene 3.1 in 2011 (Apache Software Foundation, 2011); and XSTEM is first submitted in 2022 (Baker, 2022).

Interpretation boundary. A verified publication date does not establish knowledge, influence, or independent reinvention. Likewise, native-action inclusion does not establish end-to-end superiority. This monograph uses chronology to prevent impossible or misleading ancestry diagrams and uses experiment to decide performance questions. The two evidential tasks are intentionally kept separate.

3.7 A mechanism matrix instead of a single family tree

For later experimental chapters, the distinctions in Table 3.1 are more useful than ancestry alone. The table asks what evidence a method consumes, what object is executed at runtime,

whether an unseen type can receive a non-trivial transformation, and whether the conflation policy is externally replaceable without rewriting the engine.

Table 3.1: A comparison by experimental affordance rather than by historical family label. “OOV change” means that a form absent from the training lexicon can receive a non-identity transformation; it does not imply that the transformation is correct.

Line	Evidence	Runtime object	OOV change	Policy control
Rule-based stemmers	authored rules	ordered rewrite program	yes	edit the rules
Krovetz-style gating	rules + lexicon	lookup-constrained rules	limited or fallback	partial
Morfologik line	lexical dictionary + hybrid logic	FSA/dictionary analyser	varies by implementation	replace resources
Egothor/Radixor	lexical relation	context-selected patch command	yes	rebuild policy data
Contextual lemma-rule systems	annotated tokens + features	predicted edit or rewrite rule	yes	retrain

The matrix makes the historical role of this chapter precise. It does not select a winner. It identifies which comparisons are genealogical, which are mechanistic, and which require a controlled empirical study. That distinction becomes essential once the monograph moves from history to the formal object that all subsequent experiments manipulate.

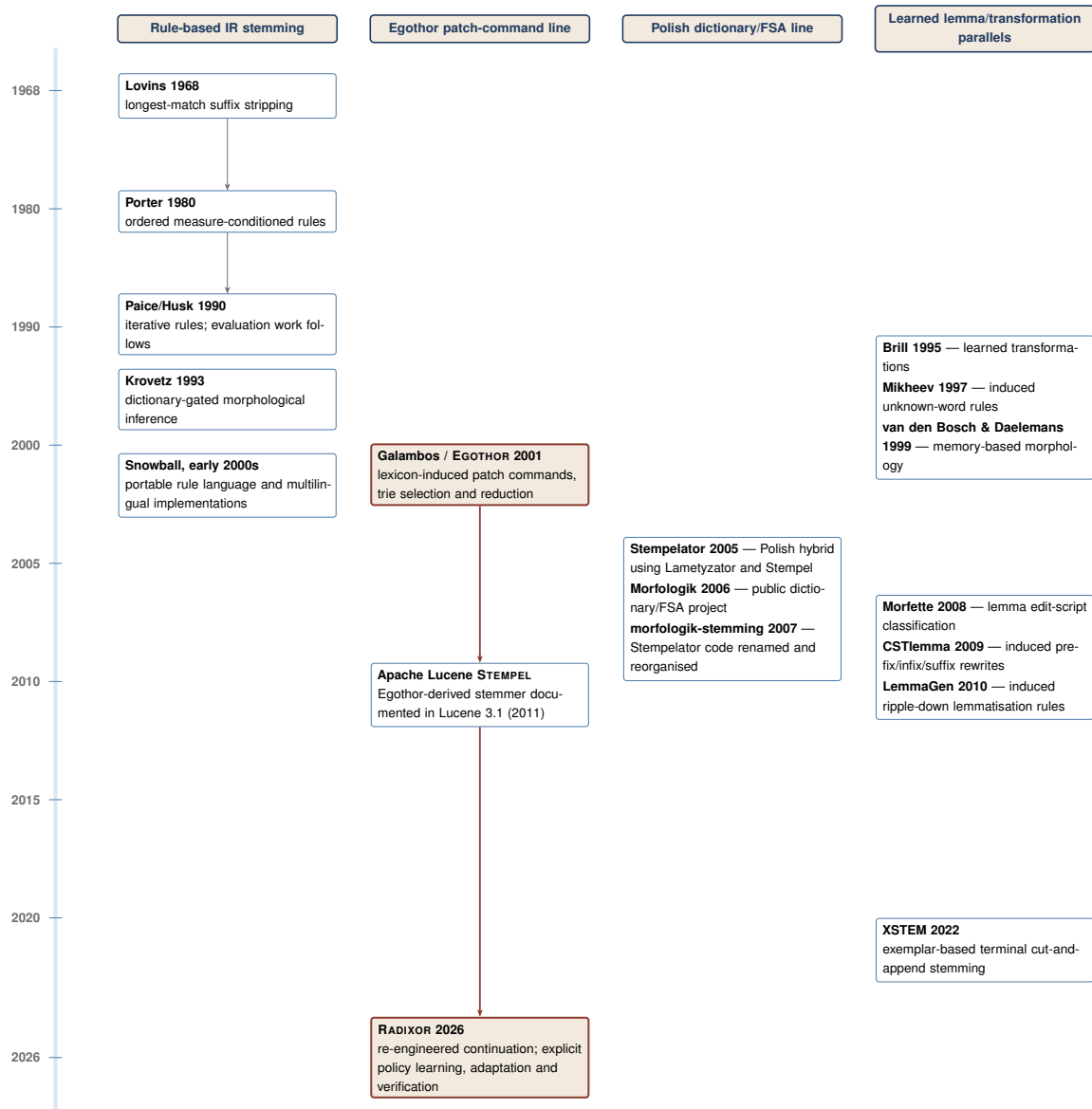


Figure 3.1: A verified chronology organised as parallel technical lanes. Solid red arrows are used only where the later artefact is documented as a continuation, derivative, or rename of the earlier one. Vertical placement records chronological order; dense publication windows are shown as dated clusters so that labels remain legible. Horizontal proximity does not assert influence. In particular, the Morfologik/Stempelator lane is shown *after* the 2001 Egothor publication and is not drawn as an ancestor of Egothor or Radixor.

Chapter 4

From Egothor to Radixor: What Was Inherited and What Changed

A research programme built on an older algorithm has two obligations that are easy to confuse. It must credit the inherited mechanism precisely, and it must state equally precisely what the modern work adds. Treating the entire current system as new would erase the 2001 contribution; treating the 2026 programme as mere maintenance would erase the formal, empirical and deployment work carried out since. This chapter establishes the boundary.

The inherited idea is compact: learn a vocabulary of edit programs from a lexical relation, index those programs by form context, and execute the selected program on the runtime input. The modern programme keeps that computational premise and changes almost everything around the evidential contract: the compiler is formalised, the parameter space is measured, the training resource is audited, generalisation is isolated by family holdout, error trade-offs are made explicit, policies can be changed after the standard model exists, and the compiled result can be verified against a declared vocabulary before retrieval is evaluated.

4.1 The 2001 design decision

The SOFSEM paper (Galambos, 2001) was motivated by a practical search problem. A manually written suffix stemmer embeds linguistic decisions in code; a pure dictionary embeds answers in a finite list. Egothor instead consumed lexical examples during model construction and produced a runtime object that selected a transformation. The lexical resource was therefore evidence used to *induce behaviour*, not a database that had to be present for every query token.

The conceptual inheritance of Radixor is not “a large dictionary stemmer”. It is an offline-to-online separation: lexical relations are used to learn patch commands and the runtime model carries the learned transformation policy. This is the premise tested directly by **P1** (Galambos, 2026a) and formalised by **P7** (Galambos, 2026g).

That distinction explains why out-of-vocabulary behaviour is meaningful in this lineage. If a compiled state chooses a command for an unseen word, the command operates on the unseen word’s own characters. A successful output is therefore not an exact dictionary hit

disguised as generalisation. Conversely, the mere fact that an output exists says nothing about linguistic correctness; that is why P1 uses withheld lexical families rather than runtime non-failure as its criterion.

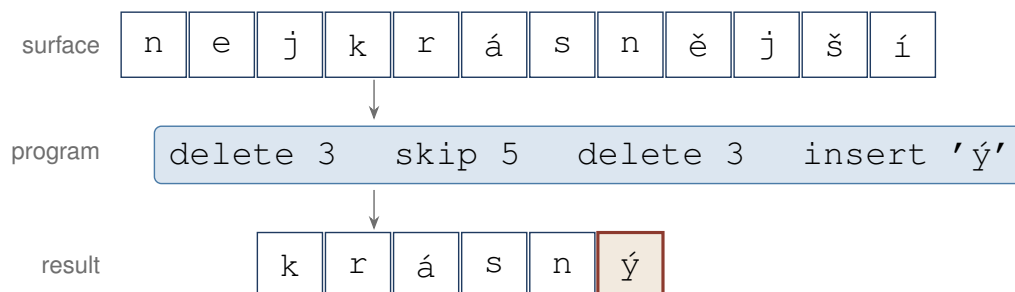
4.2 Patch commands as the inherited instruction language

Definition 4.1 (Patch command, informal). A *patch command* is a finite program of cursor-relative edit operations. When executed on a surface form, the program may copy a region unchanged, delete one or more input characters, insert new characters, or replace input characters. The formal execution function is defined in Chapter 5.

The primitive operations used throughout the monograph are summarised in Table 4.1. Their importance is not the syntax of a historical implementation; it is the ability to represent a transformation with separated edit sites and an unchanged span between them.

Table 4.1: Conceptual operations of the patch executor. Concrete serialisations are implementation details; the semantic distinction is between copying, deleting, inserting and replacing around a moving cursor.

Operation	Argument	Semantic effect
skip	count n	copy the next n input characters and advance
delete	count n	consume the next n input characters without copying them
insert	character/string x	append new material without consuming input
replace	character x	emit x while consuming the current input position



The transformation edits two separated regions while preserving the material between them.

Figure 4.1: An illustrative patch that maps Czech *nejkrásnější* to *krásný*. The example is valuable because it exposes the native action language: an edit can occur before and after a copied interior span.

4.2.1 A bounded expressivity comparison

P9 (Galambos, 2026i) revisits this instruction language when comparing Radixor with later configurable systems. The strongest safe conclusion is about *native actions*, not arbitrary system composition.

Proposition 4.2 (Native-action inclusion). A *terminal cut-and-append transformation* can be represented by an Egothor patch: copy the retained prefix, delete the terminal remainder, and insert

the replacement. A patch that edits two separated sites while preserving an interior span has no equivalent as one terminal cut-and-append action.

The proposition does not say that a complete system restricted to a simpler primitive could never emulate a richer system by repeated application, additional state, or a different wrapper. It says that the historical Egothor executor already contains the simpler terminal action and also admits native programs outside that action class. This narrower statement is both more useful and easier to verify.

4.3 The first study already treated configuration as a variable

The 2001 work did not present a single immutable dictionary as the algorithm. It varied construction choices and examined how the resulting models changed (Galambos, 2001). That point connects the historical work directly to the modern research programme: learned transformation systems have a construction space, and a reported model is one operating point in that space.

P2 (Galambos, 2026b) turns that observation into a large controlled experiment. Across 20 UniMorph-derived resources, 234 normalised edit-cost settings per resource produce 4,680 resource–cost configurations but collapse exactly to 334 induced command behaviours. The result is not “costs do not matter”. It is more specific: costs are structural hyperparameters, yet much of the sampled numeric space is behaviourally redundant. This is the modern answer to a question already visible in the original design.

The same logic recurs elsewhere. **P6** (Galambos, 2026f) shows that source resource consistency changes the support topology from which commands are learned. **P4** (Galambos, 2026d) changes the target conflation policy while holding the learning mechanism fixed. **P8** (Galambos, 2026h) then shows that once the policy is changed, the ranking environment with which it interacts can also change. Configuration is therefore not an implementation afterthought; it is part of the experimental treatment.

4.4 Stempel as continuity evidence

Apache Lucene’s Stempel documentation explicitly attributes the core stemming algorithm and implementation to Egothor (Apache Software Foundation, 2024). Historical Lucene 3.1 documentation provides a conservative availability witness in 2011 (Apache Software Foundation, 2011). This matters in two different ways.

First, it documents software continuity. The algorithm did not disappear with the original research prototype; an Egothor-derived implementation became part of a production IR ecosystem and remained maintained across later Lucene releases. That is unusually strong evidence that the mechanism is technically usable over long periods.

Second, Stempel is the most informative missing *ancestor control* for the modern programme. A same-resource comparison between Stempel and Radixor can ask what the new compiler, model representation and surrounding adaptation machinery add while holding the broad algorithmic lineage unusually close. The validation agenda in Chapter 18 therefore treats this comparison as a priority rather than as historical decoration.

Interpretation boundary. Production adoption is evidence of continuity and engineering relevance. It is not evidence that Stempel or Radixor has higher stemming accuracy, lower latency, or better retrieval effectiveness than an alternative. Those claims require the matched experiments defined elsewhere in the monograph.

4.5 What Radixor changes

The relationship between the inherited mechanism and the nine-paper programme is summarised in Table 4.2. The table intentionally avoids the language of a wholesale replacement. Several core operations are inherited; what changes is the precision with which their semantics, evidence and deployment are specified.

Table 4.2: The modern programme separates inherited mechanism from new formal, empirical and operational contributions.

Dimension	Inherited line	2026 programme
Transformation object	patch commands induced from lexical examples	retained and specified as an executable semantic object (P7)
Context selection	trie/suffix-context lookup	retained; compiler observables and reduction contracts formalised (P7)
Construction parameters	configurable induction	exhaustive sampled cost study with exact behaviour collapse (P2)
Training evidence	lexical resource	audited as a causal input to learned command support (P6)
Beyond-lexicon behaviour	intended property of executable rules	measured with family-level holdout and leakage control (P1)
Quality evaluation	agreement/error measurement	overlapping-reference pair semantics, Pareto strata and support analysis (P3)
Policy choice	implicit in resource and configuration	experimentally varied under fixed mechanism (P4)
Deployment adaptation	rebuild/edit-oriented	positive assignments + negative boundaries + closure verification (P9)
Downstream effect	IR motivation	full-corpus MIRACL development experiment and BM25 interaction analysis (P8)
Runtime	Java implementation	multilingual microbenchmark campaign tied to compiled structure (P5)

The final three rows are where the research programme moves beyond a classical “train a stemmer and benchmark it” story. The policy can be changed without changing the transformation engine, the changed policy can be verified after compilation, and its retrieval consequences can be measured as part of a coupled search configuration. Those steps are what allow the rest of the book to discuss a *conflation policy* as an experimental

variable rather than a metaphor for algorithm choice.

4.6 A historical line becomes an experimental platform

The practical interpretation of the lineage is therefore neither nostalgic nor competitive. Egothor supplies a transformation language and a compiled lookup idea that survived in Stempel. Radixor turns that line into an instrument for asking a different class of questions: which lexical relation should be realised, how faithfully is it realised after compilation, how does it transfer beyond enumerated forms, what error frontier does it occupy, and what happens when the realised relation enters an actual ranker?

That transition—from an algorithm to an experimental platform—is the reason the historical material belongs before the formal chapters. The next part can now define the model without blurring inherited operations with the new claims made about them.

Part II

Learning and Compiling the Policy

Chapter 5

The Executable Policy Object

The rest of the monograph reasons about a compiled analyser, not about an abstract “stemmer”. This chapter therefore defines the smallest set of objects needed to connect lexical evidence to runtime behaviour: a lexical relation, a vocabulary of patch commands, a context-indexing trie, a command selection function, and the realised string transformation. The definitions are intentionally implementation-oriented enough to map onto the Egothor/Radixor code line, but abstract enough to separate semantic claims from storage format.

The key separation is between *selection* and *execution*. A model selects a command from form context; the command then executes on the current input. Compiler proofs in P7 (Galambos, 2026g) concern the first function. Generalisation and retrieval observe the second function composed with it.

5.1 Lexical evidence

Definition 5.1 (Lexical resource and relation). A finite lexical resource consists of canonical targets $r \in \Sigma^*$ with finite sets of associated surface forms $V(r) \subseteq \Sigma^*$, where the resource may include r itself among its variants. It induces

$$\mathcal{L} = \{(w, r) : w \in V(r)\}.$$

The symbol r denotes the resource’s designated target. It may be a lemma, root-like form, or another canonical representative; the formal machinery does not require one linguistic interpretation.

The resource can contain ambiguity. The same surface string may occur in more than one lexical family, which is why later evaluation is formulated as a relation over form pairs rather than assuming that the gold data always form a partition. For training/test separation, however, the family row is treated as an atomic unit. This prevents a model from being trained on one member of a paradigm and credited for “generalising” to another member of the same held-out row.

5.2 Patch commands are programs, not stored answers

Definition 5.2 (Patch command). Let

$$\Pi = \{\text{skip}(n), \text{delete}(n), \text{insert}(x), \text{replace}(x) : n \in \mathbb{N}, x \in \Sigma\}.$$

A patch command is a finite sequence $c = \langle p_1, \dots, p_k \rangle$ with $p_i \in \Pi$. The set of commands induced for a model is denoted $\mathcal{C}$.

Definition 5.3 (Execution). $\text{Exec}(c, w)$ executes command c against the characters of input w . Cursor movement is bounded at the end of the string, so execution is total over $\mathcal{C} \times \Sigma^*$.

Algorithm 5.1: $\text{Exec}(c, w)$ – execute a patch command

Input: command $c = \langle p_1, \dots, p_k \rangle$; input $w = w_1 \dots w_m$
Output: transformed string y

```

1  $y \leftarrow \epsilon; i \leftarrow 1;$ 
2 for  $j \leftarrow 1$  to  $k$  do
3   switch  $p_j$  do
4     case  $\text{skip}(n)$  do
5        $t \leftarrow \min(n, m - i + 1)$ ; append  $w_i \dots w_{i+t-1}$  to  $y$ ;  $i \leftarrow i + t$ ;
6     case  $\text{delete}(n)$  do
7        $i \leftarrow \min(i + n, m + 1)$ ;
8     case  $\text{insert}(x)$  do
9       append  $x$  to  $y$ ;
10    case  $\text{replace}(x)$  do
11      append  $x$  to  $y$ ; if  $i \leq m$  then  $i \leftarrow i + 1$ ;
12 if  $i \leq m$  then append  $w_i \dots w_m$  to  $y$ ;
13 return  $y$ ;
```

Example 5.4 (A transformation with preserved interior). For $w = \text{nejkrásnější}$, the illustrative command

$$\langle \text{delete}(3), \text{skip}(5), \text{delete}(3), \text{insert}(\acute{y}) \rangle$$

removes a prefix, copies an interior region, removes later material and appends a replacement ending, producing *krásný*. The copied interior is the native-action distinction used in Proposition 4.2.

A command is reusable because it contains an operation, not a memorised target string. When a command selected from a familiar suffix context is applied to an unseen word, preserved characters come from that unseen input itself. This is the representational basis for the family-transfer experiment in Chapter 12.

5.3 The command-selection structure

Definition 5.5 (Patch-command trie). A patch-command trie $\mathcal{T}$ is a rooted character-indexed structure whose node n may carry a value $\text{val}(n) \in \mathcal{C} \cup \{\perp\}$. For suffix-oriented models, the lookup key is the reversed surface string $\overleftarrow{w}$, so a path from the root represents progressively longer suffix context in the original word.

Reversal turns morphological suffix similarity into shared prefix structure in the trie. This is an indexing choice rather than a linguistic theorem; a prefix-oriented model could

use the forward form. What matters for the compiler is that multiple contexts share substructure that can later be canonicalised or contracted.

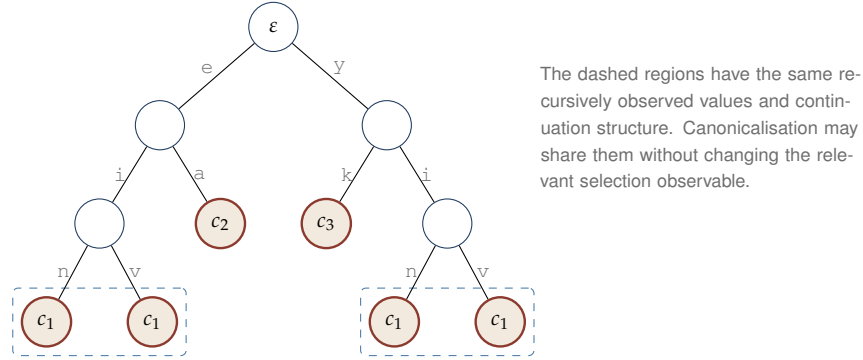


Figure 5.1: A suffix-context trie. Valued nodes carry commands; routing nodes need not. Recursively equivalent regions are candidates for semantic canonicalisation in Chapter 9.

5.4 Selection and realised output are different functions

Definition 5.6 (Command selection). Let $P(\overleftarrow{w})$ be the nodes visited while traversing the reversed input. The selection function returns the deepest visited value-bearing node,

$$C(w) = \text{val} \left(\arg \max_{n \in P(\overleftarrow{w})} \{ \text{depth}(n) : \text{val}(n) \neq \perp \} \right),$$

with $C(w) = \perp$ if no visited node carries a value.

Definition 5.7 (Realised transformation). The deployed analyser computes

$$H(w) = \begin{cases} \text{Exec}(C(w), w), & C(w) \neq \perp, \\ w, & C(w) = \perp. \end{cases}$$

The distinction is the formal hinge of the compiler analysis. A reduction may preserve which command is selected while the resulting stem still depends on the full runtime input through Exec. Conversely, extending a command selection to a new continuation can change the domain on which the decision is made without storing any new target strings.

Three runtime properties follow immediately. Traversal consumes at most the word length; accepting leaves can terminate the search early; and a word that matches no command context falls back to identity rather than being forced into an unrelated class. P5 measures the cost of the actual implementation instead of treating these bounds as a performance prediction.

5.5 Reduction mode is part of the model condition

The compiler can expose different observables—for example a preferred command, a ranked candidate sequence, or an unordered candidate set. A reduction mode specifies which observable must agree before two regions are shareable. More informative observables constrain sharing more strongly.

Algorithm 5.2: $H(w)$ – runtime lookup and command execution**Input:** surface form w ; compiled trie $\mathcal{T}$ with root n_0 **Output:** normalised output

```

1  $n \leftarrow n_0$ ;  $best \leftarrow \text{val}(n_0)$ ;
2 for  $i \leftarrow |w|$  to 1 do
3   if  $n$  has no edge  $w_i$  then break;
4    $n \leftarrow$  child reached by  $w_i$ ;
5   if  $\text{val}(n) \neq \perp$  then  $best \leftarrow \text{val}(n)$ ;
6   if  $n$  is an accepting leaf then break;
7 if  $best = \perp$  then return  $w$ ;
8 return Exec( $best, w$ );

```

This is why the mode belongs to the reproducible policy condition. Two builds from the same lexical resource and the same cost vector can legitimately differ if they compile under different observables. Theorems in Chapter 9 are therefore parameterised by the semantic contract rather than asserting a single unqualified notion of “equivalent subtree”.

5.6 Object ledger

Table 5.1: The formal objects and the studies that interrogate them.

Object	Symbol	Primary role
Lexical relation	$\mathcal{L}$	policy evidence; family holdout; overlapping reference construction
Patch command	$c \in \mathcal{C}$	learned reusable transformation; vocabulary-size studies
Compiled trie/DAG	$\mathcal{T}$	context selection, compiler reduction and runtime structure
Selection function	C	semantic observable preserved or extended by compilation
Realised transformation	H	actual output observed by intrinsic and retrieval experiments
Policy condition	π	resource + construction + adaptation information defining intended relation

The next chapter explains how $\mathcal{L}$ becomes $\mathcal{C}$ and $\mathcal{T}$. The chapter after that shows that apparently small upstream choices—edit costs and resource support—can materially change the object before the compiler ever attempts to reduce it.

From Lexical Pairs to Reusable Transformations

The learned model is produced by a deterministic compilation pipeline. A surface/target pair is aligned under an edit-cost vector, the chosen edit path is normalised into a patch command, and the command is indexed by the surface form's context. Only after all training evidence has been inserted does the compiler reduce the structure.

This ordering matters scientifically. P2 changes the edit-cost vector and observes different induced command vocabularies; P6 changes resource support and observes different command vocabularies; P7 begins only after that vocabulary and trie have already been built. Induction and reduction are therefore separate causes of the final artefact.

6.1 Construction pipeline

Algorithm 6.1: BUILD($\mathcal{L}, \gamma, \mu$) – induce and compile a model

Input: lexical relation $\mathcal{L}$; edit-cost vector γ ; reduction mode μ

Output: compiled model $(\mathcal{T}, \mathcal{C})$

```

1  $\mathcal{C} \leftarrow \emptyset$ ;  $\mathcal{T} \leftarrow$  empty trie;
2 foreach  $(w, r) \in \mathcal{L}$  in the frozen deterministic order do
3    $s \leftarrow \text{ALIGN}(w, r, \gamma)$ ;
4    $c \leftarrow \text{NORMALISE}(s)$ ;
5    $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$ ;
6    $\text{INSERT}(\mathcal{T}, \overleftarrow{w}, c)$ ;
7  $\mathcal{T} \leftarrow \text{CANONICALISE}(\mathcal{T}, \mu)$ ;
8  $\mathcal{T} \leftarrow \text{CONTRACT}(\mathcal{T})$ ;
9 return  $(\mathcal{T}, \mathcal{C})$ ;
```

The relation $\mathcal{L}$ supplies target evidence; γ determines which edit program is chosen when several programs can realise the same pair; μ determines what the compiler is allowed to identify. These are distinct configuration dimensions and are reported separately throughout the empirical chapters.

6.2 Minimum-cost alignment

For $w = w_1 \dots w_m$ and $r = r_1 \dots r_q$, dynamic programming computes a minimum-cost path through the standard edit lattice. Match, replacement, deletion and insertion are all candidates where defined. An explicit match or skip cost is retained because the experiment in P2 varies the relative price of preserving material as well as changing it.

Algorithm 6.2: $\text{ALIGN}(w, r, \gamma)$ – minimum-cost edit sequence

Input: w of length m ; r of length q ; non-negative costs $\gamma = (\gamma_{ins}, \gamma_{del}, \gamma_{rep}, \gamma_{skip})$

Output: one deterministically selected minimum-cost edit sequence

```

1 for  $i \leftarrow 0$  to  $m$  do
2    $D[i][0] \leftarrow i\gamma_{del}$ 
3 for  $j \leftarrow 0$  to  $q$  do
4    $D[0][j] \leftarrow j\gamma_{ins}$ 
5 for  $i \leftarrow 1$  to  $m$  do
6   for  $j \leftarrow 1$  to  $q$  do
7      $d_{diag} \leftarrow D[i-1][j-1] + \begin{cases} \gamma_{skip}, & w_i = r_j; \\ \gamma_{rep}, & w_i \neq r_j; \end{cases}$ 
8      $d_{del} \leftarrow D[i-1][j] + \gamma_{del};$ 
9      $d_{ins} \leftarrow D[i][j-1] + \gamma_{ins};$ 
10     $D[i][j] \leftarrow \min(d_{diag}, d_{del}, d_{ins});$ 
11 trace back from  $(m, q)$  using the frozen deterministic tie order;
12 return the corresponding edit sequence;
```

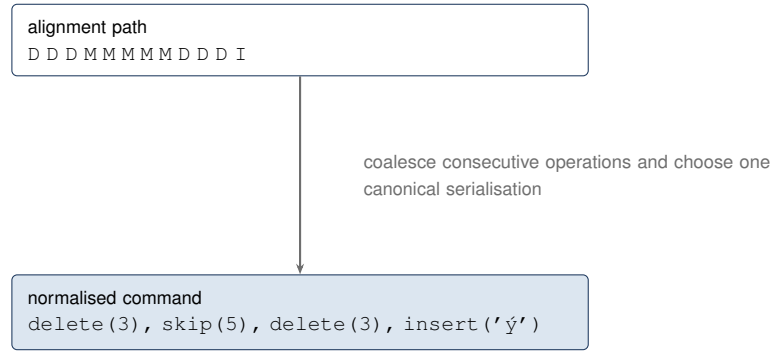
Two details are part of the model definition rather than implementation minutiae. First, deletion and insertion remain alternatives even when $w_i = r_j$; otherwise the procedure would not in general compute a minimum under arbitrary non-negative costs. Second, traceback ties are resolved by a fixed order. Equal total cost does not imply an equal command, so nondeterministic tie-breaking would make model identities irreproducible.

6.3 Normalising an edit path into a command

The raw traceback contains one operation per local step. Normalisation coalesces consecutive compatible operations and chooses a canonical encoding for commuting local choices. The purpose is semantic identity: when two training pairs induce the same reusable transformation, the command vocabulary should contain one symbolic command rather than several serialisations of the same action.

P2 defines exact behavioural equivalence at precisely this level: two cost settings are equivalent on a resource when the complete ordered sequence of normalised commands agrees. The definition is intentionally upstream of quality metrics.

The two verbs in Figure 6.1 do different theoretical work. *Coalescing* replaces a run of locally separate edits by one operation on a contiguous segment; *canonicalising* chooses one serialisation when several serialisations would execute to the same observed transformation. Without the first step, the vocabulary would contain accidental fragmentation caused only by local traceback granularity. Without the second, the same structural command could appear under several surface encodings and later be miscounted as several distinct learned



Different minimum-cost paths may encode different structural hypotheses even when they transform the training word to the same target; this is why edit costs become structural hyperparameters.

Figure 6.1: Illustrative conversion from a character-level edit path to the command representation executed at runtime.

hypotheses.

This is also the point at which a stemmer ceases to be merely an alignment procedure. Dynamic programming supplies an optimal path under a chosen local cost model, but induction must still decide what symbolic object will be stored and reused. P7 shows that this symbolic layer is where strong compiler properties become available: accepting-leaf generalisation, deterministic value selection and semantic-preservation claims all depend on the command language having a stable identity relation.

P2 provides the complementary warning. If edit costs change the preferred traceback, they can change the normalised command even when the immediate training pair still transforms correctly. Hyperparameters therefore act on the support topology seen by the compiler, not just on an auxiliary training loss.

6.4 Indexing commands by context

Commands are inserted under the reversed surface form. A node may receive support for more than one command; the active model mode determines how such support is represented and which observable the later reduction must preserve. The conceptual insertion procedure is shown in Algorithm 6.3.

Algorithm 6.3: $\text{INSERT}(\mathcal{T}, \overleftarrow{w}, c)$ – add command evidence

Input: trie $\mathcal{T}$; reversed key $\overleftarrow{w}$; command c

- 1 $n \leftarrow \text{root of } \mathcal{T}$;
 - 2 **foreach** character x in $\overleftarrow{w}$ **do**
 - 3 **if** n has no outgoing edge x **then**
 - 4 create child node n' and edge x ;
 - 5 $n \leftarrow$ child reached by x ;
 - 6 update the mode-specific support/value information of n with command c ;
-

Insertion is evidence accumulation, not yet compiler optimisation. P6 is therefore able to observe resource-consistency effects at the command-support level before the reduction of P7 is applied.

6.5 Determinism and row order

For a frozen resource, cost vector, normalisation rule and conflict semantics, the final unreduced evidence state should not depend on an arbitrary ordering of lexical rows. Where support aggregation is commutative and associative, row order can affect intermediate memory state but not the represented evidence.

Proposition 6.1 (Row-order independence under aggregated support). *Assume that command evidence stored at a node is combined by an associative and commutative aggregation operator and that command serialisation is canonical. Then any permutation of the same multiset of lexical training pairs yields the same node-level aggregate evidence before reduction.*

The proposition is intentionally conditional on the aggregation semantics. An implementation that resolves conflicts by “first wins” or “last wins” would not satisfy it. The reproducibility appendix records the deterministic input ordering anyway, because build identity should not rely on an unstated algebraic assumption.

6.6 What induction guarantees—and what it cannot

For each training pair, alignment and command formation produce a program that can transform the surface to the target under the stated execution semantics. That fact does not guarantee that the command will later be selected uniquely for every training surface after conflicts and reduction, nor that the same command is linguistically correct on a new surface sharing its context.

Interpretation boundary. Induction provides candidate transformation evidence. Compiler reduction provides a selected runtime representation. Generalisation quality is an empirical property of the composition. These three layers are measured by different studies and should not be collapsed into the statement “the algorithm learns the dictionary”.

The next two chapters examine the two upstream sources of structural variation before formal reduction: edit-cost choice (P2) and lexical-resource consistency (P6). Chapter 9 then asks what the compiler is entitled to merge or generalise once that induced evidence has been fixed.

When Alignment Parameters Become Model Structure

The compiler does not learn patch commands from surface/target pairs directly. It first chooses an edit alignment, and that choice depends on the edit-cost vector. In a conventional dynamic-programming exercise, the costs can look like local numerical preferences. In an induced stemmer they have a deeper role: changing the minimum-cost alignment can change the patch command, which changes the command vocabulary, the trie contents, the opportunities for sharing, and eventually the compiled runtime object.

This chapter is the monograph’s synthesis of **P2** (Galambos, 2026b). Its purpose is not to identify one universal cost vector. It asks a more basic question: how much of the nominal parameter space corresponds to genuinely different learned behaviour, and how much is only a different numerical name for the same induced model?

7.1 The experimental object

Let γ denote the tuple of insertion, deletion, replacement and match costs used by Algorithm 6.2. P2 samples delete/insert/replace costs from $\{1, 2, 3, 5, 10\}$ and match costs from $\{0, 1\}$. Removing settings that are equivalent under common positive scaling leaves 234 normalised configurations. The experiment applies those configurations to 20 UniMorph-derived lexical resources and evaluates the resulting models under the same family-level protocol used by the generalisation study.

The campaign scale is summarised in Table 7.1. The key quantity is not the number of numeric settings but the number of exact induced behaviours.

Table 7.1: The sampled grid and the exact behavioural collapse reported in P2. Logical observations count the nominal resource–cost–evaluation cells; physical scenarios exploit proven exact equivalence and are therefore not an approximation.

Quantity	Count	Interpretation
Normalised cost settings	234	distinct sampled relative-cost points
Resource–cost combinations	4,680	20 resources $\times$ 234 settings
Logical observations	234,000	nominal repeated evaluation cells
Physical evaluated scenarios	16,700	scenarios after exact reuse
Exact induced behaviours	334	resource-specific command-sequence classes

7.2 Exact behavioural equivalence

Two cost settings are declared equivalent on a resource only when they induce the same ordered command sequence for the complete mapping sequence. This is stronger than “similar accuracy” and stronger than “same output on the training forms”. It is an identity statement about the learned symbolic behaviour before later compilation effects are considered.

Proposition 7.1 (Exact command-sequence equivalence). *For a fixed finite lexical resource, equivalence of two sampled cost settings is decidable by finite comparison of the normalised induced command sequence. No numerical tolerance or statistical test is involved.*

Because the equivalence is established on the complete resource, it survives restriction to any subset.

Proposition 7.2 (Subset invariance). *If two cost settings induce identical commands for every row of a resource $\mathcal{L}$, then they induce identical commands for every subset $\mathcal{L}' \subseteq \mathcal{L}$.*

This property is operationally important. Once a resource-specific equivalence class has been identified, every nested coverage subset used by the family-holdout protocol can reuse that identity exactly. The campaign can therefore collapse repeated builds without changing the estimand.

Across the 4,680 resource–cost combinations, P2 observes only 334 exact resource-specific induced behaviours. The resulting 234,000 logical observations are realised by 16,700 physical scenarios. The reduction is a consequence of symbolic identity, not a sampling shortcut.

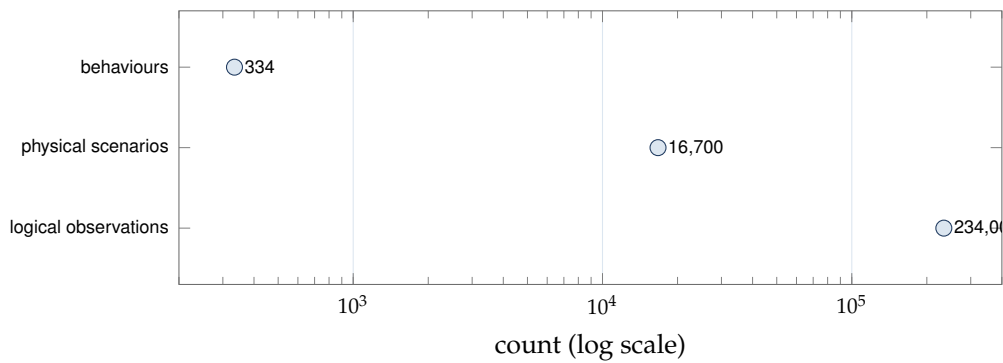


Figure 7.1: The cost grid is large numerically but highly redundant behaviourally. The logarithmic axis is used only for visibility; Table 7.1 carries the exact counts.

7.3 Why a continuous-looking parameter space collapses

For a particular surface/target pair, an alignment is chosen by comparing sums of primitive edit costs. Within a region of cost space where the ordering of all competing paths remains unchanged, the same traceback is selected and the same command is emitted. Behaviour

therefore changes at boundaries where two candidate paths exchange order; between those boundaries the induced symbolic model is piecewise constant.

This explains why nominal grid density is not a useful measure of model-space coverage. Ten adjacent numeric settings may represent ten different models, or one model repeated ten times. The same reasoning generalises beyond Radixor: whenever a discrete structure is selected by an optimisation over continuous or ordinal hyperparameters, reporting only the number of parameter settings can substantially overstate experimental diversity.

7.4 Sensitivity of quality inside the sampled grid

Behavioural collapse does not imply that the remaining behaviours are interchangeable. P2 evaluates the complete sampled-grid quality landscape in matched resource $\times$ split $\times$ coverage strata. Across 900 held-out strata, the median range in unseen changed-form exactness is only 0.0067 percentage points, with an interquartile range of 0–0.0608 and a 95th percentile of 0.360. The largest observed single-stratum range is 2.577 percentage points for Czech. The corresponding $F_{0.5}$ landscape is also usually flat but not identically so.

The statistically cautious reading is therefore two-sided. It would be wrong to claim that edit costs “do not matter”, because local resource-specific sensitivity is real and the structural object can change markedly. It would be equally wrong to infer that every numeric adjustment is meaningful, because the majority of sampled settings fall into extensive equivalence regions and most matched quality ranges are small.

7.5 Structural slack is larger than quality sensitivity

A particularly useful result is that model structure can often be reduced without a corresponding measurable loss in the reported held-out criterion. The full-grid structural minimum yields a smaller full-dictionary command vocabulary for 17 of 20 resources, with reductions exceeding 24% in some cases. Tied structural-minimum behaviours agree exactly on changed-form exactness in the matched strata, and the largest median paired exactness difference from the baseline reported by P2 is +0.041 percentage points.

This does not prove that fewer commands cause equal or better retrieval, and it does not identify a universal compression recipe. It reveals a practical property of the learned representation: the symbolic vocabulary contains resource-specific slack that can be explored as a structural objective rather than guessed from training-set size.

Table 7.2: A separation between exact facts about the frozen campaign and descriptive summaries of its observed resources.

Exact/deterministic	Descriptive/empirical
Command-sequence equivalence classes	distribution of quality ranges
Subset invariance	resource-level sensitivity differences
4,680 configurations $\rightarrow$ 334 behaviours	guarded operating points
16,700 physical scenarios realise 234,000 logical observations	associations among representation counts

Interpretation boundary. The 20 resources are an audited multilingual collection, not a probability sample of languages. Numeric summaries across them describe the campaign. The exact equivalence and subset-invariance statements are mathematical properties of the constructed finite objects; the cross-resource medians and correlations are not population estimates for “languages in general”.

7.6 Consequences for the rest of the programme

P2 changes how the remaining studies must be read in three ways.

First, a model condition is under-specified if it names the lexical resource but not the induction configuration. The cost vector is part of the route from policy evidence to executable artefact.

Second, hyperparameter searches should report *distinct induced behaviours* as well as parameter points. Without that count, robustness may be nothing more than repeated evaluation of the same model.

Third, the result connects naturally to the resource audit of Chapter 8. Both studies show that upstream representation choices change command support and compiled structure before runtime is ever measured. This is why Part II places edit-cost sensitivity and resource consistency before compiler semantics: they determine what object the compiler is asked to reduce.

Chapter 8

Resource Consistency as a Compiler Input

A lexicon-trained stemmer inherits more than vocabulary coverage from its training resource. Duplicate analyses, inconsistent canonical targets and uneven support for transformations alter the frequency with which commands are observed and therefore the symbolic vocabulary presented to the compiler. The resource is part of the model-construction process, not merely a source of examples.

This chapter synthesises **P6** (Galambos, 2026f). P6 does not ask whether one lexical database is linguistically better than another in the abstract. It asks a narrower systems question: after conservative consistency operations, how much does the learned transformation vocabulary change, and what does that change reveal about the mapping from lexical rows to executable commands?

8.1 The row-to-command map is many-to-one

A tempting mental model is that each suspicious lexical row creates one suspicious patch command. The actual compiler is less local. Many rows can induce the same normalised command; one row can cease to support a command that remains strongly supported elsewhere; removing a conflicting assignment can change the relative evidence for several commands at a shared context. Consequently, the effect of resource cleaning is mediated by *support topology*, not by a one-row/one-command correspondence.

This observation explains why lexical sanitation should not be evaluated only by counting edited rows. The relevant downstream quantities include the number of distinct commands, their support, the stored value references and the shape of the compiled trie.

8.2 Audit protocol

P6 freezes the resource universe and applies an ordered audit pipeline. The pipeline first resolves source-structural defects that can be identified without consulting the resulting stemmer. It then materialises candidate changes and measures their effect on the learned representation. A change is accepted as a resource correction only when its source justification is independent of the desired model outcome; model size is used as a diagnostic, not as a criterion for rewriting the linguistic source.

That separation protects the analysis against a common circularity. If a resource row is modified merely because the compiler becomes smaller or the benchmark improves, the resource is no longer an external source of evidence. It has become a tuned parameter.

P6 therefore distinguishes resource sanitation from model optimisation even when both happen to reduce command vocabulary. Figure 8.1 makes the ordering explicit. The resource itself remains immutable; auditing materialises a separate candidate, and only that candidate is recompiled and measured. This separation is not cosmetic. It is what allows the article to report positive, zero-change and no-safe-change outcomes under one frozen protocol instead of silently overwriting the source until the compiler looks cleaner.

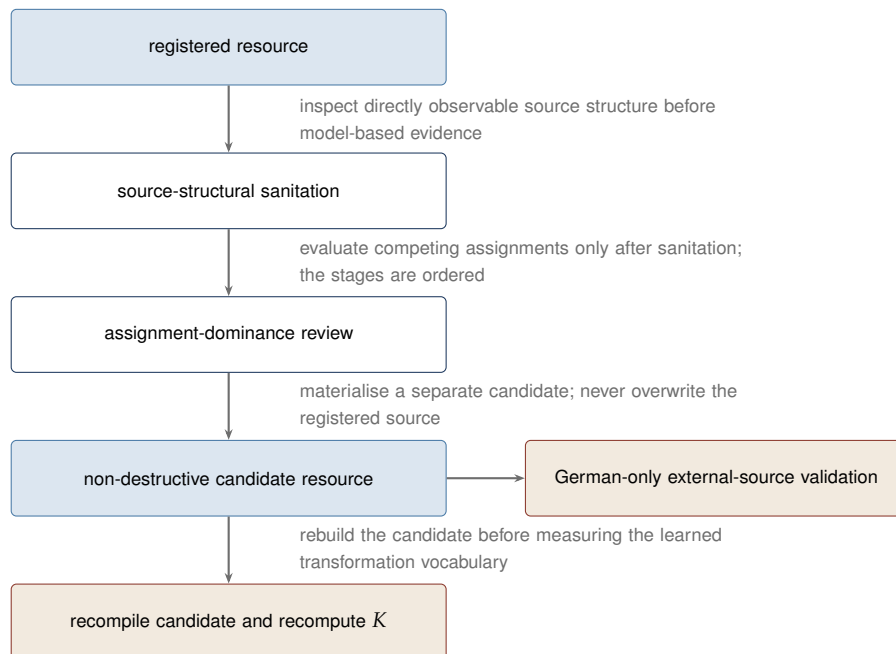


Figure 8.1: The ordered evidence path used in P6. The registered source is never rewritten in place; auditing creates a separate candidate whose induced command vocabulary is then recomputed. External-source validation is available only for the German case study.

8.3 Observed structural effect

P6’s audit census is informative precisely because its outcomes are mixed. The headline counts are not ten automatic “fixes” but ten materialised candidates chosen from a high-command audit queue, plus explicit no-safe-change cases. Dutch and Persian cross the same queue threshold as the edited resources yet do not admit an automatic intervention under the frozen evidence rules. Among the grouped-rule candidates, some lose many lexical assignments but no command at all because every affected command remains supported elsewhere; others lose commands through sanitation even when no dominance-qualified assignment is removed. The response variable is therefore the support topology of the learned transformation inventory, not the raw number of edited source rows.

The frozen census contains 12 high-command resource configurations. Ten non-destructive candidate resources are materialised, while Dutch and Persian are explicit no-safe-change outcomes under the automatic evidence rule. Nine candidates use the homogeneous grouped rule and German is kept separate because it uses a deliberately less conservative individual rule. Among the nine grouped-rule candidates, eight reduce distinct patch-command vocabulary. Their summed command count moves from 20,281 to

19,842, a 2.165% reduction, with a median per-resource reduction of 1.334%. German falls from 6,986 to 5,313 commands, a 23.948% reduction, and is not pooled into that headline.

Table 8.1: Published structural outcomes of P6. The German case is reported separately because it uses an external-source validation path rather than being pooled as another draw from a common population.

Scope	Before	After	Interpretation
Grouped audited stratum	20,281	19,842	−2.165% summed command vocabulary; median resource change −1.334%
German external-source case	6,986	5,313	−23.948% command vocabulary

The German result is deliberately not generalised to the other languages. It is useful because it demonstrates the maximum kind of structural effect that a resource inconsistency can have in a real audited case, not because it estimates an expected reduction for arbitrary lexicons.

8.4 The informative cases are the non-linear ones

The central P6 result is not simply that command vocabulary becomes smaller. It is that lexical edits and learned commands are connected through many-to-one support topology. The most useful cases are therefore those in which source edits and command counts move differently.

Table 8.2: Selected audited cases that demonstrate why row edits, command vocabulary and package bytes are different measurement units.

Case	Observed change	Mechanistic interpretation
Finnish	3 dominance-qualified assignments removed; $\Delta K = 0$	every affected command retains support elsewhere
Hebrew	0 dominance-qualified assignments removed; 41 commands disappear	source-structural sanitation changes the accepted evidence before dominance filtering
German	2,055 dominance-qualified assignments removed; 1,673 fewer commands	mapping count and command count are different units; staged K values are unavailable
Polish UniMorph	3 commands disappear while candidate GZip grows by 121 bytes	candidate materialisation adds a 237-byte provenance block and reserialises accepted rows; package bytes are not model complexity
Dutch / Persian	baseline vocabularies 538 / 704 commands; no automatic candidate	crossing the audit threshold is insufficient evidence for a safe intervention

The Polish case is particularly useful because it falsifies a tempting shortcut: compressed artifact size cannot be substituted for lexical payload or compiled structure when the baseline and candidate are not serialised under an identical metadata policy. P6 therefore demotes package-byte deltas to provenance diagnostics. A controlled footprint study would have to canonicalise both baseline and candidate through the same serializer and

separately measure trie nodes, edges, value references, memory and runtime.

German also supplies the only external-source relation-level check in P6. The candidate improves the frozen micro CISTEM-style score under both related CELEX2-derived gold policies, primarily through recall recovery with a small precision increase. Because the two standards share the same CELEX2 source, that agreement is directional corroboration, not two independent external replications.

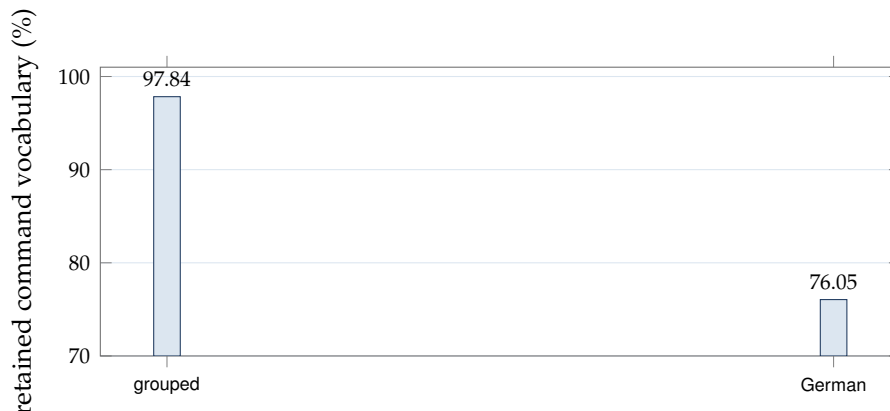


Figure 8.2: Relative command vocabulary remaining after the audited changes. The bars are not estimates of a common language-level effect; they show two differently scoped audit results.

8.5 What the reduction means—and what it does not mean

A smaller transformation vocabulary is not automatically a better stemmer. Redundancy can be harmless, and a rare command can encode an important irregular pattern. The structural reduction is valuable because it reveals that source inconsistency can propagate non-locally into the executable model. It provides a triage signal for resource review and a mechanism for explaining why two seemingly similar lexical resources can compile differently.

The result also sharpens the interpretation of P2. Edit costs and lexical consistency are different upstream perturbations, but both alter the support from which commands are induced. A construction protocol is therefore not fully identified by naming the compiler and the language. It includes the specific lexical source, its version and audit state, as well as the induction hyperparameters.

A second lesson concerns validation scope. German is reported separately not only because the induced change is large, but because it is the only case with a deliberately less conservative individual rule and an external-source check against two related CELEX2/CISTEM-derived policies. Its positive direction is therefore useful corroboration, yet it is not an independent multilingual replication of the grouped-rule audit. The monograph keeps these strata apart so that a strong case study does not silently inflate a cross-resource claim.

Interpretation boundary. The audit does not license the rule “minimise the number of commands”. A model with fewer commands can be worse, equal or better depending on which transformations were removed. Command-vocabulary reduction is evidence of a structural consequence of resource consistency; quality and retrieval require separate measurements.

8.6 Why this chapter precedes compiler semantics

The compiler can only preserve or generalise the object it receives. P6 shows that the object already depends on the topology of support in the lexical resource. In the causal chain of the programme, the order is therefore resource $\rightarrow$ induced commands $\rightarrow$ compiled structure, not resource $\rightarrow$ compiler as if induction were transparent. This is why resource consistency belongs in the mechanism part of the monograph rather than in an appendix on data cleaning.

Chapter 9

Two Compiler Operations, Two Semantic Contracts

The word *reduction* hides two different operations in the Egothor/Radixor compiler. One operation shares recursively equivalent substructures; the other replaces a uniform represented region by an accepting leaf. Both can make the model smaller, but only the first is purely a representation change. The second can make a command decision apply to inputs whose continuations were not represented before.

This distinction is the central formal contribution of **P7** (Galambos, 2026g). It explains why a compiled lexicon-trained model can be compact and still exhibit rule-like behaviour on unseen inputs, while also showing exactly where a proof of semantic preservation ends and an empirical claim about linguistic correctness must begin.

9.1 Canonicalisation: quotienting without changing selection

Fix a reduction mode μ . Each node exposes a mode-specific observable $\text{obs}_\mu(n)$: for example, a preferred command or another declared view of the node's candidate information. The compiler may share two rooted substructures only when their observables and all labelled continuations agree recursively.

Definition 9.1 (Reduction signature). The mode- μ signature of a node is

$$\text{sig}_\mu(n) = \langle \text{obs}_\mu(n), \{(x, \text{sig}_\mu(n_x)) : n \xrightarrow{x} n_x\} \rangle,$$

where children are ordered canonically by edge label.

Definition 9.2 (Canonicalisation quotient). For nodes n, n' , let $n \equiv_\mu n'$ iff $\text{sig}_\mu(n) = \text{sig}_\mu(n')$. Canonicalisation replaces each equivalence class by one representative and redirects incoming edges to it.

The construction belongs to the classical family of minimisation and acyclic canonicalisation results (Daciuk et al., 2000; Mihov and Maurel, 2001; Mohri, 2000; Nerode, 1958; Revuz, 1992). The point of P7 is not to claim a new Myhill–Nerode theorem; it is to state the correct observable for this value-carrying patch-command structure and prove what the actual mode-specific compiler is allowed to share.

Theorem 9.3 (Selection preservation). *For every public reduction mode μ , $\equiv_\mu$ is a congruence for labelled edge traversal. Canonicalisation therefore preserves command selection:*

$$C_{\mathcal{T}}(w) = C_{\mathcal{T}/\equiv_\mu}(w) \quad \text{for all } w \in \Sigma^*.$$

Proof sketch. Equal signatures contain equal current observables and equal labelled child signatures. Traversal from equivalent nodes therefore remains in equivalent classes after every consumed character. The sequence of value-bearing observables along a represented path is unchanged, so the deepest applicable command is unchanged. $\square$

Proposition 9.4 (Insertion-order stability). *When children are represented by the canonical set of labelled child signatures, the computed signature is independent of the order in which those children were inserted.*

Theorem 9.5 (Relative minimality of the canonical DAG). *In the contraction-free exact-trie phase, equality of reduction signatures produces one state per distinct rooted, edge-labelled, mode- μ observable subtrie. No deterministic acyclic representation obtained solely by sharing such equivalent represented subtrees can use fewer states while preserving the same represented paths and node observables. The quotient remains acyclic and bottom-up construction terminates.*

Interpretation boundary. Theorem 9.5 is relative to a specific representation class and precedes accepting-leaf contraction. It is not a theorem that the production file is globally minimal in bytes, that no other transducer encoding can be smaller, or that the resulting model has minimum runtime. Those are different optimisation problems.

Algorithm 9.1: CANONICALISE($\mathcal{T}, \mu$) – share equal semantic subtrees

Input: exact trie $\mathcal{T}$; reduction mode μ

Output: DAG quotient preserving the selected observable

```

1  $R \leftarrow$  empty map from signatures to representative nodes;
2 foreach node  $n$  in reverse topological order do
3    $\sigma \leftarrow \langle \text{obs}_\mu(n), \{(x, \text{sig}_\mu(n_x))\}_x \rangle$  in canonical edge order;
4   if  $\sigma \in R$  then
5     | redirect incoming edges of  $n$  to  $R[\sigma]$  and discard  $n$ 
6   else
7     |  $R[\sigma] \leftarrow n; \text{sig}_\mu(n) \leftarrow \sigma;$ 
8 return  $\mathcal{T}$ ;
```

Remark 9.6 (Reference-implementation complexity). A naively materialised recursive signature can perform quadratic total work on a degenerate chain because long descendant descriptions are repeatedly incorporated. This is a reference-implementation bound, not a claim about classical linear acyclic minimisers. The production implementation can cache or hash incrementally, but the formal presentation records the worst case of the stated construction.

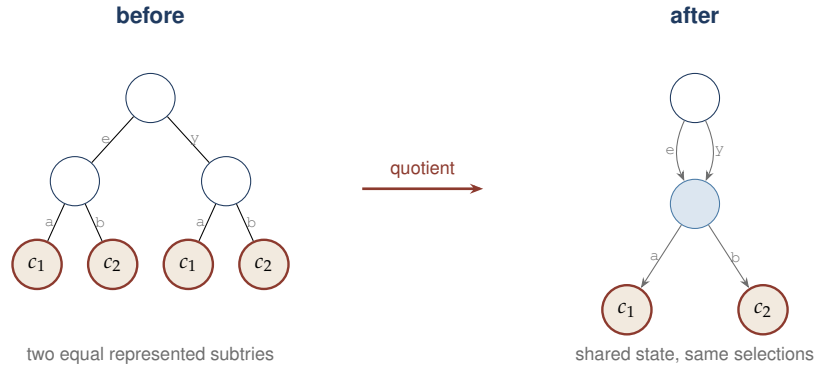


Figure 9.1: Equal recursively observed substructures are represented once. The set of represented paths and selected commands is unchanged.

9.2 Accepting-leaf contraction: generalising a decision

Canonicalisation cannot explain why a word whose continuation was never present in the exact trie may nevertheless receive a command. That behaviour comes from a second operation.

Definition 9.7 (Locally uniform represented region). A region rooted at n is locally uniform with value v when every relevant value on the represented training support below n is compatible with v under the active mode. Missing outgoing continuations are not part of that represented support.

Definition 9.8 (Accepting-leaf contraction). A uniform region is replaced by an accepting leaf carrying v . Traversal that reaches the leaf returns v immediately, without inspecting the remaining input characters.

Theorem 9.9 (Value-decision extension). Let $\mathcal{T}'$ be obtained by contracting a locally uniform region rooted at n with value v . Then (i) every represented training path through the region selects the same value as before; (ii) any previously unrepresented continuation that reaches n now receives the explicit decision v ; and (iii) command selection is unchanged for all training inputs.

Proof sketch. Uniformity guarantees that every represented valued state in the region agrees with v , so replacing the represented remainder by a leaf cannot change a training-path decision. A continuation that formerly encountered a missing edge now stops at the accepting leaf and receives v . Thus represented behaviour is preserved while the domain on which v is an explicit decision is expanded. $\square$

9.3 The generalised object is a command selection, not a stored stem

Suppose a contracted leaf carries “delete the final three characters”. Two unseen words reaching that leaf do not receive the same output string. Each receives the same *program*, executed against its own characters. The compiler therefore generalises a transformation decision, not a dictionary answer.

This observation is the formal bridge to P1. Theorem 9.9 shows that the decision domain can expand. P1 measures whether the resulting transformations recover targets on held-out

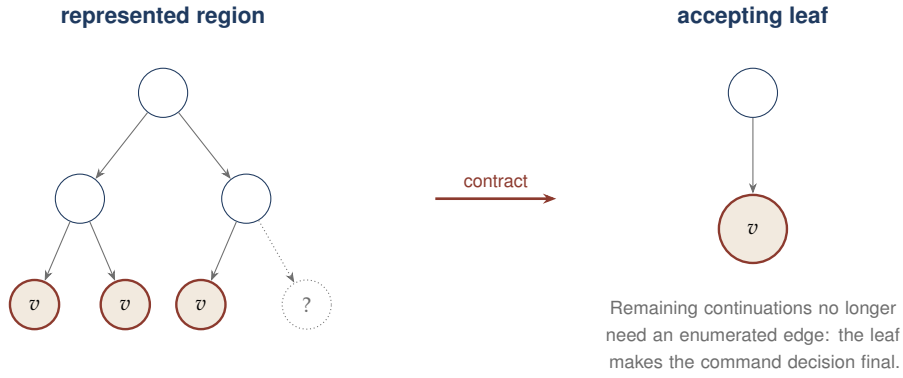


Figure 9.2: Training-path selections remain unchanged, while continuations that were not explicitly represented can now inherit the uniform command decision.

Algorithm 9.2: $\text{CONTRACT}(\mathcal{T})$ – replace uniform represented regions

Input: canonicalised structure $\mathcal{T}$

Output: structure with accepting-leaf contractions

```

1 foreach node  $n$  in reverse topological order do
2    $U \leftarrow \text{UNIFORMVALUE}(n)$ ;
3   if  $U \neq \perp$  and  $n$  is eligible for contraction then
4      $\perp$  discard descendants of  $n$ ; mark  $n$  accepting;  $\text{val}(n) \leftarrow U$ ;
5 return  $\mathcal{T}$ ;

```

lexical families. The theorem cannot answer the empirical question, and the holdout result cannot replace the theorem about what the compiler actually did.

Interpretation boundary. Accepting-leaf contraction preserves the declared value decision on represented training paths and extends that decision to additional continuations. It does not prove that the extended decision is morphologically correct. Linguistic or retrieval correctness remains an empirical property of the newly covered inputs.

9.4 Implementation verification accompanies the proof

P7 is not only a paper proof detached from code. Its frozen artifact maps the formal objects back to the Java compiler and adds three layers of corroboration: mode-specific unit cases for semantic canonicalisation, explicit accepting-leaf cases for domain extension, and property-based tests over generated tries. The publication package also records source-file identities and a theorem-to-source correspondence so that later implementation changes can be checked for semantic drift.

This evidence has a deliberately limited role. Tests do not prove the theorem, and the theorem does not prove that every future implementation preserves the same contract. Together they close a common systems-research gap: the formal statement describes the observable actually implemented by the frozen compiler, while the tests exercise boundary cases where an accidental change would be most likely to invalidate that correspondence.

No multilingual size reduction or runtime percentage is needed to regenerate the

principal P7 claims. Canonicalisation soundness and accepting-leaf domain extension are mathematical consequences of the frozen algorithm; multilingual compression and latency are separate empirical questions addressed elsewhere in the programme.

9.5 What P7 contributes to the programme

Table 9.1: The compiler claims are deliberately split by operation and scope.

Result	Interpretation
Selection-preserving canonicalisation	representation sharing under the mode-specific semantic signature
Relative minimality	minimal only within the stated equivalent-subtrie DAG representation class before contraction
Acyclic finite construction	quotienting equal signatures does not introduce cycles
Accepting-leaf domain extension	uniform command decisions become explicit for previously unrepresented continuations
Complexity caveat	the reference recursive signature construction is not claimed to match optimal classical minimiser complexity

The chapter’s broader contribution is methodological: “compression” is not a sufficient semantic description of a compiler transformation. Some reductions are observational quotients; others deliberately change the domain of a decision. Once those two cases are separated, the rest of the Radixor evidence can refer to the compiled model without ambiguity about where generalisation entered the pipeline.

Part III

Measuring the Realised Policy

Chapter 10

Evaluating Conflation as a Relation

Literal stem equality is a poor universal evaluation contract. Two systems may choose different representative strings while grouping exactly the same forms; a lexical resource may allow one surface form to belong to several legitimate groups; and a multi-output analyser may not define one deterministic partition at all. The evaluation framework used by the Radixor programme therefore starts from a pairwise relation rather than from exact output labels.

The framework is the methodological core behind **P3** (Galambos, 2026c). It is also the part of the monograph least tied to Radixor: once a system's output is mapped to a declared single-output or candidate-output policy, the counts depend only on the resulting relation and the reference relation.

10.1 Reference covers can overlap

Let V be the evaluation vocabulary and let $G_1, \dots, G_k$ be lexical groups from a reference resource. A form may occur in more than one group. The gold relation is therefore defined directly on pairs.

Definition 10.1 (Overlapping reference relation). For distinct $u, v \in V$,

$$R(u, v) = 1 \iff \exists i : u \in G_i \wedge v \in G_i.$$

No disjoint-partition assumption is made.

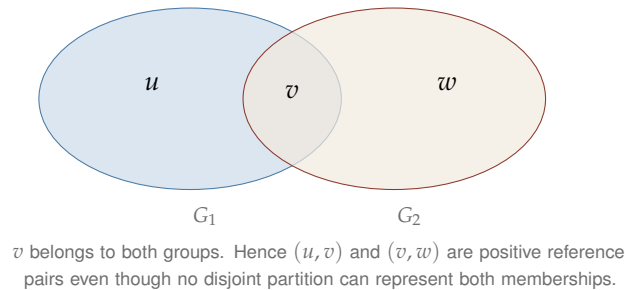


Figure 10.1: The evaluation object is the union of within-group pair relations. A surface form can participate in several groups without forcing an arbitrary disambiguation.

This choice avoids a conceptual error: applying an adjusted Rand index or other partition metric after first forcing an overlapping resource into disjoint classes changes

the reference object. The evaluation would then measure the arbitrary disambiguation procedure as well as the stemmer.

10.2 Predicted relation and pairwise counts

For a deterministic single-output system with mapping h , the predicted pair relation is $P(u, v) = 1$ iff $h(u) = h(v)$. Comparing P with R yields the four usual pairwise counts.

Definition 10.2 (Pairwise confusion counts). Over all unordered distinct pairs $\{u, v\} \subseteq V$:

$$TP = |\{\{u, v\} : R = 1, P = 1\}|,$$

$$FN = |\{\{u, v\} : R = 1, P = 0\}|,$$

$$FP = |\{\{u, v\} : R = 0, P = 1\}|,$$

$$TN = |\{\{u, v\} : R = 0, P = 0\}|.$$

Reference-positive pairs are usually extremely sparse relative to all possible pairs. That imbalance is why raw accuracy is nearly useless: a system that keeps almost everything separate can obtain enormous pairwise accuracy while missing the relation the benchmark was constructed to test.

The population unit in the confusion matrix is a *pair*, but the pairs are not independent observations. They share forms, families and resource structure. Counts are exact descriptive summaries of a finite relation; they must not be fed into an i.i.d. binomial test as though every pair were an independent trial.

10.3 Metrics are views of the same finite counts

Table 10.1 records the principal scalar summaries used in the programme. The monograph treats them as views, not as competing definitions of one true quality number.

Balanced accuracy is useful when a scalar is required because it does not let the huge negative-pair population dominate raw accuracy. It is not a universal utility function. P3 therefore moves to the (OI, UI) plane before drawing comparative conclusions.

10.4 What the finite multilingual census shows about metric choice

The relation framework is not only a formal convenience. P3's checked multilingual census shows why the choice of scalar summary changes substantive comparative conclusions. In the complete-resource matrices, reference-positive pairs are extremely sparse: the median positive-pair density is only 0.0135%, corresponding to a median negative-to-positive pair ratio of about 7,556. Raw pair accuracy therefore gives overwhelming numerical weight to the negative class and can prefer a system for reasons almost unrelated to the reference conflations that motivated the benchmark.

P3 recomputes comparator optima from the same finite pair counts under several scalars. Relative to balanced accuracy, the set of tie-aware best non-Radixor comparators is disjoint in 12 of 38 matrices for raw pair accuracy, 28 for positive-pair precision, 15 for $F_{0.5}$, and 12 for both F_1 and Matthews correlation. These are not disagreements between datasets

Table 10.1: Metrics derived from the same pairwise counts. Undefined denominator cases are reported as undefined, not silently replaced by zero.

Metric	Formula	Interpretation
Under-conflation UI	$FN/(TP + FN)$	fraction of reference-positive pairs missed
Over-conflation OI	$FP/(TN + FP)$	fraction of reference-negative pairs falsely joined
Balanced accuracy BA	$\frac{1}{2}[TP/(TP + FN) + TN/(TN + FP)]$	symmetric scalar summary of the two class recalls
Precision of positive pairs	$TP/(TP + FP)$	purity of predicted conflations
Recall of positive pairs	$TP/(TP + FN)$	coverage of reference conflations
F_β	standard precision/recall harmonic weighting	explicit scalar preference between positive-pair precision and recall

or implementations: they are different orderings of the same measured relation under different utilities.

Table 10.2: P3 sensitivity summary over 38 comparative matrices. Counts report matrices in which the best non-Radixor comparator set under the named scalar is disjoint from the set selected by balanced accuracy.

Alternative scalar	Disjoint best-comparator sets / 38
Raw pair accuracy	12
Positive-pair precision	28
$F_{0.5}$	15
F_1	12
Matthews correlation coefficient	12

This sensitivity result is one reason the monograph refuses to identify “stemming quality” with a single scalar. Balanced accuracy is a useful class-balanced view, but Pareto dominance in Chapter 11 is the comparison that does not require a prior exchange rate between false splits and false merges.

10.5 Candidate-aware outputs change the relation, not merely the score

P3 also evaluates systems that expose more than one candidate analysis. The complete-resource candidate-aware census contains 31 language–system units. Activating all available candidates improves $F_{0.5}$ in 29 of those units and worsens it in two. This trajectory is constrained rather than arbitrary: admitting more candidates can only add predicted positive relations under an intersection-based candidate policy, so recall-like behaviour tends to improve while false merges can accumulate.

The result is useful because it separates two questions that are often collapsed. A multi-output analyser can contain potentially useful evidence that its primary output does not expose, yet the policy used to turn those candidates into pair relations still determines whether that evidence is beneficial. The scorer must therefore declare both the candidate-generation system and the output policy under which candidates become retrieval-equivalent.

10.6 Output policy must be declared

A multi-output analyser complicates the predicted relation. Suppose system $A(w)$ returns a candidate set rather than one key. Several reasonable policies exist: use the first candidate, use a preferred candidate, treat any shared candidate as evidence of conflation, or score a trajectory as more candidates are admitted. These policies are not algebraically interchangeable.

The tempting ANY_CANDIDATE rule—declare u and v related whenever $A(u) \cap A(v) \neq \emptyset$ —can fail transitivity. For example, $A(u) = \{a\}$, $A(v) = \{a, b\}$ and $A(w) = \{b\}$ make u related to v and v related to w but not u to w .

Proposition 10.3 (Candidate overlap need not define an equivalence relation). *The relation $u \sim v$ iff $A(u) \cap A(v) \neq \emptyset$ is reflexive and symmetric for non-empty candidate sets but need not be transitive. It therefore cannot in general be represented by one deterministic partition, although its pairwise confusion counts remain well-defined.*

This is another reason to keep the evaluation at the relation level. Pairwise comparison remains valid even when the predicted relation is not an equivalence relation.

10.7 Comparison rules before statistics

The programme uses three rules before any scalar interpretation.

Same population. Two candidates can be compared only when their TP, FP, FN, TN counts refer to the same vocabulary and the same reference pair universe. Missing outputs must be handled by a declared policy rather than by silently changing the denominator.

Same output contract. Deterministic outputs and candidate-set semantics are reported separately unless an adapter makes the relation contract identical.

Exact ties remain ties. When two candidates collapse to the same aggregate error vector, the comparison records that equivalence instead of inventing a ranking from implementation labels.

Those rules are what make Chapter 11’s dominance relation interpretable.

10.8 Inference follows the experimental unit

Because the pair counts share forms and groups, uncertainty should be attached to the unit that was sampled or perturbed: lexical family, resource, seed, query, or another design-level cluster. The current studies mostly report exact finite-population counts and descriptive cross-resource summaries. When inferential intervals are added, C6 in Chapter 18 recommends cluster-aware resampling rather than pairwise i.i.d. tests.

Interpretation boundary. A high agreement score against a lexical resource is evidence of agreement with that declared resource policy. It is not automatically evidence of linguistic truth or retrieval effectiveness. Conversely, a retrieval gain can justify a task-specific policy without proving that the policy is a better linguistic lemmatisation. The three registers remain distinct.

10.9 Why the scalar is postponed

Once UI and OI are explicit, a candidate can be worse on both, better on both, or trade one error for the other. Only the first two cases admit an exchange-rate-free ordering. Chapter 11 therefore begins with Pareto dominance, adds recursive strata, and introduces an explicit pair-cost ratio only after the non-dominated geometry is visible.

Chapter 11

Pareto Geometry of Conflation Errors

Any scalar stemmer score embeds an exchange rate between two qualitatively different mistakes: failing to join forms the reference groups together and joining forms the reference keeps apart. That exchange rate may be useful for a particular application, but hiding it inside one headline number makes method comparison look more objective than it is.

P3 (Galambos, 2026c) therefore analyses candidates first in the (OI, UI) error plane. It uses dominance to identify unconditional inferiority, recursive fronts to summarise depth, qualitative regimes to interpret positions on the front, and exact support intervals to show which frontier points are optimal under some positive linear pair-cost ratio.

11.1 Dominance is the comparison that needs no exchange rate

Definition 11.1 (Pareto dominance). Candidate a dominates candidate b when

$$OI_a \leq OI_b, \quad UI_a \leq UI_b,$$

with at least one strict inequality.

A dominated candidate is worse on one error axis and no better on the other. No assumption about the relative cost of the errors is needed. Non-dominated candidates are different: choosing among them requires an application preference or another objective.

Definition 11.2 (Recursive Pareto strata). Stratum 1 is the non-dominated subset of candidate set S . Remove it; stratum 2 is the non-dominated subset of what remains. Continue until every candidate has a stratum index.

This is non-dominated sorting used as an evaluation summary (Deb, 2001). Depth is conditional on the candidate set, not an intrinsic property of a system. Add a strong new candidate and several existing systems can move to a deeper stratum without changing one output.

11.2 The measured candidate set has four strata

In the checked P3 snapshot, 190 deterministic implementation rows collapse to 174 distinct aggregate error vectors. Those vectors occupy four recursive Pareto strata containing 40, 97, 27 and 10 vectors. The collapse is itself an important warning: distinct implementations or settings can become identical at the aggregate-error level even when their per-form outputs differ.

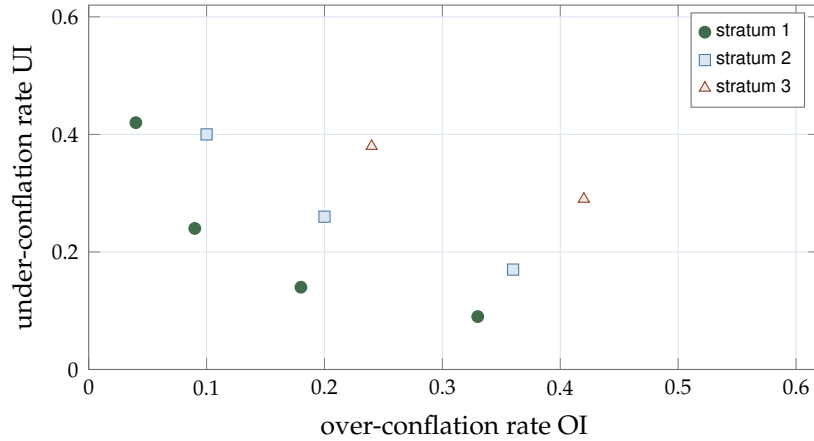


Figure 11.1: Schematic illustration of non-dominated fronts. Point locations are explanatory, not measured values.

Proposition 11.3 (Aggregate error equivalence is weaker than output equivalence). *Two systems can disagree on individual form pairs while having identical (TP, FP, FN, TN) counts. Equality of aggregate error vectors therefore does not imply equality of the predicted relation.*

The candidate-level analysis is nevertheless useful because it answers a specific question: which measured systems are unconditionally inferior on both error rates, and how many non-dominated fronts separate them from the current frontier?

11.3 Radixor shifts the within-reference frontier

The strongest empirical result in P3 concerns the primary ALL_WORDS view. Across all 19 languages for which comparator systems are available, Radixor is the sole stratum-1 error vector and has both fewer false-negative reference pairs and fewer false-positive reference pairs than all 75 comparator implementation rows (69 distinct comparator error vectors). The nested lowercase-only view reproduces the same component-wise ordering for its 75 comparator rows.

That result is stronger than a scalar win because it is strict two-axis dominance, but its margins are asymmetric. Table 11.1 reports the nearest-comparator gaps summarised over the 19 complete-resource matrices. The under-conflation advantage is uniformly substantial in rate terms, whereas the over-conflation ordering is numerically thin in a few matrices.

Table 11.1: P3 complete-resource Radixor margins over 19 comparative matrices. Small OI margins are reported explicitly because strict dominance should not be rhetorically detached from annotation uncertainty.

Nearest-comparator gap	Minimum	Median
FN pairs	1,237	85,841
UI percentage points	2.197	21.992
FP pairs	3	4,263
OI parts per million	0.000211	0.863

The same-resource boundary is essential. Radixor is induced from the lexical resource family that defines the evaluated relation, whereas most comparators are not. The finding

therefore demonstrates very strong adaptation to the declared inflectional policy; it is not independent evidence of universal linguistic superiority on derivational, contextual, or task-specific gold standards.

11.4 Frontier position has different meanings

A non-dominated point can occupy a recall-heavy edge, a specificity-heavy edge, a supported interior compromise, or an “unsupported” efficient location that is never optimal for the particular linear cost family introduced below.

Table 11.2: The labels describe position relative to the measured frontier and one declared linear pair-cost family; they are not universal quality grades.

Regime	Meaning
Recall edge	low UI at the price of higher OI
Specificity edge	low OI at the price of higher UI
Supported compromise	frontier point optimal for a non-empty interval of positive linear pair-cost ratios
Unsupported efficient	frontier point optimal for no positive ratio in that linear family

The term “unsupported” must be read literally. It does not mean irrational, bad, or unusable. A non-linear utility, a third objective, a hard constraint or a deployment-specific preference can select the point. It means only that no member of the one-parameter linear pair-cost family makes it the minimum among the current candidates.

P3 makes these regimes empirical rather than merely schematic. Among the 97 stratum-2 error vectors, 30 are recall edges, 30 specificity edges, 20 supported compromises, nine unsupported-efficient points, and eight single-point fronts. A recursive Pareto depth therefore does not define one homogeneous “runner-up” class.

German provides a concrete cost interpretation after the dominating stratum-1 point is removed. Under $C_\rho = \rho FN + FP$, German Minimal is selected for $0 < \rho \leq 0.0659$, direct Snowball for approximately $0.0659 \leq \rho \leq 0.700$, and CISTEM for $\rho \geq 0.700$. The ordering is not a ranking of implementation quality: it is an explicit map from an application’s relative cost of false splits and false merges to the preferred point on that conditional frontier.

Table 11.3: Exact support intervals reported by P3 after conditioning on the German second stratum.

System	Regime	ρ support
German Minimal	specificity edge	$(0, 0.0659]$
Snowball direct	supported compromise	$[0.0659, 0.700]$
CISTEM	recall edge	$[0.700, \infty)$

The implementation-category profiles reinforce the same point. In the complete-resource view all 20 Radixor vectors are in stratum 1; all seven minimal-filter vectors are in stratum 2, six at the specificity edge; Snowball language stemmers span strata 2–4; Hunspell places six of eight vectors in stratum 2; and Morfologik places two of three there. These profiles describe how particular implementations interact with the declared reference policy, not immutable quality grades attached to software families.

11.5 Making the exchange rate explicit

Definition 11.4 (Linear raw pair cost). For $\rho > 0$, define

$$C_\rho = \rho FN + FP.$$

ρ states how many false merges are considered equivalent to one false split in raw pair-count space.

Because all candidate costs are affine functions of ρ , the set of ratios for which one candidate is optimal can be computed exactly from pairwise line intersections.

Proposition 11.5 (Exact support intervals). *For a finite candidate set evaluated on one common pair universe, the set of $\rho > 0$ for which candidate a minimises C_ρ is an interval, possibly empty or unbounded. Its endpoints are determined exactly from the integer (FN, FP) counts.*

Algorithm 11.1: STRATIFY(S) – dominance depth and linear-cost support

Input: finite candidate set S with common-universe pair counts

Output: stratum and, for first-front points, cost-support interval

```

1  $R \leftarrow S; i \leftarrow 1;$ 
2 while  $R \neq \emptyset$  do
3    $F \leftarrow$  candidates in  $R$  dominated by no member of  $R$ ;
4   assign stratum  $i$  to every member of  $F$ ;
5    $R \leftarrow R \setminus F; i \leftarrow i + 1;$ 
6 foreach  $a$  in stratum 1 do
7   intersect the half-lines defined by  $C_\rho(a) \leq C_\rho(b)$  for all  $b \in S$ ;
8   classify the resulting interval as recall-edge, specificity-edge, supported compromise, or unsupported;
```

The support calculation makes the scalar assumption auditable. A paper that selects $\rho = 1$ has chosen an exchange rate; it has not discovered a neutral one. The ratio must also be interpreted together with the imbalance of the positive and negative pair populations from which FN and FP were counted.

11.6 Candidate-set dependence is measured, not hidden

P3 explicitly perturbs the candidate set. After removing one non-Radixor error vector, 551 of 592 retained-vector assignments remain unchanged and 41 improve by one stratum. The result illustrates both stability and relativity: most assignments survive the deletion, but the affected candidates move even though their outputs do not.

The study also compares two nested processing modes; stratum identity is preserved for 88 of 95 implementation-language units. Again, the correct interpretation is descriptive stability under the tested perturbation, not an intrinsic immutable grade.

Interpretation boundary. Pareto stratum, regime and support interval are conditional on the language, reference relation, output-policy adapter, processing mode and candidate set. They summarise a declared comparison. They should never be quoted as if they were properties attached permanently to a software package.

11.7 Why an external floor is still needed

A candidate-relative framework benefits from a reproducible weak baseline. Paice's truncation-relative work supplied such an anchor in a different scalar form (Paice, 1994). The present Pareto analysis should include a simple length-truncation family—and potentially a character- n -gram normalisation reference—so that future candidate additions do not leave the frontier without an explicit low-information floor (Mayfield and McNamee, 2003).

That baseline is C3 in Chapter 18. More importantly, C4 applies the entire framework to external stemmer families. Only that experiment can show whether the strata and regimes are a useful field-level resource rather than merely an elegant description of the current candidate set.

Generalisation Beyond Enumerated Lexical Families

Compiler semantics can show that a command decision extends to an unseen continuation; it cannot show that the decision is linguistically useful. **P1** (Galambos, 2026a) supplies the missing empirical test. Its central question is deliberately narrow: when entire lexical families are excluded from model construction, can transformations induced from the remaining families recover canonical targets for changed surface forms that were not present in the training-surface set?

The distinction between “unseen form” and “unseen family” is decisive. A protocol that hides one inflected form but leaves sibling forms of the same paradigm in training measures near-neighbour completion. P1 instead removes families atomically and adds explicit surface-leakage control.

12.1 Protocol: family-level holdout

The study uses 20 UniMorph-derived language resources. For each resource, five deterministic family rankings define nested coverage conditions from 10% through 90%. A family and all of its variants are either available for construction or held out together. The protocol yields 1,000 nominal resource–seed–coverage scenarios, of which 900 contain a non-empty primary evaluation population.

Algorithm 12.1: Family-holdout evaluation used by P1

Input: lexical families F ; deterministic ranking seed s ; coverage k

- 1 rank families in F using the frozen deterministic procedure for s ;
- 2 $F_{train} \leftarrow$ first k proportion of families;
- 3 $F_{test} \leftarrow F \setminus F_{train}$;
- 4 construct the model from all rows in F_{train} ;
- 5 $S_{train} \leftarrow$ set of all surface strings occurring in F_{train} ;
- 6 **foreach** changed surface w in F_{test} **do**
- 7 retain w in the primary scope only if $w \notin S_{train}$;
- 8 execute the trained model and compare with the held-out target;
- 9 aggregate within the declared resource, seed and coverage strata;

The surface exclusion matters because lexical resources can contain the same string in more than one family. A family may be held out while one of its surface strings remains

visible elsewhere. Counting that occurrence as unseen would overstate transfer.

The same mechanism also imposes a measurable ceiling on exact recovery. If a surface string occurs only in held-out families but is paired there with more than one distinct target, then no deterministic context-free analyser can be correct on every such occurrence simultaneously. P1 therefore speaks carefully of held-out *lexical families* rather than implying that the protocol reconstructs fully disambiguated morphological paradigms. The aim is to measure transferable transformation behaviour under controlled leakage, not to attribute every residual error to a failure of the command formalism.

12.2 Three evaluation scopes

P1 retains multiple scopes so that the effect of increasingly strict leakage control is observable rather than implicit.

Table 12.1: Only the final row is the primary transfer estimand. The broader scopes remain useful diagnostics because they show what each restriction removes.

Scope	Population	Interpretation
held-out	all forms in held-out families	family holdout without controlling surface reuse
changed	held-out forms whose target differs from the surface	removes trivial identity mappings
unseen changed	changed held-out forms whose surface string never occurs in training	primary evidence for transfer beyond exact membership

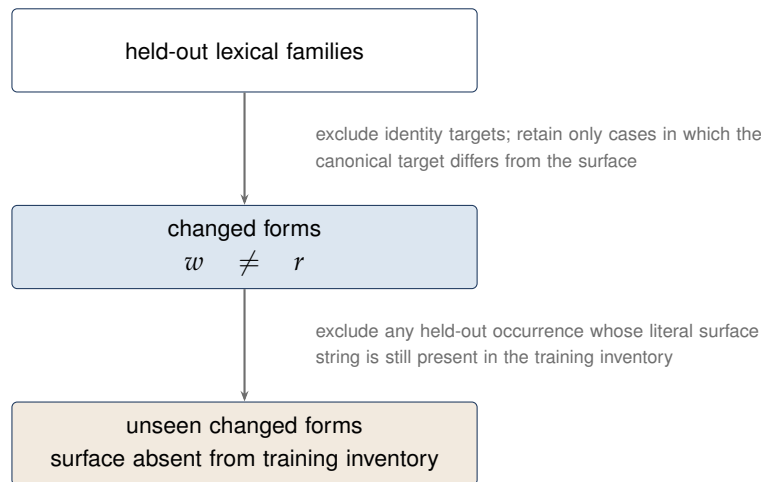


Figure 12.1: The primary “unseen changed” population is obtained by successively removing trivial identity targets and any held-out occurrence whose literal surface string is still present in the training inventory.

12.3 Result: measurable within-resource transfer

At 10% training coverage, the resource-macro mean exact recovery on unseen changed forms is 60.49%; at 90% coverage it is 69.07%. The corresponding across-resource medians

are 63.72% and 70.44%. All 20 resource medians are higher at the 90% endpoint than at the 10% endpoint, with a median descriptive endpoint difference of 7.76 percentage points.

The result rejects the “compressed lookup table” interpretation. Under family-level hold-out and exact-surface leakage control, a model induced from one set of lexical families applies learned patch-command behaviour successfully to many changed forms from families that did not participate in construction.

The measurement is stronger than simply observing non-identity output on OOV words, but weaker than an out-of-domain linguistic generalisation claim. The training and test material come from the same resource framework and are predominantly inflectional. The increasing endpoint numbers also do not by themselves identify a causal learning curve, because the test population changes as coverage changes.

12.4 What the study does not establish

P1 does not compare Radixor with a trivial suffix baseline on the same held-out families. It therefore measures transfer in absolute terms but does not yet say how much of that transfer requires the full patch-command machinery. It also does not use an independent lexical source for testing, and it does not measure retrieval. Those gaps are explicit rather than defects hidden in terminology.

A particularly important open design is a fixed-test learning curve. Choose one evaluation family set, keep it fixed, and increase only the training families. That experiment would separate the effect of additional training evidence from changes in the composition of the held-out population. It is listed as C7 in Chapter 18.

Interpretation boundary. “Generalisation” in this chapter means within-resource transformation transfer to held-out lexical families under the specified leakage controls. It does not mean contextual disambiguation, transfer to an independent dictionary, derivational coverage beyond the source resources, or downstream retrieval effectiveness.

12.5 Why the result matters to the programme

The formal compiler result in P7 and the empirical holdout result in P1 answer different halves of one question. P7 shows *how* a compiled decision can extend beyond enumerated paths. P1 shows that this extension is not vacuous: it recovers many withheld targets across 20 resources. The combination is the basis for calling the system lexicon-trained without calling it lexicon-bound.

Runtime Cost of the Compiled Policy

A learned normaliser can be scientifically interesting and still be unusable in an analyser hot path. **P5** (Galambos, 2026e) therefore treats runtime as a separate empirical object rather than inferring it from trie depth, dictionary size or asymptotic intuition. The study asks three questions: what execution cost is observed across the 20-resource census, how does that cost compare with Snowball under the frozen benchmark workload, and which structural quantities track the measured latency most closely?

13.1 Measured execution envelope

Under the canonical Java benchmark configuration, per-occurrence execution cost ranges from 42.8 to 150.6 ns across the 20 resources, with an unweighted resource median of 67.8 ns. These are benchmark measurements, not universal hardware constants. The workload is intentionally controlled so that the structural comparisons are interpretable.

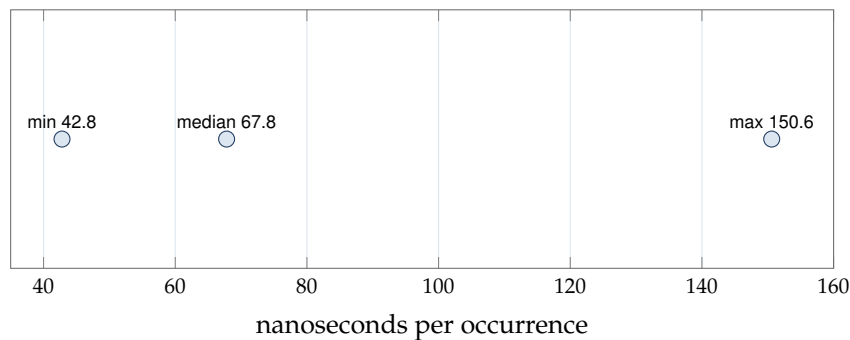


Figure 13.1: Published canonical execution range across the 20-resource benchmark census in P5. The points describe the frozen workload and should not be read as portable machine-independent timing constants.

13.2 Direct Snowball comparison

P5 includes 18 language comparisons for which a Snowball implementation is available under the study’s mapping. The median Radixor/Snowball throughput ratio is approximately $2.33\times$ under one reported policy and $2.57\times$ under the other. The comparison is useful because it demonstrates that a lexicon-trained compiled transformation engine need not pay an obvious online penalty for having learned from a large resource.

It does not establish a universal “fastest stemmer” claim. The two systems have different implementation histories, language coverage, policy semantics and memory layouts. Benchmark results are workload- and platform-dependent. The scientifically reusable result is the measured envelope and the fact that Radixor’s online work remains small despite potentially large training resources.

P5 also defines a robustness check across its two prespecified Radixor timing denominator policies. Seventeen of the 18 direct Snowball comparisons remain Radixor advantages under both policies; Finnish is the sole parity case, with overlapping score-error ranges around the two point estimates. This “17 stable advantages plus one parity” statement is a deterministic summary of one frozen campaign, not a significance test.

A separate resource-wide relation audit reports higher Radixor balanced accuracy in every matched direct-Snowball comparison against the declared resource relation. That result is deliberately not merged with the timing sample: the quality audit uses a different population and a reference related to Radixor’s training data. Its role is narrower—to show that the observed runtime advantage is not obtained by simply moving to a systematically weaker operating point under that particular intrinsic reference.

Table 13.1: Headline measurements from P5.

Quantity	Published observation
Canonical Radixor execution	42.8–150.6 ns/occurrence; median 67.8 ns across 20 resources
Direct Snowball comparisons	18 language comparisons
Median throughput ratio	about $2.33\times$ and $2.57\times$ under the two reported policy views
Spearman association: internal nodes vs runtime	$\rho = 0.931$
Spearman association: training rows vs runtime	$\rho = 0.723$

13.3 Training size and online structure are different quantities

The most informative structural result is not the raw speed ratio but the association pattern. Across the resource census, compiled internal-node count has Spearman $\rho = 0.931$ with runtime, whereas training-row count has $\rho = 0.723$. This is consistent with the architecture: training examples are an offline input, while runtime traverses the compiled representation.

The benchmark supports a representation-centred explanation of online cost: compiled structure is more closely associated with measured runtime than the number of lexical rows from which that structure was learned.

That statement must remain descriptive. Node count and training size are correlated, the resource census is small, and no intervention independently varies one while holding the other fixed. The study therefore does not identify a causal coefficient saying how many nanoseconds one additional node costs.

The resource ordering illustrates why compilation is the relevant intermediate layer. English has 396,939 training rows but compiles to 30,868 internal nodes and runs at about 80.8 ns/occurrence; German has only 54,092 training rows but compiles to 81,363 internal

nodes and runs at about 150.6 ns/occurrence. The compiler has substantially reordered lexical-resource scale before runtime is measured. P5 further reports that the internal-node association remains stronger than the training-row association under every one-resource deletion, so the contrast is not created by a single extreme language in the 20-resource census.

13.4 Runtime is not a proxy for lexical quality

P5 also examines a speed–reference-agreement envelope rather than assuming that smaller or faster models are automatically better. This is important for the programme’s logic. P2 can reduce structural vocabulary; P6 can remove unsupported complexity; the compiler can share equivalent subgraphs. None of those operations is licensed by runtime alone. Quality constraints remain separate.

The correct systems question is consequently multi-dimensional: can a policy be realised with acceptable relation error, compiled into a manageable structure, and executed within the analyser’s latency budget? The nine-study programme addresses those dimensions with separate experiments rather than collapsing them into one score.

Interpretation boundary. The benchmark sweeps a controlled type-oriented workload. Natural document streams have frequency skew, cache effects, tokenisation costs and pipeline interactions that are not represented by a uniform type sweep. P5 establishes runtime feasibility and descriptive structural association under its frozen workload; C5 in Chapter 18 asks whether the conclusions transfer to realistic token-frequency distributions.

13.5 Why runtime closes the measurement part

By the end of Part III the policy has been studied as a relation, as a multi-objective error point, as a transformer of held-out lexical families, and as an executable runtime object. The next question is therefore no longer whether the standard policy can be characterised. It is whether the policy can be *changed deliberately* while the mechanism remains fixed. That shift from measurement to intervention begins in Chapter 14.

Part IV

Adapting the Policy to Retrieval

Chapter 14

A Controlled Test of Policy Learnability

Calling the training relation a “policy” is meaningful only if changing that relation can change the realised conflation behaviour while the transformation mechanism remains fixed. **P4** (Galambos, 2026d) provides that controlled experiment. It uses two German CISTEM gold standards that differ in how aggressively they group forms, trains the same Radixor mechanism from each, and asks whether the output relation moves in the direction implied by the changed reference.

The experiment is intentionally modest. It does not establish that arbitrary policies can be learned in arbitrary languages. Its value is identificational: it holds architecture fixed and changes the target relation, which isolates a capability that default-vs-default stemmer comparisons cannot observe.

14.1 A nested policy contrast

The public CISTEM material supplies two reference partitions. GS1 is broader and more derivation-friendly; GS2 is more conservative and lemma-oriented. On the common unambiguous universe used by P4, GS2 strictly refines GS1. That nesting creates a directional set of pairwise “switches”: pairs joined under GS1 but separated under GS2.

Because the public files preserve class membership but not the original CELEX-derived target strings from which the classes were built, P4 treats the partition itself as the policy. Three deterministic class-to-target encodings are used as a nuisance-factor sensitivity check. The target string is thus a construction device; the estimand remains the induced relation.

14.2 The architecture is held fixed

The experiment uses deterministic five-fold component cross-fitting. Training and test units are separated at the connected-component level so that the pair relations being evaluated are not trivially reconstructed from the same component in training. The same Radixor learning and runtime mechanism is used for both policy conditions. The experimental contrast is therefore not “Radixor vs another stemmer” but “Radixor trained to realise policy A vs the same mechanism trained to realise nested policy B”.

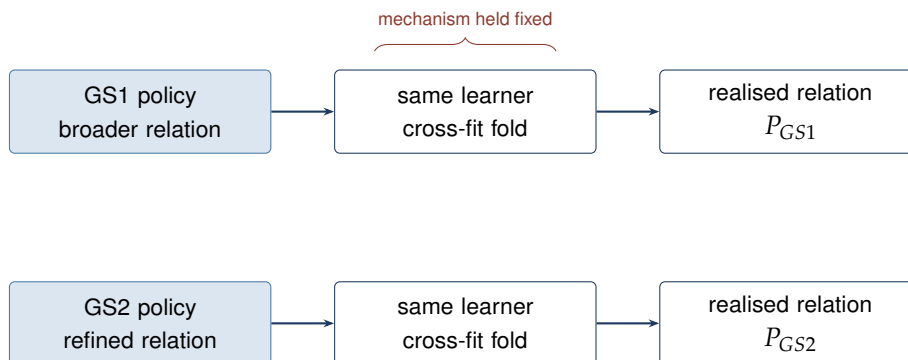


Figure 14.1: The causal contrast in P4 changes the declared training relation while keeping the learning and runtime mechanism fixed. The question is whether the realised relation moves with the policy.

14.3 Result: small, directional policy response

P4 reports a positive policy-aligned effect. The primary effect is small—the policy-switch region is sparse, on the order of 0.61% in the primary medoid view—but its direction is the important part. Training on the conservative policy makes the system more likely to separate pairs that the conservative reference separates; training on the broader policy makes it more likely to join the corresponding pairs.

Under a fixed transformation engine, changing the reference partition changes the learned relation in the corresponding direction. The experiment therefore supports *policy learnability*: part of what a stemmer does can reside in training data rather than being inseparable from the algorithm.

The magnitude should not be inflated rhetorically. The experiment contains one language and one nested policy axis, and the switch population is small. P4 is strong evidence for the existence of policy-conditioned behaviour, not a population estimate of how much any future domain policy will move a model.

14.4 Why the result is different from ordinary tuning

Hyperparameter tuning changes how one mechanism fits a fixed target. P4 changes the target itself. This distinction matters because two search systems can prefer different conflation objectives even when both are implemented by the same code. One application may prefer conservative lemma-like groupings; another may prefer derivation-friendly conflation to increase candidate recall. If the mechanism can learn both, “the stemmer” is no longer a single relation. It is a family of relations indexed by policy evidence.

This observation also reframes historical stemmer comparisons. A fixed Snowball implementation represents one carefully designed operating point. A lexicon-trained adaptable system exposes additional degrees of freedom. The fair question is not automatically whether its default model beats Snowball’s default. It may instead be whether each system is compared at the policy and information budget permitted by the deployment.

Interpretation boundary. P4 does not show that a policy learned from CISTEM transfers to retrieval, nor that the learned policy is preferable for users. It shows that a fixed mechanism responds to a changed lexical objective. Retrieval consequences are a separate experiment in Chapter 16.

14.5 From learned policy to deployable policy

P4 establishes that policy information can affect the learned relation, but it still rebuilds from policy-specific training data. Production adaptation asks for a stronger capability: start from a standard compiled model, express positive and negative policy changes, rebuild the executable structure, and check that the resulting memberships are exactly what the deployment intended over a declared vocabulary. That operational problem is the subject of P9 (Galambos, 2026i) and the next chapter.

Chapter 15

From Policy Edits to a Verified Deployment

A configurable stemmer is not automatically an auditable one. Adding an exception list may make named words behave as desired while leaving the model’s generalised rules free to pull other words into the same output class. The resulting policy is then known only at the edited examples, not over the vocabulary that will actually be indexed.

P9 (Galambos, 2026i) turns adaptation into a two-sided contract. Positive assignments state memberships that must be created or redirected. Negative boundaries state memberships that must not arise. The adapted model is then compiled and checked against an observed inventory. This chapter separates those three stages—expression, realisation and verification—because treating them as one “customisation” step hides the failure mode measured in the study.

15.1 A standard model is a substrate, not a ceiling

Let H_0 denote the standard compiled policy. An adaptation set contains positive assignments and protected non-memberships.

Definition 15.1 (Adaptation set). For a declared observed vocabulary V , an adaptation set is $\mathcal{A} = (A^+, A^-)$ where A^+ contains required surface-to-output assignments and A^- contains prohibited class memberships or equivalent negative boundary statements. Applying $\mathcal{A}$ produces a new compiled model $H_{\mathcal{A}}$ whose realised relation is subsequently checked on V .

The definition deliberately scopes exactness to V . No finite verification can prove the behaviour of every possible future string in Σ^* . A model that passes closure today may need re-verification when the deployment vocabulary grows materially.

15.2 Positive assignments are only half of the policy

A positive edit says that a named form should map to a chosen class. In a rule-generalising engine, that edit can also change the command selected for other forms sharing the relevant context. Some of those additional memberships may be desirable; others may violate the intended class boundary.

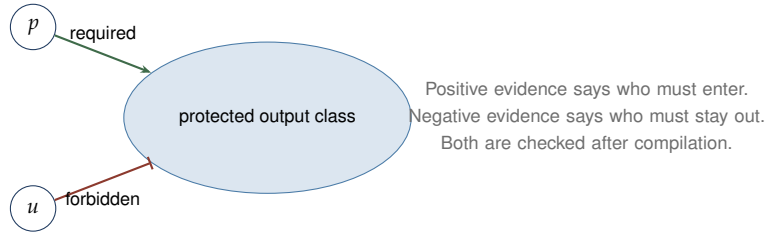


Figure 15.1: A deployable policy specifies both memberships that must be realised and boundaries that generalisation must not cross.

15.3 The measured reason closure is necessary

The English MIRACL development case shared by P8 and P9 provides a concrete failure of positive-only reasoning. After the intended positive assignments were installed, the unsealed realisation admitted 11,291 unintended surfaces to protected outputs. Those extra surfaces are not all equally important for retrieval, but their existence is enough to invalidate the claim that verifying only the edited words verifies the policy.

In the reported case, all explicitly edited words could be correct while the compiled relation was still wrong over the observed vocabulary. The measured 11,291 unintended surfaces make post-compilation closure a correctness requirement, not a defensive programming preference.

The sealing procedure adds negative barriers, recompiles, and repeats the audit until the protected-class membership reaches a fixed point over the checked inventory. Two fixed points must be distinguished: a lexical-policy fixed point, where no further protected membership violations are found, and a retrieval fixed point, where the downstream ranking objective would no longer benefit from another policy edit. P9 establishes the former; P8 analyses one selected development outcome related to the latter.

Algorithm 15.1: Two-sided adaptation with observed-vocabulary verification

Input: standard model H_0 ; positive assignments A^+ ; negative boundaries A^- ; observed inventory V

- 1 $H \leftarrow$ reopen H_0 as an editable policy substrate;
- 2 install all assignments in A^+ ;
- 3 install all currently known boundaries in A^- ;
- 4 **repeat**
- 5 compile H ;
- 6 execute H over every $w \in V$ and materialise output-class memberships;
- 7 identify unintended memberships of protected classes;
- 8 encode the required negative barriers and add them to H ;
- 9 **until** no protected membership changes;
- 10 verify all required positive assignments and all protected negative boundaries;
- 11 **return** compiled policy H_A plus verification record;

15.4 Representation, learning and deployment fidelity

P9 introduces a useful three-way distinction that clarifies several earlier claims in the book.

Representational capability asks whether the instruction language can express a desired transformation at all. Proposition 4.2 is a representational statement.

Learning capability asks whether the induction process selects that transformation from available evidence. P1, P2, P4 and P6 operate largely at this level.

Deployment fidelity asks whether the compiled model actually realises the intended classes over the inventory being deployed. The closure audit in P9 is a deployment-fidelity test.

A system can be expressive enough but fail to learn a relation; it can learn the named examples but compile a relation with unintended neighbours; or it can realise the intended relation and still fail to improve retrieval. Keeping these levels separate prevents one successful demonstration from being used as evidence for another.

15.5 Comparison with later configurable mechanisms

P9 also revisits the historical action language. XSTEM’s native terminal cut-and-append action is contained in the older Egothor patch executor, while a single patch can preserve source characters between separated edits (Baker, 2022; Galambos, 2001). CSTlemma and LemmaGen provide distinct later rule-learning mechanisms (Jongejan and Dalianis, 2009; Juršič et al., 2010). The correct conclusion is not that Radixor is the first system ever to support every kind of customisation. Earlier and later systems expose overlapping capabilities in different ways.

Table 15.1: Selected technical dimensions of later configurable mechanisms. No overall ranking is implied: tasks, information budgets and deployment contracts differ.

System	Terminal action	Separated edits	Learned?	Deployment distinction
Egothor/Radixor	cut + append included	yes	yes	P9 adds explicit positive/negative installation and vocabulary closure configurable examples; different verification contract contextual, lemmatisation-oriented rules lemmatisation objective; distinct rule formalism
XSTEM	cut + append	not as one native action	exemplar-derived	
CSTlemma	suffix-style actions	captured-span rewrites	yes	
LemmaGen	induced rewrite rules	rule-dependent	yes	

Interpretation boundary. The comparison concerns documented native transformation mechanisms and the explicit verification workflow studied here. It does not establish that other systems cannot be wrapped, extended or retrained to realise an equivalent finite-domain mapping. Equal-budget predictive and runtime comparisons remain empirical questions.

15.6 The bridge to retrieval

At this point the programme has an adapted policy whose compiled realisation is checked over a declared vocabulary. Only now is a downstream effectiveness comparison interpretable as evidence about that realised policy. This ordering is critical: without P9, P8's retrieval gain could be attributed to a set of named edits while the actual class memberships remained unknown. Together the two papers form one deployment case with two analytical responsibilities—P9 owns the realisation contract, and P8 owns the retrieval analysis.

Chapter 16

Where Retrieval Quality Goes

Intrinsic lexical evaluation tells us whether a realised relation agrees with a reference. Search users, however, encounter a ranked list produced by a retrieval model whose statistics are themselves changed by lexical conflation. **P8** (Galambos, 2026h) therefore asks the downstream question directly: when a Radixor policy is adapted for a retrieval objective and installed as the verified policy described in **P9** (Galambos, 2026i), what changes in the ranking, and how should that change be interpreted?

The study is intentionally a development-set case, not an out-of-sample victory claim. Its importance lies in demonstrating that policy is an operative search variable and that its effect is conditional on ranker parameters, metric weights and ranking depth.

16.1 Experimental setting

The final case uses the English MIRACL development split: 799 queries over approximately 32.9 million passages (Zhang et al., 2023). A standard Radixor model, PyStemmer, and the adapted/sealed Radixor policy are evaluated under a declared common BM25 operating point. **P8** also analyses a separate 164-evaluation BM25 response study. The two stages must not be conflated: the response surface characterises scorer/objective sensitivity, while the final adapted model is a selected development outcome.

The retrieval objective combines multiple metrics rather than declaring one cutoff universally sufficient. This matches the policy thesis: aggressive conflation can improve candidate recall at deeper ranks while changing lexical specificity near the top, so a stemmer can move quality between parts of the ranked list rather than simply raising or lowering one scalar.

16.2 Adaptation improves the standard model in the reported case

At the common scorer used for the final comparison, the sealed adapted policy improves the standard Radixor dictionary on all five recorded mean metrics. The reported relative gains include 3.667% in nDCG@10, 4.043% in Recall@100, and 5.340% in AP@1000. On the study's combined objective, the standard/default Radixor model is approximately 0.529014, PyStemmer approximately 0.545684, and the sealed adapted model approximately 0.550119.

Table 16.1: Selected headline results from P8 at the declared common scorer. They are development-set outcomes from one English MIRACL case, not held-out generalisation estimates.

Measure	Reported adapted-vs-standard change
nDCG@10	+3.667%
Recall@100	+4.043%
AP@1000	+5.340%
Combined objective J	standard ≈ 0.529014 , PyStemmer ≈ 0.545684 , sealed adapted ≈ 0.550119
Other recorded mean metrics	adapted model also higher at the common scorer

The default dictionary is not an upper bound on what the Radixor engine can realise for a retrieval task. In the studied development condition, an explicitly adapted and verified policy reverses the standard model's ordering relative to PyStemmer on the reported combined objective.

That result is stronger than saying that an exception list changed a few query terms: P9 verifies the realised class memberships over the observed inventory. It is weaker than saying that the adaptation procedure will generalise, because the same development evidence participates in selection and evaluation.

The full final table is useful because the gain is not confined to the three metrics used in the composite objective. At the common $k_1 = 1.125$, $b = 0.30$ scorer, the sealed policy has the largest observed mean on all five recorded metrics.

Table 16.2: P8 final means at the common scorer. Boldface marks the largest observed development mean; it is not a held-out superiority claim.

Method	nDCG@10	R@100	R@1000	AP@1000	MRR@10	J
No stemming	0.339398	0.763944	0.884677	0.284580	0.393572	0.503734
Standard Radixor	0.357492	0.800485	0.912621	0.300742	0.410104	0.529014
PyStemmer	0.365427	0.829154	0.920418	0.313090	0.419755	0.545684
Sealed Radixor	0.370600	0.832846	0.921878	0.316801	0.424842	0.550119

The deployment stages also matter. The standard model has $J \approx 0.529014$; the positive-only patch reaches approximately 0.545322; the sealed two-sided policy reaches 0.550119. Sealing therefore recovers roughly 0.004797 objective units after the leakage audit. This is a staged combined intervention, not a factorial estimate of independent positive- and negative-edit effects: a negative-only model was not evaluated.

16.3 The aggregate gain is concentrated on a minority of queries

Mean improvements can hide where support for the change actually lies. P8 therefore preserves paired query diagnostics for sealed Radixor minus PyStemmer. Most queries tie numerically; the positive means are generated by a minority of changed cases.

The Recall@1000 mean, for example, changes on only two queries, both in favour of the sealed model. The nDCG@10 difference is supported by 49 wins and nine losses while 741 queries tie. These counts guard against reading a small positive aggregate mean

Table 16.3: Descriptive P8 differences on the same 799 development queries used for policy selection. W/T/L is wins/ties/losses. The intervals and nominal values are descriptive, not selection-adjusted confirmatory inference.

Metric	Mean difference	W/T/L
nDCG@10	+0.005174	49 / 741 / 9
Recall@100	+0.003692	8 / 789 / 2
Recall@1000	+0.001460	2 / 797 / 0
AP@1000	+0.003711	90 / 685 / 24
MRR@10	+0.005086	28 / 764 / 7

as a universal per-query improvement, and they motivate the document-level transition analysis that the current archive cannot yet perform.

16.4 Vocabulary reduction and query work do not explain the gain

The final census separates policy effectiveness from simple notions of “stemming strength”. PyStemmer produces the smallest output vocabulary, whereas sealed Radixor retains 181,867 more output terms (about 3.35%) and still attains the higher in-sample objective. Mean deterministic query work is nearly unchanged between the two methods.

Table 16.4: P8 final structural summary. Posting and candidate counts are per-query means and are work counts, not latency measurements.

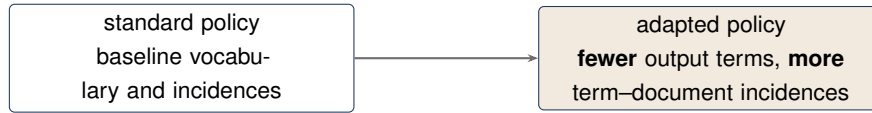
Method	Output vocabulary	Posting entries	Candidate docs
No stemming	6,046,958	41.160M	22.384M
Standard Radixor	5,782,800	41.747M	22.496M
PyStemmer	5,432,989	41.781M	22.514M
Sealed Radixor	5,614,856	41.762M	22.510M

Relative to PyStemmer, sealed Radixor evaluates about 0.045% fewer posting entries and touches about 0.017% fewer candidate documents on average. The measured effectiveness increase is therefore not explained by a materially larger deterministic candidate workload. Conversely, a smaller vocabulary is not sufficient to predict the better retrieval objective.

16.5 The representation change is not one-dimensional

One of P8’s most informative diagnostics concerns aggregate index structure. The adapted model produces a smaller output vocabulary than the standard model but a *larger* number of unique term–document incidences. Under a simple partition view, pure merging can only maintain or reduce the number of distinct term–document incidences: when two terms already occur in the same document, their incidences collapse; otherwise they are conserved. Pure splitting tends to increase vocabulary and can increase incidence counts.

Observing fewer output terms together with more incidences therefore excludes a simple total ordering in which the adapted policy is merely “more aggressive” or “less aggressive” everywhere. Some regions of the vocabulary are coarsened while others are effectively refined or redirected.



This combination is inconsistent with a pure global merge-only or split-only ordering.

Figure 16.1: The aggregate direction of two index statistics proves that the adapted mapping is not simply a globally coarser or globally finer version of the standard mapping under the stated scalar semantics.

This observation helps explain why quality can move differently at different rank depths. A policy change can simultaneously broaden some matches and restore specificity elsewhere. “Aggressiveness” is therefore at best a local descriptor, not a complete coordinate for an adapted conflation policy.

The preserved census makes the argument quantitative. Standard Radixor has 5,782,800 output terms and 1,501,142,516 full-corpus term–document incidences; the sealed policy has 5,614,856 terms but 1,502,796,144 incidences. It has therefore removed 167,944 output classes while adding 1,653,628 incidences. Under the shared scalar corpus semantics, pure coarsening cannot increase incidence and pure refinement cannot reduce the number of non-empty classes. The two realised partitions are consequently incomparable under simple global refinement.

A second counter also warns against using rewritten-token percentage as a surrogate for conflation strength. The fraction of token positions whose output string differs from the input is about 27.18% for PyStemmer, 14.75% for standard Radixor, and 74.69% for sealed Radixor. Relabelling can change that statistic without changing a class relation, so it cannot identify how much of the sealed intervention is merging, splitting, barrier insertion or mere output-label change.

16.6 The ranker optimum is endogenous to the policy

Stemming changes the term representation on which BM25 computes term frequency and document frequency. Even when document length in tokens is unchanged, the distributions entering term scoring change. It follows that the BM25 parameters preferred before a policy intervention need not remain preferred after it.

Write the retrieval objective as

$$Q(\pi, \theta; d, m),$$

where π is the lexical policy, $\theta = (k_1, b)$ the ranker configuration, d the query/document population and m the metric or metric combination. Optimising π at a fixed θ_0 and then discovering that the best θ has moved is not paradoxical: the policy intervention changed the statistics seen by the ranker.

The natural systems problem is consequently coupled:

$$(\pi^*, \theta^*) \in \arg \max_{\pi, \theta} Q(\pi, \theta; d, m).$$

A coordinate-descent interpretation is possible—tune policy under a ranker, retune the ranker under the new policy, and repeat—but P8 does not establish that such a procedure

converges globally. It establishes the empirical premise that makes the coupled formulation necessary.

16.7 The existing recall means locate where judged relevance moves

P8 derives an exact three-band accounting identity from the two reported recall cutoffs. With query means denoted by bars,

$$p_{1:100} = \overline{R@100}, \quad p_{101:1000} = \overline{R@1000} - \overline{R@100}, \quad p_{>1000} = 1 - \overline{R@1000}.$$

The arithmetic is exact for the reported means under a shared query set; the interpretation as shares of judged relevance additionally assumes non-empty relevant sets for all included queries. It is not a conservation law for nDCG, AP or utility.

Table 16.5: Macro-averaged shares in percent at the common BM25 point. The adaptive-primary row is the earlier Step-2 policy and is not the final sealed model.

Method	Ranks 1–100	101–1000	Beyond 1000
PyStemmer	82.915	9.126	7.958
Adaptive primary (Step 2)	80.838	11.088	8.074
Sealed Radixor	83.285	8.903	7.812

The earlier adaptive-primary policy has 2.077 percentage points less judged relevance in the first 100 than PyStemmer, but 1.962 points more at ranks 101–1000. Much of its Recall@100 deficit is therefore associated with relevant material remaining deeper in the run rather than disappearing by rank 1000. The final sealed policy reverses the aggregate first-band deficit: relative to PyStemmer it is +0.369 points in ranks 1–100, -0.223 in 101–1000 and -0.146 beyond 1000. These marginals do not reveal which passages moved; a true rank-transition matrix requires paired document-level runs.

16.8 The preferred fixed policy depends on the declared objective

The earlier Step-2 compromise also permits an exact sensitivity calculation without retraining. Let

$$J_\alpha = \alpha \text{nDCG@10} + (0.9 - \alpha) \text{R@100} + 0.1 \text{AP@1000}, \quad 0 \leq \alpha \leq 0.9.$$

For the fixed adaptive-primary and PyStemmer runs, the sign of the objective difference changes at approximately $\alpha = 0.8002$. In other words, that earlier policy is preferred only once the fixed objective assigns roughly 80% weight to nDCG, 10% to Recall@100 and 10% to AP. At the study's $\alpha = 0.5$, its recall deficit dominates.

The final sealed comparison is qualitatively different: its observed mean is higher than PyStemmer on all five reported metrics. Consequently, for these already-selected fixed runs, every non-negative weighted sum of the five reported means with at least one positive weight favours the sealed model. This is a robustness statement about the observed means, not evidence that a policy retrained under arbitrary weights would generalise or remain optimal. The declared J gain over PyStemmer decomposes approximately 58.3% from nDCG@10, 33.3% from Recall@100 and 8.4% from AP@1000.

16.9 Different depths answer different product questions

Top-ranked precision and deeper candidate recall serve different stages of a modern search pipeline. A top-10 metric approximates what may be presented directly to a user; Recall@100 or AP@1000 can matter more to a second-stage reranker or a retrieval-augmented generation pipeline. The appropriate conflation policy can therefore depend on where in the pipeline lexical normalisation is used.

This is the conceptual meaning of the title *Where Retrieval Quality Goes*. A policy intervention can redistribute useful evidence across the ranked list. Reporting only one cutoff can make that redistribution invisible and can turn a trade-off into an apparent contradiction across studies.

Interpretation boundary. P8 is an in-sample development experiment. It establishes a realisable policy intervention and a policy–ranker interaction on English MIRACL development data. It does not estimate held-out effectiveness of the adaptation procedure, multilingual transfer, or a universal optimum. P8 and P9 share one experimental case and must not be counted as independent replications.

16.10 The decisive next retrieval experiment

The strongest next test is matched-budget, out-of-sample and multilingual. Policy construction and BM25 selection should be performed exclusively on a training/development partition; the resulting policy and ranker should then be frozen before evaluation on held-out queries. Competing adaptive methods should receive the same supervision budget and the same opportunity for ranker tuning. The design is recorded as C2 in Chapter 18.

Until that experiment is run, the correct conclusion is methodological: a comparison between default stemmer dictionaries is not automatically a comparison between attainable family-level policies, and the effect of a policy cannot be separated from the ranker and objective under which it is measured.

Part V

What the Programme Establishes

Chapter 17

What the Nine Studies Establish Together

A collection of experiments becomes a research programme only when the individual claims constrain one another. The nine Radixor studies do not form a simple sequence in which each paper adds one feature. They form a dependency structure. Formal compiler semantics tells us what kind of generalisation is possible; the family-holdout study asks whether that possibility is useful; the relation framework tells us how to measure the realised grouping; the policy-transfer study shows that the grouping can move with the target; the adaptation study shows that the changed grouping must be verified after compilation; and the retrieval study asks whether the verified intervention matters downstream.

This chapter assembles that structure into a set of claims with explicit strength. It also separates the contributions that are system-independent from those whose empirical evidence currently comes primarily from one artefact family.

17.1 The programme as a chain of identifiability

The strongest version of the policy thesis would say that lexical conflation is a controllable system variable whose value can be learned, compiled, verified and jointly tuned with a ranker. Each verb in that sentence requires a different kind of evidence.

Learned. P1, P2 and P6 show that commands are induced from lexical evidence, that construction parameters change the induced symbolic structure, and that source-resource consistency changes the support topology from which commands emerge (Galambos, 2026a,b,f).

Compiled. P7 gives the semantic account of subtree canonicalisation and accepting-leaf contraction. It separates semantics-preserving sharing from deliberate extension of a command decision to previously unrepresented continuations (Galambos, 2026g).

Measured. P3 treats the output as a pairwise relation and keeps missed and false conflations as separate objectives before considering scalar costs (Galambos, 2026c). The method is independent of Radixor’s implementation.

Changed. P4 holds the mechanism fixed while changing a nested German reference policy and observes a directional change in the realised relation (Galambos, 2026d).

Verified. P9 shows that positive edits are not enough to identify the deployed relation: the unsealed English case contains 11,291 unintended memberships. Negative boundaries and inventory closure make deployment fidelity an explicit obligation (Galambos, 2026i).

Measured downstream. P8 reports an English MIRACL development case in which the sealed adapted policy improves the standard model across all five recorded means at

the declared scorer, while a separate response study exposes ranker/objective sensitivity (Galambos, 2026h).

The thesis is persuasive because no single study is asked to do all six jobs.

17.2 Three levels of claim

The monograph distinguishes three levels at which a conclusion can live.

Table 17.1: A claim is promoted only as far as the evidence allows. Generality of a definition is not confused with generality of an empirical result.

Level	Examples in this volume	Current evidence boundary
Formal/methodological	pairwise overlapping-reference evaluation; Pareto strata; support intervals; compiler semantic distinctions	mechanism-independent where definitions depend only on outputs or abstract trie observables
Artefact-family empirical	edit-cost collapse, family transfer, resource audit, runtime envelope, policy response	broad multilingual evidence, but principally within Radixor/Egothor implementation and resource conventions
Downstream/task empirical	verified adaptation and MIRACL retrieval response	one shared English development case; not held-out or multilingual downstream validation

This distinction answers the habilitation-style objection that motivated the second revision of the book. The field-level contribution cannot be supported merely by measuring one artefact more thoroughly. Instead, the text now treats the output-level methodology as the transferable contribution and Radixor as the principal demonstration. The missing field-wide comparator campaign is then visible as an empirical validation step rather than hidden inside a broad rhetorical claim.

17.3 The reusable methodological contribution

Two parts of the programme survive replacement of the implementation almost unchanged.

First, the relation framework of Chapters 10 and 11 requires only a declared reference relation and candidate outputs. It can evaluate Snowball, Stempel, KStem, Morfologik, Hunspell or a future neural normaliser once each is mapped to the same output-policy contract. Overlapping gold groups remain representable, pair-population imbalance remains visible, and Pareto depth remains explicitly relative to the candidate set.

Second, the reporting discipline is architectural rather than Radixor-specific. An adaptable normaliser should be identified by more than a package name. At a minimum the experimental record should identify:

lexical evidence (resource, version, filtering and audit state), **construction condition** (induction parameters and reduction mode), **policy intervention** (adaptation data and output semantics), and, for downstream IR, **retrieval condition** (corpus, query split, ranker, parameters, objective, cutoff depths and model-selection protocol).

Without those fields, two experiments nominally using the same stemmer can instantiate different treatments.

17.4 A coupled view of lexical normalisation and ranking

The downstream results motivate a more explicit system model. Let M denote the transformation engine, π the lexical policy, θ the ranker configuration, d the data population and m the evaluation objective. The observable retrieval quantity is

$$Q(M, \pi, \theta, d, m).$$

A fixed-stemmer literature often treats M as the only meaningful treatment and the other arguments as context. The Radixor programme shows why that factorisation can fail for adaptable systems. Changing π changes the index statistics on which θ operates; changing θ changes which policy edits are rewarded by m .

This does not imply that every search system needs expensive joint optimisation. It implies that separability is an empirical assumption rather than a default truth. If the policy is learned under one scorer and then shown to prefer a different scorer after adaptation, the correct conclusion is that the two components interacted—not that one tuning stage was necessarily “wrong”.

17.5 What the programme does not yet establish

Four boundaries remain load-bearing.

Comparator breadth. The intrinsic framework has not yet been run as a single matched campaign across the principal external stemmer families. The book can show that the framework is implementation-independent by definition; it cannot yet show how much new field-level structure it reveals in practice.

Independent lexical evidence. Most multilingual construction and holdout experiments use UniMorph-derived resources. Family holdout prevents trivial row leakage but does not create an independent annotation tradition.

Held-out downstream effectiveness. P8 and P9 share a development case. Policy selection, sealing and evaluation are not an independent test of the adaptation procedure’s generalisation.

Matched adaptation budgets. Comparing an adapted Radixor policy with a fixed competitor’s default configuration answers a deployment question, but not a method-capability question if the competitor could also be adapted under comparable supervision.

These are not reasons to weaken the formal or measured results already obtained. They are reasons to state exactly which next experiments would promote the claims.

17.6 The revised thesis statement

After the nine studies are combined, the defensible central thesis is:

Lexical conflation in an adaptable retrieval analyser is best treated as an explicit, reproducible policy condition. That policy can be represented by a reference relation, learned into executable transformations, altered under a fixed mechanism, checked after compilation, and evaluated separately from the ranker with which it ultimately interacts. A stemmer package name alone is therefore insufficient to identify the treatment whenever these policy degrees of freedom are exposed.

The statement is deliberately about experimental method rather than about a universal winner. It is stronger than “stemming parameters matter” because it specifies the object that should be reported and verified; it is narrower than “Radixor is the best stemmer” because the present evidence does not license that global conclusion.

17.7 What would promote the thesis from framework to field result

The next threshold is not another internal Radixor ablation. It is external application of the framework. A same-resource Stempel comparison would isolate what has been added since the 2001 line. Adding Snowball, KStem, Morfologik and other same-task systems under frozen adapters would test whether the Pareto strata and policy geometry illuminate the field rather than only one system. Held-out multilingual retrieval would then test whether policy/ranker interaction generalises beyond the English development case.

Those experiments define the research agenda of the next chapter. Their role is not to repair an otherwise invalid monograph; it is to determine how far a well-supported artefact-level programme can be promoted into a field-level empirical conclusion.

Chapter 18

External Validation Programme

The most useful way to conclude a mature experimental programme is not to list “threats” as though they were generic disclaimers. It is to convert each remaining uncertainty into a test that would change the scientific status of a claim. This chapter therefore treats external validity as a programme of experiments with defined evidential roles.

The ordering follows the logic of Chapter 17. First test whether the ostensibly system-independent framework produces informative comparisons outside Radixor. Then break the two most important dependence structures—same-resource lexical evidence and in-sample retrieval selection. Finally strengthen the reference floor, inference and workload realism.

18.1 C4: turn the framework outward

The highest-value missing experiment is a field-level comparator campaign under the relation and Pareto framework of Part III. The starting point should be STEMPER, because it is the direct Egothor-derived ancestor documented in Lucene ([Apache Software Foundation, 2024](#)). With an identical lexical resource where possible, the comparison asks what the modern representation, compiler and training machinery add beyond the inherited mechanism.

The second ring should contain same-task systems with different construction premises: Snowball, KStem, Hunspell, Morfologik and a transparent truncation floor. Each adapter should be frozen before the final comparison and should state whether the system returns one value, several candidates, or an identity fallback. Morfessor may be included only through an explicit adapter that turns segmentation into a one-output conflation decision; otherwise it is a neighbouring morphological method rather than a directly commensurate stemmer.

The output should be a stratification resource, not an overall ranking. Pareto strata, error regimes, support intervals and candidate-set sensitivity are precisely valuable because they avoid inventing a universal exchange rate between false splits and false merges.

18.2 C1: independent lexical evidence

The multilingual studies gain breadth from 20 resources, but most of those resources are derived from the same UniMorph framework. Family-level holdout prevents direct training-family leakage; it does not make the gold relation independent of the source

Table 18.1: The comparator set should answer different questions rather than produce a single league table.

Comparator	Why it belongs	Question resolved
Stempel	direct Egothor-derived ancestor	what has the modern programme added to the same broad mechanism?
Snowball	strong fixed rule-based baseline	where does a hand-designed universal policy sit in the two-error space?
KStem	dictionary-gated English normalisation	how does conservative lexical gating compare with learned transformation?
Morfologik/Stempelator line	compact dictionary/FSA-oriented Polish line	what are the quality/runtime/memory trade-offs under aligned resources?
Hunspell	affix+dictionary ecosystem	how does a widely deployed lexical-rule architecture behave under the same relation contract?
Truncation	intentionally simple floor	how much structure is gained beyond a non-linguistic reduction baseline?

representation. C1 should therefore train from one lexical source and evaluate against another independently curated source after an explicit reconciliation of vocabulary and target semantics.

This experiment would answer a different question from P1. P1 shows within-resource transfer beyond enumerated families. C1 would test whether the learned transformations and relation conclusions survive a change of annotation source. A lower score would not invalidate P1; it would quantify source specificity that P1 intentionally does not estimate.

18.3 C2: held-out multilingual retrieval

The MIRACL case demonstrates a realisable policy intervention but reuses development evidence in the selection loop. C2 should split policy/ranker selection from final evaluation. For every language included, adaptation data, negative sealing, objective weights and BM25 parameters must be frozen before held-out queries are scored.

A particularly informative design would compare four conditions: fixed default policy/fixed ranker; adapted policy/fixed ranker; fixed policy/tuned ranker; and adapted policy/tuned ranker. That factorial structure would separate the incremental contributions of lexical policy and scorer calibration while retaining their interaction.

C2 is the experiment that can promote “policy/ranker interaction is observable in one development case” to “an adaptation procedure generalises downstream”. No amount of additional in-sample optimisation can substitute for it.

18.4 C3: establish an absolute intrinsic floor

Pareto depth is candidate-set relative. Removing a point can move another point to a better stratum without changing its outputs. That property is not a flaw; it is the definition of dominance depth. It does mean that a field resource benefits from a deliberately weak reference floor whose role is to anchor the geometry.

A truncation family is attractive for this purpose because its behaviour is simple, deterministic and controllable by retained length. The objective is not to make truncation competitive but to make the lower envelope of the candidate set explicit and reproducible across future additions.

18.5 C5 and C6: workload and inference

P5's microbenchmark controls the executed inputs to isolate model cost. A production analyser sees Zipf-like token frequencies, repeated hot terms, cache effects and surrounding tokenisation overhead. C5 should replay natural frequency-weighted streams and report throughput, latency distribution, memory footprint and warm/cold behaviour. The controlled type sweep should remain in the report as a diagnostic rather than be replaced.

The empirical chapters also use resources, families, seeds or queries as clusters rather than pretending that all pairwise observations are independent. C6 should extend that discipline with uncertainty aligned to the actual experimental unit: resource-level bootstrap or hierarchical summaries for multilingual campaigns, family-level resampling where appropriate, and query-level paired inference for retrieval. Pair counts are combinatorial outcomes, not independent Bernoulli trials.

18.6 C7: fixed-test learning curves

P1's nested coverage conditions change both the training set and the held-out population. C7 should freeze one test family set and increase only training coverage. The resulting curve would have a cleaner learning interpretation and would show whether command-inventory saturation precedes, tracks or lags target recovery.

18.7 How the nine studies should be counted

The nine manuscripts are provenance units, not nine statistically independent confirmations of one hypothesis. P8 and P9 explicitly share one English MIRACL development case; P2 and P6 interrogate different upstream determinants of the same construction pipeline; P1 and P7 form empirical and formal halves of the generalisation question. Appendix D therefore records which claims and data each study owns instead of treating publication count as scientific weight.

This point also strengthens the book's narrative. The programme is coherent because the studies answer different dependencies in one argument, not because there happen to be nine of them.

Table 18.2: Experiments ordered by how strongly they would change the interpretation of the current monograph.

Id	Primary uncertainty	What a completed experiment would add
C4	field-level comparator evidence	demonstrates the output-level framework across independent stemmer families; quantifies the increment over Stempel
C1	same-resource dependence	tests lexical transfer across independently curated sources
C2	in-sample downstream selection	estimates held-out, multilingual adaptation and policy–ranker interaction
C3	candidate-set relativity	supplies a stable intrinsic reference floor
C6	descriptive uncertainty	aligns inference with resources, families and queries
C5	benchmark workload realism	tests timing conclusions under natural token frequencies and memory conditions
C7	changing holdout population	yields a fixed-test learning curve and saturation analysis

18.8 Prioritised programme

The priority order is scientific, not logistical. C4 comes first because it addresses the central identity question raised by a field-level monograph: does the framework reveal something about conflation systems generally, or only about Radixor? C1 and C2 then address the two strongest external-validity limits. The remaining experiments sharpen calibration and inference.

18.9 The programme is open by design

A final monograph need not pretend that no important experiment remains. The criterion is whether the present claims are commensurate with present evidence and whether the next steps are specified strongly enough to falsify or promote them. On that criterion, the research line is complete enough to state its methodological thesis and open enough to be tested outside the artefact that motivated it.

Conclusions

The conventional question “which stemmer is better?” is too coarse for an adaptable normalisation system. It suppresses the lexical relation being implemented, the construction choices that produce the runtime model, the way the compiled structure generalises, and the scorer under which the relation is finally rewarded. This monograph replaces that question with a sequence of identified objects and experiments.

19.1 A policy can be made explicit

The first result is conceptual but operational. A lexical conflation policy can be represented as a declared relation over observed forms rather than as an implicit property of a program name. Once the relation is explicit, missed conflations and false conflations become separate measurable errors and alternative policies can be compared without requiring identical literal stem strings.

That representation also accommodates overlap. A form can legitimately participate in more than one lexical group, so the gold object need not be a partition. Chapters 10 and 11 therefore operate on pair relations and make candidate-set dependence and scalar cost assumptions visible.

19.2 A policy can be learned into an executable object

The Egothor/Radixor line learns patch commands from lexical relations rather than storing only answers. P2 shows that edit costs are structural hyperparameters, yet much of the sampled numeric space collapses to exact behavioural equivalence. P6 shows that lexical-resource consistency changes the support topology from which command vocabularies are learned. P7 then separates compiler operations that preserve the represented command-selection semantics from accepting-leaf contraction, which deliberately extends a uniform command decision to unrepresented continuations.

The empirical counterpart is P1: with whole lexical families held out and surface leakage controlled, transformations learned from the remaining families recover many unseen changed forms across 20 UniMorph-derived resources. The combination of formal semantics and holdout evidence is what supports the phrase *lexicon-trained but not lexicon-bound*.

19.3 A learned policy can be changed and checked

P4 supplies the controlled policy intervention. Under a fixed mechanism, a nested change in German reference policy produces a small but directional change in the realised relation. That experiment establishes policy learnability without relying on retrieval.

P9 makes the stronger deployment distinction. Positive assignments say what must join a class; negative boundaries say what must not join it; post-compile closure checks what actually happened. The 11,291 unintended surfaces in the unsealed English case demonstrate why a configurable system is not necessarily a verified system.

This leads to a useful three-level vocabulary for future work: *representational capability*, *learning capability*, and *deployment fidelity*. Claims at one level should not be promoted to the next without evidence.

19.4 A verified policy can change retrieval

P8 closes the chain with a full-corpus English MIRACL development experiment. At the declared common BM25 point, the sealed adapted policy improves the standard Radixor model on all five recorded mean metrics; the reported relative gains include 3.667% nDCG@10, 4.043% Recall@100 and 5.340% AP@1000. The adapted representation simultaneously has fewer output terms and more term–document incidences, proving that it is not a simple global move toward more or less conflation.

The accompanying BM25 response study shows why the ranker cannot be treated as an inert background component. Lexical policy changes term statistics; the ranker’s preferred parameters can therefore change with the policy. The appropriate system-level object is a coupled condition (π, θ) rather than a stemmer label with an assumed default scorer.

Interpretation boundary. The MIRACL result is a selected development outcome shared by P8 and P9. It establishes feasibility and interaction, not held-out multilingual effectiveness. The latter is the purpose of experiment C2.

19.5 What is transferable beyond Radixor

The monograph’s most durable contribution is not a claim that Radixor wins every comparison. It is an experimental grammar for conflation systems.

When a normaliser exposes adaptable lexical behaviour, the reproducible experimental unit is the **policy condition**, not the package name alone. Intrinsic evaluation should preserve false-split and false-merge objectives; deployment should verify the compiled relation over its declared inventory; and downstream evaluation should report the ranker, parameters, metric weights, rank depths and model-selection protocol together with the lexical policy.

The pairwise relation framework and Pareto stratification can be applied to external systems without knowing how they were implemented. The reporting standard likewise

applies to any system whose dictionary, exception rules, training data or deployment overrides change the realised relation.

19.6 What remains to earn a field-level empirical claim

The next decisive evidence is external rather than another internal Radixor ablation. A same-resource Stempel comparison can quantify what the modern compiler and experimental machinery add to the documented Egothor lineage. Extending the same framework across Snowball, KStem, Morfologik, Hunspell and a simple truncation floor can test whether the proposed geometry describes the field rather than one artefact family. Independent lexical resources and held-out multilingual retrieval can then test how far the observed effects transfer.

That agenda is a strength of the final formulation because it makes promotion criteria explicit. Formal claims already proved need not wait for another benchmark. Multilingual artefact-level findings remain valid within their resource and protocol boundaries. Field-wide comparative and downstream claims are reserved for experiments that actually sample the field and test held-out deployment conditions.

19.7 Final statement

Twenty-five years after the original Egothor publication, the research line has moved from a semi-automatic Java stemmer to a broader proposition about how lexical normalisation should be studied. The inherited patch-command mechanism provides an unusual long-lived experimental substrate; the nine modern studies make its construction, measurement, adaptation and retrieval effects auditable.

The resulting conclusion is deliberately procedural rather than promotional:

A stemmer is not fully identified by its algorithm name when its lexical conflation relation can change. The policy, its compiled realisation, and the retrieval condition under which it is evaluated are part of the scientific object.

That formulation is strong enough to be tested outside Radixor. Its value will ultimately be determined by whether future comparisons adopt that experimental unit and whether the external-validation programme confirms the same structure across independent systems, resources and retrieval tasks.

Part VI

Appendices

Appendix A

Notation, with Cross-References

The table repeats the front-matter notation with a pointer to the definition or result in which each symbol is introduced or first used substantively.

Symbol	Meaning	Introduced
Σ, Σ^*	alphabet; strings over it	Def. 5.1
w, u, v	surface forms	Def. 5.1
$\overleftarrow{w}$	reversal of w ; trie indexing key	Def. 5.5
$\mathcal{L}$	lexical training relation	Def. 5.1
$V(r)$	variant set of root r on its row	Def. 5.1
$G(w)$	gold groups containing w	Def. 10.1
Π	primitive edit operations	Def. 5.2
$c, \mathcal{C}$	patch command; command vocabulary	Def. 5.2
$\text{Exec}(c, w)$	command execution	Def. 5.3, Alg. 5.1
γ	edit-cost vector	Alg. 6.2, Ch. 7
$\mathcal{T}, \mathcal{N}(\mathcal{T})$	patch-command trie; its nodes	Def. 5.5
$\text{val}(n)$	value carried at node n	Def. 5.5
$\text{dom}(n)$	inputs whose traversal reaches n	Thm. 9.9
$\text{sig}_\mu(n)$	mode- μ reduction signature	Def. 9.1
$\equiv_\mu$	canonicalization congruence	Def. 9.2
$C(w)$	command selection function	Def. 5.6
$H(w)$	realised stemming function	Def. 5.7
μ	reduction mode	Ch. 5, §5.6
π	lexical policy	Def. 2.2
$\mathcal{A} = (P, N)$	adaptation set: positives, negatives	Def. 15.1
TP, FP, FN, TN	pairwise confusion counts	Def. 10.2
UI, OI	understemming, overstemming indices	Def. 10.2
BA	balanced accuracy; navigation metric	Tab. 10.1
C_ρ	raw pair cost $\rho FN + FP$	Def. 11.4
ρ	caller's exchange rate	Def. 11.4
I_a	support interval of candidate a	Prop. 11.5
$\text{nDCG}@k$	normalised discounted cumulative gain	Tab. 16.1
$R@k, AP@k$	recall and average precision at depth k	Tab. 16.1

Appendix B

Metric Definitions and Applicability

This appendix expands Table 10.1 with the sensitivity of each metric and its applicability under the three output policies of Section 10.6.

B.1 Confusion-derived metrics

All metrics are computed from unrounded raw integer counts. An undefined denominator yields n/a, never zero, NaN or infinity, and n/a values are excluded from macro averages rather than imputed.

Metric	Definition	Primary sensitivity
Under-stemming rate	$FN/(TP + FN)$	false splits within gold groups; equals $1 - \text{recall}$. Paice's UI.
Over-stemming rate	$FP/(TN + FP)$	false merges across groups. The denominator is very large, so values are small and small absolute differences are meaningful. Paice's OI.
Precision	$TP/(TP + FP)$	over-conflation. Degrades sharply when a common suffix is over-generalised.
Recall	$TP/(TP + FN)$	under-conflation. Bounded above by the output policy: a primary-output model cannot recover pairs requiring a discarded candidate.
Specificity	$TN/(TN + FP)$	false conflations relative to the very large negative population.

Metric	Definition	Primary sensitivity
Balanced accuracy	$\frac{1}{2}(\text{recall} + \text{specificity})$	both error types with equal weight on their own populations. The primary navigation metric ; ranking uses its unrounded value.
Pairwise accuracy	$(TP + TN) / (TP + TN + FP + FN)$	dominated by TN under class imbalance. Reported for completeness; not used for ranking.
F_1	$2TP / (2TP + FP + FN)$	positive class only; ignores TN .
$F_{0.5}, F_2$	$\frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2FN + FP}$	$\beta < 1$ weights precision, $\beta > 1$ weights recall. Reporting all three exposes the exchange rate that a single F score hides.
Jaccard	$TP / (TP + FP + FN)$	positive-class overlap; excludes TN entirely.
Fowlkes–Mallows	$\sqrt{\text{precision} \cdot \text{recall}}$	geometric mean of the two positive-class rates (Fowlkes and Mallows, 1983).
MCC	$\frac{TP\,TN - FP\,FN}{\sqrt{S_1S_2S_3S_4}}$, with $S_1=TP+FP, S_2=TP+FN,$ $S_3=TN+FP, S_4=TN+FN$	all four cells; informative under strong imbalance (Matthews, 1975). Undefined when any factor is zero.

B.2 Applicability by output policy

Table B.2: Which metrics are defined under each output policy. ANY_CANDIDATE yields no single confusion matrix (Proposition 10.3), so every confusion-derived score is n/a under it, while its separate error counts and denominators remain defined.

Quantity	PRIMARY	ALL_CAND.	ANY_CAND.
Under-error count and denominator	yes	yes	yes
Over-error count and denominator	yes	yes	yes
UI, OI	yes	yes	yes
TP, FP, FN, TN	yes	yes	n/a
Precision, recall, specificity	yes	yes	n/a
Balanced accuracy, F_β , Jaccard, FM, MCC	yes	yes	n/a
Used for headline ranking	yes	no	no

B.3 Metrics deliberately not computed

The adjusted Rand index (Hubert and Arabie, 1985), homogeneity, completeness, V-measure and normalised mutual information require a contingency table in which each item carries exactly one gold label. Under the overlapping gold relation of Definition 10.1 no such table exists. An overlapping-clustering analogue would need a separately specified random model and would not be comparable to the standard indices, so reporting one under a familiar name would mislead. These indices are therefore absent by decision.

B.4 Aggregation rules

Macro metrics average defined per-language values, giving each language equal weight regardless of resource size. **Micro** metrics sum the raw counts across languages and then compute the metric, which weights languages by pair volume. Both are reported where both are meaningful, because they answer different questions and can rank systems differently.

Cross-stemmer aggregates are computed only over the exact common supported-language intersection. Unsupported language and adapter combinations are absent from the aggregate rather than scored zero, and an aggregate over a different intersection is a different quantity that must not be compared with it.

Appendix C

Protocol Identities and Reproducibility

C.1 Provenance discipline

Every campaign in this volume is identified by a frozen protocol string, and every artefact it produces carries an identity that can be checked independently. The elements are the following.

- **Protocol identifier.** A version string naming the procedure, for example `radixor-generalization-v1`. The identifier is part of the hash input for row ranking, so a protocol change produces a different split rather than a silently different result under the same name.
- **Model identity.** Compiled model files are identified by SHA-256. Independent regeneration is expected to reproduce the identity exactly, and Chapter 14 reports a clean replay that regenerated all 30 persisted cross-fit models with matching identities.
- **Resource identity.** Compressed lexical resources carry SHA-256 digests, so a claim about a language is attached to a specific resource state rather than to a resource name.
- **Declared seeds.** Seeds are declared before results are inspected and are published with the protocol.
- **Raw counts.** Numerators and denominators are stored; percentages are derived at publication time. An undefined denominator is recorded as `n/a`.

C.2 Declared seeds

The five seeds of `radixor-generalization-v1`:

```
2654435761 2611923443488327891 7046029254386353131
11400714819323198485 15111065706836454659
```

C.3 Generalisation campaign scale

Smallest and largest resources in the primary study bracket the reliability of the reported ranges: Persian contributes 69 rows and Yiddish 802, which is why the five-split median and range are reported rather than a single figure.

Table C.1: Scale of the frozen generalisation campaign and of the extended sweep published alongside it.

Quantity	Count
Default model resources in the primary study	20
Coverage levels (10% to 100%)	10
Declared seeds	5
Build/evaluation scenarios (primary)	1,000
of which with non-empty held-out population	900
Additional scenarios in the extended sweep	6,150
Rows in the merged snapshot	7,150
Disjoint model sets covered by the merged snapshot	20 and 123
Extended matrix (models $\times$ coverage $\times$ seeds)	$143 \times 10 \times 5$

C.4 Reproduction entry points

The published harness exposes the campaign through named build tasks. The sequence below reproduces the generalisation snapshot and verifies the published documentation against it.

Task	Effect
<code>dictionaryGeneralizationStandalone</code>	run one resource's build/evaluation scenarios and emit raw counts
<code>mergeDictionaryGeneralizationSnapshots</code>	merge per-resource outputs into the campaign snapshot
<code>publishDictionaryGeneralizationDocumentation</code>	render the published tables from the snapshot
<code>verifyDictionaryGeneralizationDocumentation</code>	check that the published tables match the snapshot exactly

The last task is the one that matters for trust. Rendering documentation from data is ordinary; *verifying* that published documentation still matches its data catches the failure mode in which a table is edited by hand, or a regenerated campaign quietly diverges from the text describing it.

C.5 Edit-cost campaign

Quantity	Count
Sampled grid points	234
Physical model builds	16,700
Logical observations	234,000
Distinct induced behaviours	334

Behavioural equivalence is decided by exact comparison of normalised command sequences (Proposition 7.1); no tolerance is involved, and the reduction is therefore a deterministic fact about the campaign rather than a clustering result.

C.6 Quality campaign

The published quality appendix retains 66 directly compared language-mode matrices across 33 unique languages, 216 RADIXOR-only matrices and 4 undefined matrices, and maps 190 deterministic implementation rows to 174 aggregate error vectors. Undefined matrices are retained explicitly rather than dropped, on the same principle that records n/a rather than zero.

C.7 Where the material lives

Protocol definitions, raw result tables, per-language appendices and reproduction instructions are published with the project documentation ([Galambos, 2026j](#)). This volume restates results; it does not replace the frozen artefacts, and where the two differ the artefacts are authoritative.

Appendix D

Evidence Ledger: The Nine Constituent Studies

This appendix records the empirical and formal provenance of the monograph. The nine studies are not chapters in disguise, nor are they nine replications of a single experiment. They isolate different links in one argument: how a lexical conflation policy is induced, compiled, measured, modified, verified, and finally exposed to retrieval evaluation. The order below is the series order; the book deliberately follows logical dependency rather than publication sequence.

The persistent identifiers identify the manuscripts synthesised here. Availability as a preprint is not represented as journal or conference peer-review status. Numerical statements below are reproduced from the respective study; no new experimental observation is introduced by this appendix.

P1 — Generalisation beyond enumerated lexical families

(Galambos, 2026a)

Does a lexicon-trained stemmer generalize? A 20-language study of held-out lexical families

Evidence unit. Twenty UniMorph-derived resources, ten training-coverage levels and five pre-declared seeds: 1,000 build/evaluation scenarios, 900 with a non-empty held-out population after parser-normalised surface-overlap control.

Principal result. Exact transformation accuracy on unseen changed forms rises systematically with training coverage. Across the 20 resources, the median increases from 63.72% at 10% coverage to 70.44% at 90%, and every resource has the higher median at 90%.

Boundary. This is within-resource, out-of-training-family transfer. It does not establish transfer to an independently sourced lexicon, domain, or language.

Used in this volume. Chapter 12.

P2 — Alignment costs as structural hyperparameters

(Galambos, 2026b)

Edit costs as structural hyperparameters in dictionary-induced stemming: A multilingual sensitivity study

Evidence unit. A sampled 234-point edit-cost grid over 20 resources: 4,680 resource–cost combinations, 16,700 physical build/evaluation scenarios and 234,000 logical observations under the study’s evaluation protocol.

Principal result. The grid collapses to 334 exact command-sequence behaviours. Much of the quality landscape is flat, but the induced structure is not: some settings reduce compiled structure by more than 24%, showing that edit costs are model hyperparameters rather than innocuous implementation constants.

Boundary. The grid is deliberately sampled, not exhaustive over all non-negative cost vectors; observed plateaus therefore characterise the tested region only.

Used in this volume. Chapter 7.

P3 — Multi-objective comparison and Pareto strata

(Galambos, 2026c)

Classifying multilingual stemmers by Pareto strata: Error regimes in pairwise lexical conflation

Evidence unit. Pairwise relation evaluation over overlapping lexical gold groups, with 190 implementation rows mapping to 174 aggregate error vectors; recursive Pareto sorting and exact support intervals for the positive linear cost family $C_p = \rho FN + FP$.

Principal result. The observed vectors occupy four recursive Pareto strata (40, 97, 27 and 10 vectors). In the primary view Radixor is the sole first-stratum vector in all 19 comparative languages and strictly dominates all 75 comparator rows on both pair-error axes; the second stratum further separates into recall, specificity, supported-compromise and unsupported-efficient regimes.

Boundary. “Unsupported” is a property relative to that scalar family; it is not a claim that the point is irrational or useless under every possible utility.

Used in this volume. Chapter 11.

P4 — Policy learnability under a fixed mechanism

(Galambos, 2026d)

The stemming policy is data

Evidence unit. A controlled German experiment built from two nested CISTEM-derived reference policies while retaining the same Radixor induction and execution mechanism.

Principal result. Cross-fit models show sparse but directionally stable policy-conditioned changes along the tested aggressive-to-conservative axis; the primary medoid encoding yields approximately 0.61% pair-micro switch coverage. Independent clean replay regenerates all 30 persisted models with matching SHA-256 identities.

Boundary. This is one language and one nested policy axis. It establishes learnability of a controlled policy difference, not universal policy learnability.

Used in this volume. Chapter 14.

P5 — Compiled structure and online cost

(Galambos, 2026e)

Runtime cost of a compiled lexicon-trained stemmer

Evidence unit. Twenty compiled Radixor models, with repeated occurrence-level measurements and 18 directly available Snowball comparisons under two comparison policies.

Principal result. Observed Radixor cost spans 42.8 ns to 150.6 ns per occurrence, with a median of 67.8 ns. Internal-node count correlates more strongly with runtime (Spearman $\rho = 0.931$) than raw training rows ($\rho = 0.723$); 17 of 18 Snowball comparisons favour Radixor and one is approximately at parity.

Boundary. The experiment characterises the tested JVM, models and harness. Correlation with structure is explanatory evidence, not a causal decomposition of every runtime component.

Used in this volume. Chapter 13.

P6 — Resource consistency and transformation vocabulary

(Galambos, 2026f)

Lexical-resource consistency as a compiler input

Evidence unit. A multilingual audit of high-command configurations with a homogeneous grouping rule applied only where it is non-destructive under the study’s criterion.

Principal result. Twelve high-command resources yield ten non-destructive candidates and two explicit no-safe-change outcomes. Eight of nine homogeneous grouped-rule candidates reduce command vocabulary, from a summed 20,281 to 19,842 (−2.165%, median −1.334%); German is separate at 6,986 to 5,313 (−23.948%). Finnish, Hebrew and Polish further show that row edits, command counts and package bytes are not interchangeable.

Boundary. The result concerns consistency transformations supported by the audited resources. It is not evidence that grouping is always beneficial or safe.

Used in this volume. Chapter 8.

P7 — Compiler semantics

(Galambos, 2026g)

Compiler semantics for trie-based lexical transformation

Evidence unit. Formal definitions of the selection function, semantic subtree signatures, canonicalisation and uniform accepting-leaf contraction for the Egothor/Radixor trie compiler.

Principal result. Mode-specific semantic canonicalisation preserves the represented selection function and yields a relatively minimal quotient of the exact contraction-free trie under the stated equivalence. Uniform accepting-leaf contraction instead extends a value decision to previously unrepresented inputs and is therefore a generalisation operation, not semantics-preserving minimisation.

Boundary. The minimality statement is relative to the represented trie and equivalence before generalising contraction; it is not a global minimum over arbitrary program representations.

Used in this volume. Chapter 9.

P8 — Retrieval depth, scorer interaction and adapted policy

(Galambos, 2026h)

Where retrieval quality goes: Rethinking stemming in modern search

Evidence unit. The English MIRACL development set: 799 queries and approximately 32.9 million passages, with a task-adapted sealed policy compared with the standard model and other baselines; a separate 164-evaluation BM25 response study examines scorer and objective sensitivity.

Principal result. Against the standard dictionary, the sealed policy improves every reported metric: +3.667% nDCG@10, +4.043% Recall@100 and +5.340% AP@1000 among them. Query-level diagnostics show that most of 799 queries tie; depth bands expose where judged relevance is redistributed; objective reweighting explains the earlier Step-2 compromise; and the final index is incomparable to the standard one under pure global coarsening/refinement.

Boundary. Policy selection and evaluation use the same development case. The result is an in-sample feasibility demonstration, not evidence of unseen-query or cross-collection superiority.

Used in this volume. Chapter 16.

P9 — Two-sided adaptation and deployment verification

(Galambos, 2026i)

Two-sided lexical policy adaptation and verified deployment

Evidence unit. Positive lexical assignments, negative membership boundaries, model reopening, persistence, observed-vocabulary closure and replay verification, using the same English MIRACL development case that motivates P8.

Principal result. A positive-only realisation admits 11,291 unintended observed surfaces; negative boundaries and closure expose the distinction between intended policy, representational capability and realised deployment. The accompanying expressivity analysis shows that XSTEM-style terminal cut-and-append actions embed in the older Egothor patch executor, while the latter can also preserve interior material between edits.

Boundary. P9 and P8 are complementary analyses of one adaptation case, not independent retrieval replications. Expressivity inclusion does not imply that one learner will discover another system's policy.

Used in this volume. Chapter 15.

How the studies combine

No single paper carries the monograph's field-level thesis. P7 defines what the compiled policy object means; P2 and P6 show that induction inputs change that object; P1 tests behaviour beyond enumerated families; P3 supplies a system-independent error geometry; P5 characterises execution cost; P4 tests whether a reference policy can be changed while the mechanism is held fixed; P9 turns that change into a two-sided verified deployment; and P8 measures its downstream retrieval consequences. Chapter 17 is where these links are joined explicitly.

Three contributions belong to the synthesis rather than to an individual study: the verified historical reconstruction in Chapter 3; the cross-study argument developed in Chapters 17 and 18; and the reporting standard in Section 17.4. Those claims should be evaluated as claims of this volume, not retroactively attributed to P1–P9.

Bibliography

- Apache Software Foundation. Apache lucene 3.1 stempel analysis documentation. Apache Lucene API documentation, 2011. URL https://lucene.apache.org/core/3_1_0/. Historical availability witness; Stempel components are documented in Lucene 3.1.
- Apache Software Foundation. StempelStemmer — apache lucene api documentation. https://lucene.apache.org/core/9_9_1/analysis/stempel/org/apache/lucene/analysis/stempel/StempelStemmer.html, 2024. Documentation states that the core stemming algorithm and implementation are taken from the Egothor project; accessed 2026-09-21.
- Kirk Baker. XSTEM: An exemplar-based stemming algorithm, 2022. URL <https://arxiv.org/abs/2205.04355>. First submitted 9 May 2022; later revised.
- Toms Bergmanis and Sharon Goldwater. Context sensitive neural lemmatization with lematus. In *Proceedings of NAACL-HLT 2018*, pages 1391–1400, 2018.
- Martin Braschler and Bärbel Ripplinger. How effective is stemming and compounding for german text retrieval? *Information Retrieval*, 7(3–4):291–316, 2004.
- Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.
- Grzegorz Chrupała, Georgiana Dinu, and Josef van Genabith. Learning morphology with morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC)*, 2008.
- Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational Linguistics*, 26(1):3–16, 2000.
- Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.
- Elastic. Hybrid search. Elastic Documentation, 2026. URL <https://www.elastic.co/docs/solutions/search/hybrid-search>. Accessed 21 September 2026.
- Edward B. Fowlkes and Colin L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.
- Leo Galambos. Lemmatizer for document information retrieval systems in JAVA. In *SOFSEM 2001: Theory and Practice of Informatics*, volume 2234 of *Lecture Notes in Computer Science*, pages 243–252. Springer, 2001.

- Leo Galambos. Does a lexicon-trained stemmer generalize? a 20-language study of held-out lexical families. Research manuscript, 2026a. URL <https://doi.org/10.13140/RG.2.2.34383.85921>. Radixor Research Series, Paper 1.
- Leo Galambos. Edit costs as structural hyperparameters in dictionary-induced stemming: A multilingual sensitivity study. Research manuscript, 2026b. URL <https://doi.org/10.13140/RG.2.2.12573.47845>. Radixor Research Series, Paper 2.
- Leo Galambos. Classifying multilingual stemmers by pareto strata: Error regimes in pairwise lexical conflation. Research manuscript, 2026c. URL <https://doi.org/10.13140/RG.2.2.25995.25126>. Radixor Research Series, Paper 3.
- Leo Galambos. The stemming policy is data: Adapting a lexicon-trained stemmer to alternative conflation objectives. Research manuscript, 2026d. URL <https://doi.org/10.13140/RG.2.2.23898.09926>. Radixor Research Series, Paper 4.
- Leo Galambos. Runtime efficiency of lexicon-trained stemming: Compiled structure and a speed-reference-agreement envelope in radixor. Research manuscript, 2026e. URL <https://doi.org/10.13140/RG.2.2.27332.18569>. Radixor Research Series, Paper 5.
- Leo Galambos. Lexical resource consistency and learned transformation vocabulary in multilingual stemming. Research manuscript, 2026f. URL <https://doi.org/10.13140/RG.2.2.34043.07203>. Radixor Research Series, Paper 6.
- Leo Galambos. Compiling learned lexical transformations: Semantic canonicalization and accepting-leaf generalization in patch-command tries. Research manuscript, 2026g. URL <https://doi.org/10.13140/RG.2.2.12232.69128>. Radixor Research Series, Paper 7.
- Leo Galambos. Where retrieval quality goes: Rethinking stemming in modern search. Research manuscript, 2026h. URL <https://doi.org/10.13140/RG.2.2.22208.90887>. Radixor Research Series, Paper 8; shares the English MIRACL development case with Paper 9.
- Leo Galambos. From lexical edits to verified stemming policies: Two-sided adaptation in radixor. Research manuscript, 2026i. URL <https://doi.org/10.13140/RG.2.2.24725.49120>. Radixor Research Series, Paper 9; systems companion to Paper 8.
- Leo Galambos. Radixor project documentation: Benchmarks, protocols and reproduction material, 2026j. <https://leogalambos.github.io/Radixor/>.
- Donna Harman. How effective is suffixing? *Journal of the American Society for Information Science*, 42(1):7–15, 1991.
- Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985.
- David A. Hull. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84, 1996.

- Bart Jongejan and Hercules Dalianis. Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike. In *Proceedings of ACL-IJCNLP*, pages 145–153. Association for Computational Linguistics, 2009. URL <https://aclanthology.org/P09-1017/>.
- Matjaž Juršič, Igor Mozetič, Tomaž Erjavec, and Nada Lavrač. LemmaGen: Multilingual lemmatisation with induced ripple-down rules. *Journal of Universal Computer Science*, 16(9):1190–1214, 2010. doi: 10.3217/jucs-016-09-1190.
- Kimmo Kettunen and Eija Airio. Is a morphologically complex language really that complex in full-text retrieval? *Advances in Natural Language Processing*, pages 411–422, 2006.
- Robert Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–202, 1993.
- Julie Beth Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31, 1968.
- Brian W. Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta*, 405(2):442–451, 1975.
- James Mayfield and Paul McNamee. Single n -gram stemming. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 415–416, 2003.
- Microsoft. Hybrid search using vectors and full-text search in azure ai search. Microsoft Learn, 2026. URL <https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview>. Accessed 21 September 2026.
- Stoyan Mihov and Denis Maurel. Direct construction of minimal acyclic subsequential transducers. In *Implementation and Application of Automata (CIAA)*, volume 2088 of *Lecture Notes in Computer Science*, pages 217–229, 2001.
- Andrei Mikheev. Automatic rule induction for unknown-word guessing. *Computational Linguistics*, 23(3):405–423, 1997.
- Marcin Miłkowski and Dawid Weiss. Morfologik project: Initial public releases. Project blog and SourceForge project record, 2006. URL <https://morfologik.blogspot.com/2006/05/>. Project registered 13 May 2006; initial dictionary and Stempelator-related releases documented in May 2006.
- Marcin Miłkowski and Dawid Weiss. Morfologik-stemming 1.1. Morfologik project blog, 2007. URL <https://morfologik.blogspot.com/2007/05/>. Announcement of the Stempelator code rename to morfologik-stemming, 17 May 2007.
- Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1–2):177–201, 2000.
- Thomas Müller, Ryan Cotterell, Alexander Fraser, and Hinrich Schütze. Joint lemmatization and morphological tagging with lemming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2268–2274, 2015.

- National Institute of Standards and Technology. Ai risk management framework playbook. NIST AI Resource Center, 2023. URL <https://airc.nist.gov/airmf-resources/playbook/>. Living implementation guidance; accessed 21 September 2026.
- Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
- OpenSearch Project. Hybrid search. OpenSearch Documentation, 2026. URL <https://docs.opensearch.org/latest/vector-search/ai-search/hybrid-search/index/>. Accessed 21 September 2026.
- Chris D. Paice. Another stemmer. *ACM SIGIR Forum*, 24(3):56–61, 1990.
- Chris D. Paice. An evaluation method for stemming algorithms. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–50, 1994.
- Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
- Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, 1992.
- Elham Tabassi. Artificial intelligence risk management framework (ai rmf 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology, 2023. URL <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf>.
- Antal van den Bosch and Walter Daelemans. Memory-based morphological analysis. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 285–292, 1999.
- Dawid Weiss. Stempelator: A hybrid stemmer for the polish language. Technical Report RA-002/05, Institute of Computing Science, Poznań University of Technology, 2005.
- Xinyu Zhang, Nandan Thakur, Odunayo Ogundepo, Ehsan Kamalloo, David Alfonso-Hermelo, Xiaoguang Li, Qun Liu, Mehdi Rezagholizadeh, and Jimmy Lin. MIRACL: A multilingual retrieval dataset covering 18 diverse languages. *Transactions of the Association for Computational Linguistics*, 11:1114–1131, 2023.