Stratified Semantics for Negative Layout Constraints

Mireille D. Saunac
Department of Foundations of Computing, Institut Aurevin
Toulouse, France
m.saunac@aurevin.example

Kenjiro Matsunobe
Center for Language Engineering, Suirei Institute of Technology
Okayama, Japan

Eloísa Antúnez Cárden
Laboratory for Document Systems, Instituto Meridión
Rosario, Argentina

capstract:2611.00029v1 [cs.PL] 16 Nov 2026

Abstract

Page construction languages let authors reserve space and test geometric properties of earlier layout results. In a bound document the reservation can itself be observable, and the code that reports the result of a layout test can invalidate the test. We give a typed intermediate language for intentional blanks in which content, declarations of intent, and visible evidence of intent have separate effects. The semantics computes ordinary occupancy by stratified fixed points over a frozen page map, then emits attestations that carry no ordinary occupancy at their authorized sites. We prove convergence and preservation of occupancy under emission and characterize the cycles rejected by a syntax-directed checker. A separate validator detects changes to the compiled page stream after imposition. On 240 document fixtures, 228 are in the accepted fragment, and their outputs match independently specified page contracts. The 12 rejected fixtures produce 8 negative-cycle diagnostics and 4 scope diagnostics. The language distinguishes a page that prints a blankness notice from a blank page on which no notice was intended, even when both have identical ordinary content.

CCS Concepts

• Software and its engineering → Semantics; • Applied computing → Document preparation.

Keywords

document languages, stratified negation, effect systems, pagination, translation validation

Reference Format:
Mireille D. Saunac, Kenjiro Matsunobe, and Eloísa Antúnez Cárden. 2026. Stratified Semantics for Negative Layout Constraints. Preprint, November 16, 2026.

1 Introduction

A print contract can require page 48 to exist, have no running head or folio, and carry the sentence “This page intentionally left blank.” Existence is an obligation of pagination. The choice to omit the folio is an obligation of the page style. The sentence is typically inserted by a person after reviewing a proof. Moving that last operation into a program exposes a difficulty. If the operation is guarded by a test that the page has no marks, its result invalidates its premise. If the operation is unconditional, an editorial change may place a table on page 48 beside an authoritative statement about the table’s absence.

Some print workflows keep the sentence in a separate layer of the page file, which helps the press operator inspect the output [7, 14]. Layer membership alone supplies a weak specification. A line in the attestation layer can still refer to the wrong page, and a mark in the body layer can still be absent because a resource failed to load. Several systems in our fixture set pass a Boolean called empty between layout stages. The Boolean has meant, at different points in the same pipeline, that the page has no source paragraphs, that it has supplied no boxes to the compositor, or that the display list can be erased without removing any body ink. Only the last of those interpretations is even defined on the artifacts received by the printer.

On a spread whose left leaf reports a blank on the right, the proposition and its rendering have different addresses. Imposition moves physical surfaces while preserving the logical address named in the notice.

We take the page map as fixed and give ordinary occupancy one bit per page. Section 3 defines the constraints, Section 4 describes the linker and validator, and Section 5 compares their output with a set of print contracts.

2 Background

Knuth and Plass use boxes, glue, and penalties to choose paragraph breaks globally [11]. The page builder in TeX [10] handles a subsequent optimization problem and exposes page-dependent behavior through its output routine. Document algebras such as the libraries of Hughes and Wadler specify line structure compositionally [8, 19]. Their empty document serves as an identity for concatenation. An empty mark list contributes zero ordinary occupancy at a page position. It has extent and is observable even when its display list has length zero. In particular, concatenating a blank leaf with an eight-page gathering changes the number of pages in the document.

We apply finite monotone dataflow [5, 9] to a stratified relation over page positions [1, 2]. An SCC decomposition [17] yields the components. The report of a successful check needs a separate effect. Treating it as another occupancy fact can invalidate the result.

Our checker follows the standard use of judgments to restrict what programs can express [12, 15]. The witness layer requires a smaller output vocabulary than the ordinary layer. Work on final-artifact validation [13, 16] suggests a practical split between finding an appropriate compilation result and checking a result already found. Print-specific precursors infer likely insertions from imposed page streams [6] or carry manifest labels across

page extraction [3]. We keep a logical page identifier on every print surface and transport certificate targets using that identifier.

3 A Core Language for Page Intent

3.1 Pages and marks

A preprocessing pass fixes a map P of logical page identifiers, their trim rectangles, and a region reserved for evidence on each eligible site. This pass can run a general page builder. Once the map is fixed, every block in the core language has a site and a bounded rectangle. Ordinary blocks draw from a finite set of resource-resolved marks. Normalization removes zero-area clips, fully transparent marks, and whitespace-only text. The remaining primitive operations are clipped within the trim rectangle. We classify occupancy by the presence of an ordinary operation in that normalized list, even if another ordinary block later paints it over. This choice gives a reproducible pre-raster contract. A program that paints white over a black label has emitted two operations. A program whose label reduces to zero area has emitted neither.

A declaration $I(p)$ records the author's reservation of p for absence of ordinary marks. A leaf can end up empty without such a declaration, for instance when its float fits on the preceding page. A declared leaf with positive occupancy fails compilation.

An attestation is a value with two page arguments, a site p and a subject q. Its proposition is $I(q) \wedge \neg o_q$, where o_q is ordinary occupancy. Its rendering is licensed by a capability for the pair (p, q) and a certified region at p. When $p = q$ the permitted phrasing includes the usual notice for an intentional blank. When $p \neq q$ the referenced page must appear in the output, either as a logical reference in a machine-readable edition or as a resolved page identifier in the printed string. A notice on p. 48 about the absence of content on p. 47 therefore carries two distinguishable identifiers even if the two pages have the same measured ink density.

The approved region can hold one compact witness per site in the core language. The glyph stream comes from a fixed dictionary keyed by locale and proposition form. The untrusted author does not supply the string as an argument. This restriction prevents a paragraph from becoming free of occupancy by enclosing it in a witness declaration. A proof for page 48 can occupy 42 mm of its width. The value being proved still has no ordinary occupancy.

The effect checker uses two primitive emissions, ordinary marks at p and witness marks at p about q. In Eq. (1), A is the set of authorized pairs and D the set of declared subjects. A witness command has no guard-typed result. The only page-indexed inputs to guards are ordinary occupancy reads. The judgment assigns authority to attempt emission. Actual emission still depends on the valuation and on the witness glyphs fitting in the reserved region.

$$\frac{(p, q) \in A \qquad q \in D}{A; D \vdash \mathrm{witness}(p, q) : \mathrm{Evidence}(p, q)} \tag{1}$$

3.2 Ordinary occupancy

An ordinary block b has a site $s(b)$, a normalized list of operations, and a guard g_b. The syntax of a guard is a constant, an occupancy literal, or an expression obtained by conjunction and disjunction. Negative reads appear as literals $\neg o_q$. Guards may refer to the same site as their block. Define $a_b = 1$ exactly when normalization retains at least one operation. The program supplies one equation per page as in Eq. (2).

$$o_p = \bigvee_{b : s(b) = p} (a_b \wedge g_b(o)) \,. \tag{2}$$

An empty disjunction is false. We draw an edge from each read position to its output position and label it with the sign of the literal. An SCC is admissible when every internal edge is positive. The compiler solves admissible components in topological order, substituting already fixed values for external literals. It computes the least solution in the Boolean lattice of the component, beginning with all bits clear. There can be other fixed points even when all reads are positive. In a two-page spread where each page gets a folio only if the facing page is occupied, both folios are absent in the least solution. Adding a body block to either one places both folios.

Negative reads complicate the choice of a solution. Suppose each of two empty leaves prints an ordinary line if its neighbor is empty. There are two fixed points, each certifying the other leaf at the expense of its own blankness. The two results are exchanged by the automorphism that swaps the logical leaves and their rules. A fixed point chosen by increasing page number violates that symmetry. If a single leaf prints an ordinary line when it is itself empty, the equation becomes $o_p = \neg o_p$ and there is no two-valued fixed point at all. Our checker rejects both cases. Some SCCs with syntactically negative internal edges do have a unique solution, for example if an unconditional mark forces a bit high. The present checker intentionally uses guards' static read sets, and rejects these SCCs without trying to simplify them using possible solutions.

3.3 Emission and observation

Ordinary emission traverses blocks in source order after valuation. Within each logical page, it concatenates the normalized marks of enabled blocks. Thus a guard's evaluation order can affect compilation cost but cannot change paint order. A witness pass takes the resulting valuation and emits a draw operation only for a declared and unoccupied subject, after checking the capability and site rectangle. Reserved witness sites are optional on any page, including pages with ordinary material. The common self-attestation uses a site on the subject. A certificate printed on a binding slip can name another page without changing the occupancy of either page.

Let π_O project a decorated display list to its ordinary operations. Let $R(o)$ render the ordinary blocks under valuation o, and let $W(o, I)$ insert the licensed witnesses into a separate plane. The invariant is Eq. (3). It is equality of per-page ordered lists after normalization.

$$\pi_O\big(R(o) \oplus W(o, I)\big) = R(o). \tag{3}$$

A preview can return the unprojected stream to a press operator and the ordinary projection to the editor. A scanner given loose pages cannot distinguish a certified blank from the same sentence set as an ordinary paragraph.

4 Implementation and Independent Validation

The linker assigns a fresh logical key to every generated leaf. A private constructor represents witness capability in the intermediate language. The front end exposes it through a check on the declared site and subject. The linker checks edition-specific glyph bounds once the font is known.

4.1 Solving the page graph

Algorithm 1 builds the signed dependency graph, rejects each inadmissible component with a cycle witness, and fills the valuation before rendering. Internally we compile each guard into a tree of conjunction and disjunction nodes. After external literals have been substituted, initially true leaves go onto a queue. Each parent maintains a count of unsatisfied children for conjunction or a single latch for disjunction. A true clause wakes its site once and may wake new internal occurrences of the site bit. This representation avoids rescanning all guards after each addition. Each node undergoes at most one promotion inside its component.

Algorithm 1 Evaluation on a frozen page map

Require: linked pages P, ordinary blocks B, intent I
$G \leftarrow \mathrm{SignedReads}(P, B)$
for C **in** source-first component order of G **do**
  if C contains an internal negative edge **then**
    return a cycle diagnostic
  end if
  $o_C \leftarrow \mathrm{Propagate}(C, o)$
end for
if some declared page is occupied **then**
  return an intent diagnostic
end if
return $\mathrm{Emit}(P, B, o, I)$

The graph stage and propagation together cost $O(n + m)$ time and space, where n counts page positions and m counts all guard syntax nodes, including repeated occurrences. That bound excludes glyph shaping, geometry normalization, and emitting the marks. Counting just edges between distinct pages can undercount the work by an arbitrary factor. A hundred uses of a single page key can trigger one graph edge and a hundred separate guard updates. We retain occurrence lists in addition to deduplicated graph edges. The checker reports a negative read with the source span of its literal, then gives a positive or negative return path from the output site to its input site within the SCC.

An attempted ordinary read of a witness fails in the front end. Negative recursion over ordinary occupancy fails at linking. Appendix A gives the projection argument.

4.2 Checking the printed result

The compiler emits side data comprising page identifiers, per-page normalized ordinary mark lists, and positions and subjects of witnesses. The validator independently interprets guards after reconstructing and solving the signed graph from the checked source. It compares the target stream, after normalization and inverse imposition, with the expected ordinary lists and a dictionary expansion of the witness declarations. It also checks that physical rectangles match those supplied by the imposition map. We compare canonical display commands, including the font resource identifier and the glyph transform. The validator cannot infer a reliable claim from the typographic content of a scanned surface.

The validator detects three edits drawn from production reports [3, 14]. It catches dropped leaves by key, witnesses changed to ordinary text by effect, and swapped subjects by reference identity.

5 Evaluation

5.1 Corpus and outcomes

We prepared 240 fixtures in 12 families of 20. Families vary signature size, chapter boundaries, and whether evidence is reserved on the subject or elsewhere. Table 1 groups cases by page topology. The sample includes 12 fixtures constructed to trigger diagnostics.

Two readers independently transcribed the expected page keys, ordinary marks, and witness pairs before running any compiler. They reconciled 6 mirrored coordinates and 2 supplement identifiers by recording physical coordinates and logical identifiers in separate fields. Invalid cases specify a single expected error class and offending clause.

Table 1: Fixture coverage and strict agreement with the reference on accepted cases. Denominators for the three output strategies exclude fixtures with expected diagnostics.

Logical topology	Fixtures	Valid	Inline	Late pass	Stratified
Single-sided	90	90	72	87	90
Facing spreads	98	92	46	72	92
Multiple signatures	52	46	20	27	46
Total	240	228	138	186	228

The inline baseline treats announcements as ordinary display objects and tests whether the list at its current position is empty in emitter order. It was allowed to read the intent bit and use the same dictionary as the other strategies. The late-pass baseline solves ordinary occupancy using the same stratified solver, suppresses announcements during ordinary rendering, and adds notices based on intent and emptiness. It locates the target leaf by the current emitted position and always inserts the notice on that target. Both baselines use the same resource normalization. We score strict agreement on page identities and provenance, so a line produced at the right place without a recoverable subject identity passes only if the implied identity from its site is the required one.

Figure 1 decomposes the differences. A misplaced notice may also have a false subject once later content arrives. We count just one failure per fixture using the priority order: missing or extra leaf, wrong site or subject, then wrong occupancy or intent. The main deficit of the late pass arises in fixtures that place evidence on a page other than the subject. The three single-sided cases in which that approach fails use a reserved machine-readable receipt on the preceding leaf.

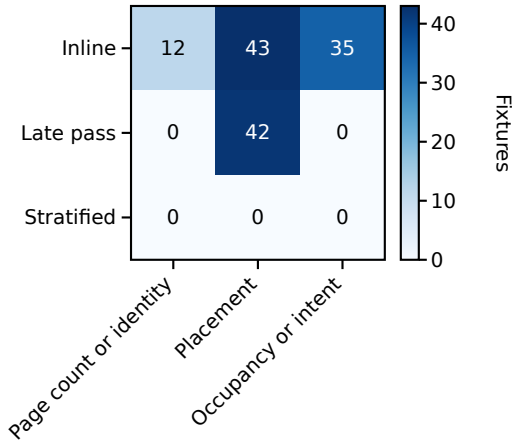


Figure 1: First difference from the reference on the 228 valid fixtures. Placement includes the evidence subject and site. Both baselines receive the same trim and resource data.

The signed checker reports eight negative cycles, four in facing spreads and four in longer impositions. The four scope errors consist of two sites with a forbidden subject and two attempts to move a certificate to a leaf lacking a reserved region. The absence of a reserved region fails before glyph metrics are considered. We also fed the validator 24 edited streams, within a separate test suite of 12 documents with two edits per document. Table 2 reports detections by edit. The stream comparison strips trace-only mark IDs but retains semantic layer and witness identifiers.

Table 2: Validation of single edits to compiled streams. Each edited stream has one injected difference. The first three rows are semantic differences and the final row is a provenance-preserving reserialization.

Edit	Streams	Rejected
Omitted blank leaf	6	6
Ordinary copy of witness	6	6
Transposed subjects	6	6
Object serialization order	6	0

5.2 Cost measurements

The scaling suite is independent of the contract fixtures. We build lists of size 128 through 32,768 logical pages. A chain distri-

bution gives all but the first page exactly one positive predecessor. A spread distribution gives the two leaves of a spread mutual positive reads and gives the second leaf a single dependency on the preceding spread. A dense distribution groups 32 pages into components with up to eight incoming positive reads per page and connects each block to its predecessor using a lower-stratum negative guard. Every group has a fixed, seed-dependent opportunity for ordinary marks. A generator chooses that seed from a fixed set of 30 seeds per size and graph family. The parser consumes the same in-memory representation in every condition. We timed normalization of guard expressions, graph construction and valuation, excluding geometry and IO. The test harness runs each seed twice before recording one elapsed time; generator and GC initialization are outside the timed section. The fixed workstation ran on a single pinned core. The elapsed-time distribution includes individual GC pauses within the timed section.

Figure 2 gives medians across the 30 generated instances in each condition, with bounds of the interquartile range given in Table 3 for the largest size. These intervals describe run and instance dispersion under the specified generator. We use the same seed index across the graph families but do not treat graph families as control treatments for computing a speed ratio.

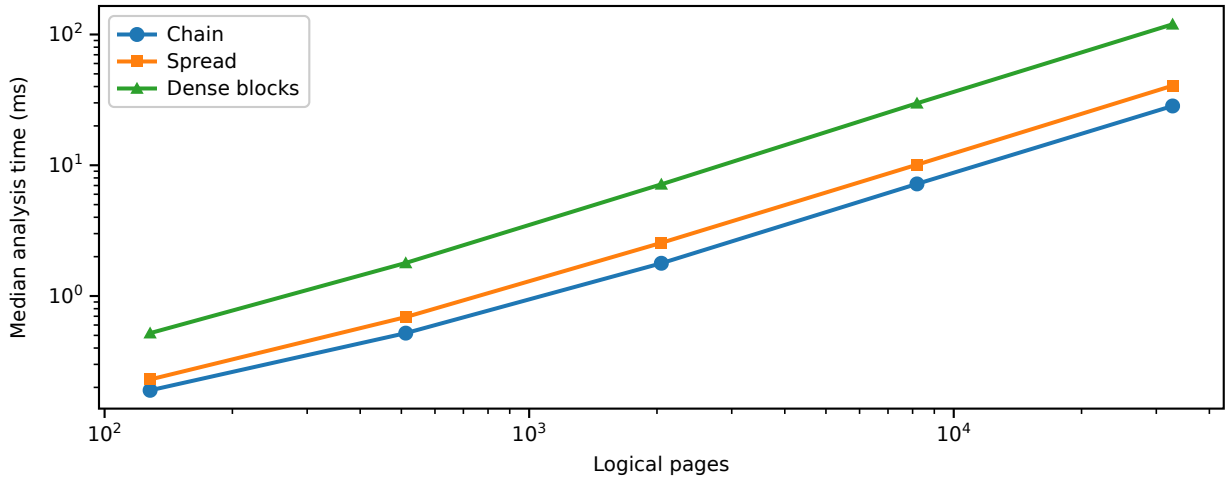


Figure 2: End-to-end analysis time by logical page count. Each point is the median of 30 generated instances. Both axes are logarithmic.

Table 3: Core analysis time for 32,768 logical pages. Quartiles describe 30 runs per graph family.

Graph family	Q1 (ms)	Median (ms)	Q3 (ms)
Chain	27.0	28.4	31.2
Spread	37.8	40.6	43.1
Dense blocks	113.1	119.7	131.5

An edit invalidates reachable SCCs and the witness sites reading them. For Figure 3 we grouped a 1,024-page document into signatures of 16 pages and chained dependencies within groups of varying length. We edited one ordinary clause at the first or final signature, then reset. Counts are invariant across groups of equal size. Rebuilding clears affected occupancy first, so removing the sole seed of a positive cycle can delete all its folios.

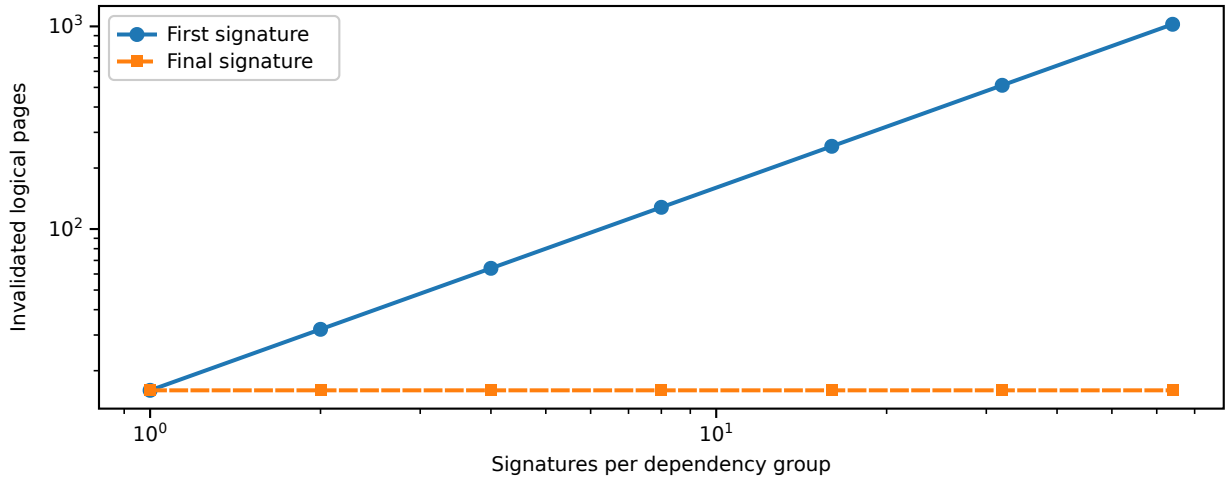


Figure 3: Extent of incremental invalidation in the 1,024-page suite. Sites are counted once even when their ordinary marks and their witnesses are both invalidated.

The count includes every invalidation. A later compiler pass could stop propagation as soon as it reestablishes the previous valuation.

6 Discussion

A practical issue is whether intentionality should be inferred. When the page builder opens every chapter on a recto, the resulting verso is often treated as a reserved blank. Some contracts instead allow ordinary material there, such as a plate or a correction inserted late. A builder may choose to synthesize a declaration from the chapter rule, provided the author consents to checking it at the end. Inferring I by testing that the final mark list is empty would let ordinary content disappear unnoticed when the typesetter loses a link to a resource. Express declaration before testing occupancy preserves the evidence needed to diagnose the missing content as separately unexpected.

The fixed page map is a restriction on optimization. A witness cannot borrow additional vertical space, push a paragraph onto a later page, or ask for a new empty leaf as part of its rendering. Suppose a printer needs an additional evidence page listing the blankness obligations of a signature. The page must be allocated during initial layout. Once registered as an evidence site it may testify for an adjacent eligible leaf under the core rules. A multi-subject receipt would require a checked list of target identities and a statically bounded witness region. Testing whether the receipt itself is an intentional blank still makes sense if all its emitted marks have the restricted witness effect. Therefore a single physical sheet could carry an exact account of several intentional absences while contributing zero ordinary occupancy.

Ink printed on the verso can show through a certified recto. Legibility and optical detection depend on paper and illumination. A verifier for visible pigment could classify an opaque white rectangle differently from our ordinary mark count. It would also have to account for occlusion, which can invalidate monotonicity under ordinary mark addition.

A Proof Details

Proposition 1 (Termination and uniqueness of the stratified least solution)**.** *For a finite linked program with no internal negative edge in any strongly connected component, component evaluation terminates. The stratified least solution is independent of the order in which incomparable components are evaluated and the order in which available facts are propagated inside a component.*

Proof. After substitution of external reads, all variable occurrences in the equations of a component are positive. The right-hand side of Eq. (2) hence gives a monotone map on a finite Boolean lattice. From bottom, iteration is an ascending chain and yields the least fixed point by finite stabilization [18]. The component implementation promotes bits one at a time when the right-hand side permits promotion. Every promoted bit belongs to every fixed point extending the already substituted predecessors, by induction on promotions. At quiescence each true clause has been propagated and each promoted bit remains justified by monotonicity. The result is a fixed point and is below every other fixed point. Incomparable components do not read one another. Their evaluation order cannot change the substituted values seen by either. ∎

Witness effects preserve ordinary absence. The ordinary emitter computes the precise marks whose feasibility was used to assign a_b. Witness expansion adds only commands bearing the witness effect, all with valid subjects and approved sites. Projection removes just those commands. Thus Eq. (3) follows by source-order induction over emitted blocks and a final pass over witness insertion positions. It continues to hold under injective renaming of page identifiers when capabilities and declared subjects are transported by the same renaming. Imposition supplies an injective mapping to print surfaces. Page numbers alone can fail that property when signatures restart numbering.

We tested the propagation proof obligations separately using generated small guards in the style of QuickCheck [4]. For every

admissible program with at most eight pages in that suite, we compared the least fixed point of each stratum with exhaustive enumeration within the stratum. For every rejected graph, replay traversed the implicated literal and the reported return path. When a comparison failed during development, the shrinker removed blocks and then simplified guards while preserving the predicate that failed, following the usual approach to input minimization [20]. The largest class of early mismatches arose because flattening folded the reserved display layer into ordinary ink. Keeping the original provenance through normalization removed that discrepancy.

B An Edition with a Duplicated Folio

Consider two gatherings compiled separately, each with a locally numbered page 16. In the first gathering that page closes a chapter. In the second gathering the corresponding page is reserved. Suppose evidence of the latter reservation belongs on the inward-facing page 15 in the second gathering. The unsuffixed string “16” names two logical sites. The linker requires an edition-specific site key in the receipt constructor, then emits the human-readable folio according to the witness dictionary. Our press profiles call that leaf “16 of gathering II” even if it physically faces page 1 of another gathering on the imposed sheet. After the sheet is folded the key still refers to the second logical page 16. Reusing the first key would survive a test comparing the visible number alone. The validator checks the resolved logical identity and would reject the stream.

The same example provides a regression for blank-page preservation under output extraction. Extracting gathering II must retain its reserved leaf and the proof site. Extracting only page 15 can retain the witness if the extraction contract permits a reference to an external subject. The core compiler treats that reference as unlinked. An edition authority can supply the missing contract as an import, but the current validator requires the subject’s attachment in the same stream. A print consisting of that single notice is evidence of a page that the print does not itself contain.

Acknowledgments

We thank the composition staff at the Institut Aurevin for identifying imposition cases that our original grammar could not express.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. 1988. Towards a Theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann.

[3] Meltem Balar and Frits Steunissen. 2023. Transport of Leaf Manifests through Document Extraction. *Transactions on Publishing Systems* 8 (2023), 119–146.

[4] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*..

[5] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the ACM Symposium on Principles of Programming Languages*..

[6] Inés M. Delario and Bertil Amseld. 2019. Recognizing Inserted Leaves in Imposed Print Streams. In *Symposium on Archival Document Engineering*..

[7] Dagmar Havelin and Louis J. Forten. 2015. Signature Boundaries and the Attribution of Unprinted Surfaces. *Journal of Composition Systems* 19 (2015), 41–57.

[8] John Hughes. 1995. The Design of a Pretty-Printing Library. In *Advanced Functional Programming*. Springer.

[9] Gary A. Kildall. 1973. A Unified Approach to Global Program Optimization. In *Proceedings of the ACM Symposium on Principles of Programming Languages*..

[10] Donald E. Knuth. 1984. *The TeXbook*. Addison-Wesley.

[11] Donald E. Knuth and Michael F. Plass. 1981. Breaking Paragraphs into Lines. *Software: Practice and Experience* 11 (1981), 1119–1184.

[12] Robin Milner. 1978. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences* 17 (1978), 348–375.

[13] George C. Necula. 1997. Proof-Carrying Code. In *Proceedings of the ACM Symposium on Principles of Programming Languages*..

[14] Mio Osada and Tatiana Balcarek. 2020. Layer Permissions for Supplementary Marks in Final Art. In *Workshop on Print Workflow Semantics*..

[15] Benjamin C. Pierce. 2002. *Types and Programming Languages*. MIT Press.

[16] Amir Pnueli, Michael Siegel, and Eli Singerman. 1998. Translation Validation. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

[17] Robert Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing* 1 (1972), 146–160.

[18] Alfred Tarski. 1955. A Lattice-Theoretical Fixpoint Theorem and Its Applications. *Pacific Journal of Mathematics* 5 (1955), 285–309.

[19] Philip Wadler. 2003. A Prettier Printer. In *The Fun of Programming*. Palgrave Macmillan.

[20] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Transactions on Software Engineering* 28 (2002), 183–200.