HERMES LABS

Tool Differentia:

Relational Static Analysis for AI Agent Tool Descriptions

Why individually valid tool descriptions can still fail as a set - and how a deterministic lint catches the missing distinction before runtime.

Rolando Bosch

Hermes Labs

ORCID 0009-0005-4896-1112

roli@hermes-labs.ai | https://hermes-labs.ai

Open-source reference implementation

LintLang 0.3.8

github.com/hermes-labs-ai/lintlang

Version 1.0.1 | August 6, 2026

Concept DOI 10.5281/zenodo.21817243

Technical Report

Contents

1 Abstract

Tool definitions are commonly validated one at a time, although an agent selects tools from a set of alternatives. Consequently, two descriptions may each be syntactically valid and individually accurate while jointly failing to explain why one tool should be selected instead of the other. We call the missing distinguishing information a **tool differentia**. This note defines tool differentia as a set-relative property and presents H1.6, a deterministic static analysis implemented in LintLang. H1.6 extracts meaning-bearing terms from tool names and descriptions, normalizes selected synonyms, and compares tool pairs within one parsed input. It reports mutual nondistinction when neither member carries a distinguishing analyzed term and directional domination when one member contributes none beyond those of the other. The method requires no model calls and is suitable for local and continuous-integration use. It is deliberately bounded: its lexicon is finite, its comparison scope is local, and the absence of a finding does not establish semantic distinguishability or runtime selection correctness.

2 Author's Note on AI-Generated Technical Writing

This report's manuscript prose was generated through a ChatGPT Deep Research workflow within an author-directed scope, then reviewed and approved for release by Hermes Labs. The Deep Research output is retained separately as supporting documentation. The selection of material, the publication decision, and responsibility for the claims and limitations stated here remain with Hermes Labs and the named author.

Keywords: AI agents; static analysis; tool descriptions; function calling; linting; software reliability.

Suggested citation: Bosch, Rolando. (2026, August 6). *Tool Differentia: Relational Static Analysis for AI Agent Tool Descriptions* (Version 1.0). Hermes Labs.

3 Problem and Related Work

Consider two tools:

```yaml
tools:
  - name: search_docs
    description: Search the documentation

  - name: find_docs
    description: Search through the docs
```

Each definition can satisfy its schema. Each description can also be factually accurate. Nevertheless, the pair provides no operational basis for choosing one tool over the other. The defect belongs neither description alone. It exists in the relation between them.

This separates two properties:

- **Intrinsic adequacy:** whether one tool's definition is sufficiently complete, specific, and struc-

turally valid when examined independently.

- **Relational adequacy:** whether the definition supplies information that separates the tool from plausible alternatives in the same selection namespace.

MCP and OpenAPI define structural fields for tools and operations, including names, descriptions, parameters, and schemas, but do not require one description to state how its operation differs from every neighboring candidate. [5, 6] This leaves a relational gap even when every individual definition is valid.

Description text matters to tool choice. Anthropic's tool-authoring guidance recommends stating what a tool does, when it should be used, and when it should not be used. Controlled research has also found that variations in tool descriptions can alter model preferences among competing tools. [1, 2] Runtime approaches such as ToolScope and JTPRO address overlapping tools or optimize tool instructions using retrieval, traces, or model-assisted processes. [3, 4] Tool differentia addresses an earlier and narrower question: can a deterministic pre-runtime check identify an explicit contrast that is absent from the authored definitions?

The contribution is therefore not a claim to discover tool ambiguity or replace behavioral evaluation. It is the formulation of missing distinguishing information as a static lint target that can be checked before, and used alongside, runtime evaluation.

4 Operational Definition

Let a tool definition be

$$t_i = (n_i, d_i, s_i),$$

where n_i is the tool name, d_i is its natural-language description, and s_i is its parameter schema.

Let ϕ be an explicit analysis function mapping the name and description to a finite set of analyzed terms:

$$M_i = \phi(n_i, d_i).$$

In H1.6, ϕ analyzes names and descriptions through deterministic token processing and a finite synonym lexicon. The schema s_i is not presently included in the term model. This omission is a limitation, not an assertion that schemas carry no distinguishing information.

For two tools t_i and t_j, define the directional differentia:

$$\Delta(i \mid j) = M_i \setminus M_j.$$

$\Delta(i \mid j)$ contains the analyzed terms attributed to t_i that are not covered by t_j under ϕ.

The pair is **mutually non-distinguishing** under ϕ when:

$$\Delta(i \mid j) = \varnothing \quad \wedge \quad \Delta(j \mid i) = \varnothing.$$

The pair exhibits **directional domination** when:

$$\Delta(i \mid j) = \varnothing \quad \wedge \quad \Delta(j \mid i) \neq \varnothing.$$

In this case, t_i is descriptionally dominated by t_j under the model: every analyzed term attributed to t_i is already covered by t_j, while t_j contributes additional terms.

The pair is **differentially adequate under** ϕ when:

$$\Delta(i \mid j) \neq \varnothing \quad \wedge \quad \Delta(j \mid i) \neq \varnothing.$$

These definitions are deliberately indexed by ϕ. A finding establishes a fact about the implemented analysis, not a proof of semantic equivalence. Likewise, an empty finding set does not prove that an agent can reliably distinguish the tools.

H1.6 approximates the candidate relation by comparing extracted tool definitions within one parsed input. It neither infers cross-file namespaces nor establishes which tools will actually compete for a particular user request. [7]

The central conceptual shift is therefore:

$$\text{relational description adequacy} \neq f(t_i)$$

but instead:

$$\text{relational description adequacy} = f(t_i, T, \phi).$$

A description's usefulness depends partly on the alternatives against which it must support a choice.

5 Reference Behavior and Scope

The versioned reference behavior is LintLang H1.6, released in LintLang 0.3.8. For each parsed input, it combines every tool's name and description, splits identifier-shaped names, tokenizes the resulting text, preserves Unicode alphanumeric words, and filters fixed stopword, non-discriminating, and low-information term sets. It canonicalizes a word only when the word or a generated suffix candidate is represented in its bounded synonym lexicon; otherwise the original word remains unchanged. The remaining terms form M_i. [7]

H1.6 compares only tools extracted from that input. It stays silent if either definition yields fewer than two analyzed terms, rather than treating missing evidence as domination. It also stays silent when a description explicitly names the other tool by its exact identifier-shaped name as a boundary; a declared alias is reported rather than suppressed. Otherwise it computes both directional set differences and reports mutual nondistinction or directional domination. The check is deterministic and

makes no model or network calls. [7]

This reference behavior is intentionally conservative. It treats lexical difference only as evidence under ϕ. Reproducibility comes at the cost of semantic coverage: neither a finding nor its absence supports a claim beyond the implemented term model.

6 Examples

6.1 Mutual synonym-equivalent descriptions

Before

```yaml
- name: search_docs
  description: Search the documentation

- name: find_docs
  description: Search through the docs
```

A simple word-overlap threshold can miss this pair because *documentation* and *docs* differ lexically, while *search* and *find* are separate surface forms. H1.6 canonicalizes the represented terms and finds no directional difference.

After

```yaml
- name: search_api_reference
  description: >
    Search API reference documentation for symbols, parameters,
    return types, and version-specific behavior. Do not use for tutorials.

- name: search_tutorials
  description: >
    Search step-by-step tutorials and task-oriented guides.
    Do not use for API symbol or parameter reference.
```

The repair is not merely more detail. It states a boundary: reference facts versus task-oriented instruction.

6.2 Conservative no-finding boundary

Input

```yaml
- name: get_order
  description: Return the order record matching the given identifier.

- name: get_orders
  description: Return order records matching the given identifiers.
```

H1.6 should remain silent. Singular versus plural can encode different cardinality, input shape, and return behavior. Treating the pair as mere morphological variation would generate a dangerous suggestion to merge or delete a legitimate operation.

This negative case is as important as a positive detector example. A useful linter must preserve short, domain-specific distinctions and stay silent when its evidence is insufficient.

7 Limitations and Practitioner Guidance

H1.6 is not a semantic equivalence prover. Its term model is based on explicit token processing and a finite synonym lexicon. Pairs using unrepresented synonyms, indirect paraphrases, world knowledge, or domain-specific equivalence may be missed.

The current synonym model is English-specific. Unicode-aware tokenization does not make the synonym analysis multilingual. The detector also evaluates only definitions extracted from one parsed input and does not infer which files, servers, or tool groups are presented together. [7]

The parameter schema, output schema, side effects, latency, authorization requirements, and implementation are not included in the current differentia set. Two similar descriptions may correspond to tools whose parameter schemas make their roles obvious; conversely, apparently distinct prose can conceal equivalent behavior.

A missing differentia does not demonstrate that a model will select the wrong tool. It identifies an authored absence under the analysis model. Runtime behavior depends on the model, system instructions, user query, tool ordering, schemas, examples, surrounding context, and potentially provider-specific tool-use behavior. Research demonstrating description sensitivity supports the relevance of the problem but does not validate H1.6's effectiveness. [2]

Precision and recall have not been measured against a labeled external corpus. H1.6 should therefore be used as a conservative authoring signal rather than a semantic verdict.

A useful tool description should answer not only **what the tool does**, but also at least one contrastive question:

Contrast dimension	Authoring question
Trigger	Under what user condition should this tool be selected?
Exclusion	When should this tool not be selected?
Object	What entity does it operate on that its neighbor does not?
Scope	Is it current versus historical, local versus remote, single versus bulk, or exact versus exploratory?
Authority	Is the operation read-only, privileged, destructive, or approval-gated?

Contrast dimension	Authoring question
Output	What result does it return that a neighboring tool does not?
Boundary	What request should be redirected to another named tool?

Anthropic's tool-authoring guidance strongly emphasizes detailed descriptions, trigger conditions, exclusions, parameters, and limitations. Tool differentia turns part of that prose guidance into a checkable relation between neighboring definitions. [1]

A compact authoring pattern is:

```
Use this tool for [positive trigger and scope].
Do not use it for [nearest confusable case];
use [neighboring tool] instead.
Returns [distinct output or effect].
```

Tool authors should prefer truthful boundary clauses over decorative vocabulary. Adding adjectives such as *advanced*, *fast*, *smart*, or *optimized* may create lexical uniqueness without creating a legitimate operational distinction.

The next evidence step is a labeled corpus of real tool pairs, followed by measurement of precision, recall, and the effect of confirmed repairs on tool-selection behavior. Possible implementation extensions include schema-aware differentia, multilingual lexicons, candidate clustering, and cross-file namespace resolution. These are future work, not capabilities claimed for H1.6.

8 Conclusion

Agents choose among alternatives, but most validation treats each alternative independently. Tool differentia names the distinguishing information required at that boundary. LintLang H1.6 provides one bounded implementation: it compares tools before runtime, reports mutual and directional absence of analyzed distinctions, and stays explicit about what its evidence cannot establish.

The contribution is neither a complete theory of tool selection nor a replacement for behavioral evaluation. It is a bounded engineering primitive: a checkable relation at the point where authored descriptions become a model's choice set.

9 References

1. Anthropic. *Tool use with Claude* and *Define tools*. Claude Developer Documentation.
2. Faghih, K., Wang, W., Cheng, Y., et al. (2025). Tool Preferences in Agentic LLMs Are Unreliable. *Proceedings of EMNLP 2025*, 20954-20969. DOI: 10.18653/v1/2025.emnlp-main.1060.
3. Ghoshal, S., Mittal, A., Singh, J., et al. (2026). JTPRO: A Joint Tool-Prompt Reflective Optimization Framework for Language Agents. *Findings of ACL 2026*, 40573-40595. DOI: 10.18653/v1/2026.findings-acl.2017.

4. Liu, M. M., Garcia, D., Parllaku, F., et al. (2026). ToolScope: Enhancing LLM Agent Tool Use through Tool Merging and Context-Aware Filtering. *Proceedings of ACL 2026*, 34095-34119. DOI: 10.18653/v1/2026.acl-long.1573.
5. Model Context Protocol. (2025). *Tools*.
6. OpenAPI Initiative. (2025). *OpenAPI Specification 3.2.0*.
7. Hermes Labs. (2026). *LintLang 0.3.8: Pairwise differentia for ambiguous tool definitions*; tagged H1.6 reference implementation.