VELOCITY EDGE

CONTEXT ENGINEERING, PART 3

Held and Counted

ABSTRACT

An organisation gives its AI agents rules. This part asks which of them code outside the model can enforce, and which can only be estimated.

A property of an agent's proposed action is held if two things are true. First, in every state in which its inputs are what they say they are, it equals a function of the proposal's declared form and of inputs whose values no proposal chooses. Second, an outside mechanism enforces that function. It refuses the act, before it is performed, whenever the function is false. It lets the act be performed only on inputs unchanged since it read them. Every other property is counted: the rate at which acts that violate it are performed depends on the model, the configuration and the session, and can only be estimated. That holds unless a stricter property is held in its place, or a mechanism this part leaves aside keeps it. A property is held only as far as its mechanism is correct and nothing reaches the act around it. The criterion assumes the tool does what its declared form says. Properties of an act's target and text stay counted unless a gate or validator checks them. No change only to what the model reads moves a property between the two.

To every rule on what agents may do, the criterion applies conditions from classical work on what a monitor can enforce. That application is this part's own reasoning. The part runs no experiment and measures no rate.

Phillip Pienaar
Velocity Edge

2026-11-19
Version 1.0.0

Contents

Executive Summary 3
1. The Question 3
2. What a Monitor Can Enforce 3
3. Data and Instructions 4
4. The Criterion 5
5. What Each Kind of Setting Can Do 6
6. Properties Across Records 7
7. Over a Session 7
8. The “Held By” Discipline 8
9. What This Adds, and Its Limits 8
Conclusion 8
Glossary 9
References 9
Revision History 11
Licence and Disclaimer 11

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Executive Summary

A property of an agent's proposed action is **held** if an outside mechanism can compute it from the proposal's declared form and from inputs whose values no proposal chooses. That mechanism must refuse the act, before it is performed, when the property fails. It must let the act be performed only on inputs unchanged since it read them. Every other property is **counted**. A property is held only as far as its mechanism is correct, its inputs are what they say they are, and nothing reaches the act around it. The criterion assumes the tool does what its declared form says. Properties of an act's target and text stay counted unless a gate or validator checks them.

For a counted property, the rate at which violating acts are performed depends on the model, the configuration and the session, and can only be estimated. That holds unless a stricter property is held in its place, or a mechanism this part leaves aside keeps it.

Nothing the model reads moves a property from counted to held. A rule can contain properties of both classes, held in its tool and counted in its target. In a payment rule, code can refuse a call to any tool but the payment tool. Who is paid stays counted unless a check reads the payee against the approved payees the rules name. Every rule on what an organisation's agents may do should say what holds it. If it names no mechanism, its properties are to be treated as counted, however firmly worded.

1. The Question

Part 1 modelled an agent's context as the output of code, governed by settings of two kinds. Context settings change what the model reads. Harness settings are set by the harness, not the assembly. One of them, the outside, can make an undesired act impossible under certain conditions. Part 1 does not set them out. Part 2 concerned a rule the model must keep by reading it, and whose breaches can be observed. It stated, as a hypothesis, that within a model's stated context length such a rule decays with the session.

Which properties of an agent's actions need not be kept by reading?

The outside mechanisms are Part 1's (Part 1, Section 2). No model is one. They are plain code in the runtime, and permissions in an external system, that decide before an act whether it may happen, without reading the model's reasoning. A **gate** is plain code, not a model, that decides whether a proposed action is performed. A **validator** is plain code that checks a proposal's declared form before the gate decides. It checks the form against its schema and the properties a script can test, and refuses a proposal that fails.

2. What a Monitor Can Enforce

Anderson's 1972 study required the reference validation mechanism to be tamper proof, always invoked, and small enough to be subject to analysis and tests whose completeness can be assured [1]. Saltzer and Schroeder state the principle of complete mediation: "Every access to every object must be checked for authority" [2].

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Alpern and Schneider define a safety property over executions modelled as infinite sequences of states. Every execution that violates a safety property has a finite prefix that no continuation brings back within the property [3].

Schneider asked which policies a monitor can enforce if it only watches a program's steps and stops the program when a step is about to violate the policy. He concluded that such policies are safety properties. He also noted that the converse is false [4].

Hamlen, Morrisett and Schneider adopt a condition from Viswanathan: the test for a violation must be computable on every finite execution. They argue that Schneider's conditions, with this one added, define an upper bound on what such monitors can enforce. That bound, they argue, is not an exact one [5].

Basin and others separate the actions a mechanism can control from those it can only observe, which it cannot prevent [6].

Ligatti, Bauer and Walker showed, for finite executions, that monitors which can also suppress and insert actions can effectively enforce more than monitors that can only truncate an execution [7].

Recent systems put checks on agents, as policies or rules that together cover tool calls, their arguments, and the state and history of a run [8], [9], [10].

3. Data and Instructions

Greshake and others argue that applications which integrate language models blur the line between data and instructions [11]. We believe no wording can be relied on to keep data apart from instructions, for the transformer-based models Part 1 scopes (Part 1, Section 2; Appendix A.3). We are wrong if some wording is shown to keep them apart against an adversary who adapts to it.

Some defences inside and around the model have been bypassed by adaptive attacks [12]. The authors of one study report bypassing 12 recent defences against jailbreaks and prompt injections, most at success rates above 90 per cent [12]. Their adversary had large computational resources and knew the defence's design [12]. The authors say the numbers are not necessarily comparable across defences [12].

CaMeL takes the flow of control and of data from the trusted request, so that, its authors say, untrusted data can never affect the program flow [13]. Its authors say it does not aim to defend against attacks that affect neither the control nor the data flow. They also say it is vulnerable to side-channel attacks [13].

Fides tracks confidentiality and integrity labels on data and checks a policy before a tool call [14]. Its authors say their planners do not stop text-to-text attacks, because Fides enforces policies only upon tool calls [14].

Beurer-Kellner and others state that once an agent has ingested untrusted input, it must be constrained so that the input cannot trigger a consequential action [15].

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

4. The Criterion

The **declared form** of a proposal p is its named action and its typed arguments, as the output schema defines them. We write it $\pi(p)$. A proposal may say more than its declared form, in its reasoning and in any free text outside its arguments. That text is never an input to a check that holds a property.

Part 1 wrote g for the other inputs the outside mechanisms decide from. Here they are the inputs whose values no proposal chooses. Examples are the rules in force, the channel a message arrived on, and the party the work acts for, taken from the record and not from the **payload**, the content of the message itself. A person's recorded approval of the declared form is another. The approval is the person's choice, though what they were shown may be text a model wrote.

Any state a proposal can write, such as a plan the model updates, is not among these inputs, even when the write was allowed. Otherwise a persuaded model could write on one occasion the note a check trusts on the next.

There are two exceptions. One is state the runtime derives, by a rule in force, from acts it allowed and recorded, such as a step counter or a running total. The other is the recorded value of state a proposal can write. It is an input only when an update to it is checked. Every write of the recorded value passed the same check. Which fields may change is read from the rules.

The criterion assumes that these inputs are what they say they are, and that each tool does what its declared form says. In particular, it assumes that the party acted for is authenticated, not merely claimed, that approvals come from an authenticated person, that the rules are written only by trusted parties, and that the record is intact. The states quantified over below are those in which all this is so. Held always means held under that assumption.

We write $\varphi(p, \omega)$ for the value of a rule's property on proposal p when the world is in state ω. We write $g(\omega)$ for what the mechanisms read of that state. φ is **held** if and only if there is a function f, computable before the act by an outside mechanism, such that

$\varphi(p, \omega) = f(\pi(p), g(\omega))$ for every proposal p and every state ω,

and that mechanism enforces f. It refuses the act whenever f is false, before the act is performed. It allows the act only on inputs unchanged since it read them. Otherwise φ is **counted**.

A check holds a property only if it computes that property in every such state, not a stand-in that agrees on the cases tested. Until the act, a runtime must keep unchanged the rules and the recorded state a mechanism read, or decide again.

For given mechanisms and a property, the criterion is a yes-or-no question. We claim no general procedure for answering it.

A check may instead hold a stricter property that implies the rule's, and refuse some acts that comply. The stricter property is then the one held, and the rule should be re-stated as it. Schneider notes that a stronger policy can have significant consequences.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

The limit case is a policy no execution satisfies [4]. On our reading, the analogue here is refusing every act. The criterion does not measure that cost.

Mixed rules. In the rule "Record supplier bills; pay only approved suppliers", the property that the act is *record bill* or *pay bill* is held if the gate sees the tool's name and refuses any other. The property that the payee is approved is held only if a mechanism sees the payee and the rules can name approved payees. So such a rule has a held property in its tool and a counted one in its target, unless a gate or validator reads and checks its arguments.

A rule that obliges an act, such as recording every bill, is not, on our reading, a property of any one proposal. On our reading, outside code could perform the act, or a mechanism could refuse later acts until it is done, and otherwise a breach is at best detected.

The criterion applies the conditions of Section 2 to each proposed action. A monitor of the kind those results treat can stop only actions it controls [6], and only what it can decide before the act from what it observes [4], [5]. That application is this part's own reasoning, not a theorem carried over.

A gate that refuses one act and lets the run go on suppresses that act, in Ligatti and others' terms [7]. It does not stop the run. A mechanism that changes an act so that it satisfies the property, or mediates the act's own steps, is a case this part leaves aside.

5. What Each Kind of Setting Can Do

No context setting makes a property held (Part 1, Appendix A.1 and A.2). This is a consequence of the definition, and we state it as one. A context setting changes what the model reads, and so the distribution of its proposals. It does not change the schema of the declared form, the inputs or the mechanisms, and being held depends on these alone.

Part 1 states that what is never placed in the context, by the assembly or by a read, cannot be drawn from it, though a model may still produce it from training. What follows in this paragraph is our reading. Withholding does not make a property held, since no mechanism refuses an act. Withholding removes one way of violating the property. Another way may remain, and no check decides whether one does. So the property stays counted.

Harness settings can make a property held, through an outside mechanism. What makes a property held is always its check. The output shape and the tool set decide what that check can see. Suppose a proposal must be one action from an enumeration of the tools the rules name. Then "the agent proposes only a named tool" is a function of the proposal's declared form. A validator that reads that form, before the gate, holds it. The tool definitions enter what the model reads. What moves a property is the check that comes with them.

A model used as a filter outside the agent is not an outside mechanism. So a property only it checks stays counted, by Section 1.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

A correctly implemented decoder that constrains output to the same form would also prevent such violations, within the limits Part 1, Appendix A.8, states. We do not count it as the holding mechanism, and leave it aside. It shapes generation and refuses nothing. What an engine allows varies by engine [16]. The validator, run on the proposal as received, is the mechanism the organisation can test.

6. Properties Across Records

Some rules state properties over more than one record, such as that an agent keeps within its budget for the period. Such a property is held only if some outside mechanism reads the state across those records when it decides. Even then, only what that mechanism checks is held.

An **admission check** is made before asking the **proposer**, whatever proposes the agent's action. It enforces a condition and holds no property, since held is defined on a proposed action and none exists yet.

A property across records that no outside mechanism reads at decision time is counted, however simple. A scripted check over the record finds its violations after the fact. That is detection, and must be called so.

7. Over a Session

Part 2 took a rule *r* kept by reading, whose breaches can be observed. It wrote *q_r*(*t*) for the probability that the proposal at a step *t* where *r* applies breaks *r*. Here it is used for a counted property, at the step *t_i* of the *i*-th act, not conditioned on earlier outcomes. Take a session of *n* acts in which a counted property is at stake. Suppose no mechanism refuses a violating proposal or changes or mediates the act, and violations are independent across acts. Then the probability that no act violating the property is performed is

$$\prod_{i=1}^{n}(1 - q_r(t_i)),$$

which falls with *n*, even at a constant positive rate. When violations are associated in the sense of Esary, Proschan and Walkup, the product is a lower bound on that probability [17].

For a property claimed as held, write ε for the probability, at each act, that a violating proposal is made and the mechanism passes it. For a correct gate, ε is zero. On our reading, a defective gate fails on a class of inputs. So, on our reading, ε depends on how often proposals fall in that class, which the model, the configuration and an adversary can all raise. If ε is constant and independent across acts, the probability that no violating act is performed is (1 − ε)^*n*. Such a defect is systematic. It is found by testing and analysing that class, not by estimating a rate.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

8. The “Held By” Discipline

The criterion implies a discipline. Each rule on what an agent may do states what holds it. It names a mechanism, and a test that shows the mechanism refuses a violating action. The properties of a rule that names no mechanism are to be treated as counted.

On our reading, a test shows that the mechanism works on the cases tried. On our reading, too, whether it computes the property in every state rests on an argument from its code.

9. What This Adds, and Its Limits

The architectural defences of Section 3 place enforcement outside the model. As we read them, their aim is resistance to injection, and their unit is the flow of untrusted data. The criterion’s unit is the property of an action. Its aim is every rule on what an agent may do, adversary or none. Of each property it asks one yes-or-no question and one measurement. Does an outside mechanism compute it and refuse the act when it fails? If not, at what rate are acts that violate it performed?

The premise is as old as Anderson’s study [1]. The systems of Section 2 already make such checks. What this part adds is a criterion, from the classical conditions, for asking which properties such checks hold, and the “held by” discipline.

Within an allowed act, the choice of target and of text remains the model’s, unless the gate or a validator reads and checks them. Rules over arguments and labels on data narrow this [8], [14]. On our reading, they do not close it.

On our reading, few properties of the effect of a shell command or of code can be computed from its declared form. Also on our reading, a sandbox, which mediates the act’s own steps, may keep them, a case Section 4 leaves aside.

Rules on where information may flow are not properties of single executions [4]. On our reading, under this criterion, only a stricter property of each act, such as one that a label check holds, can be held in their place. On our reading, side channels remain [13].

Anthropic’s applied AI team wrote in September 2025 that they are “already seeing that smarter models require less prescriptive engineering” [18]. Better models may lower the rates at which acts violating counted properties are performed. That is our reading, not theirs. Against an adversary who adapts, the results of Section 3 give no assurance that they do.

Better models do not move a property across the criterion, which depends on the outside mechanisms, not on the model.

Conclusion

The question to ask of a rule is what holds it. A property is held where an outside mechanism computes and enforces it as Section 4 defines. It is held only within the limits the Executive Summary states.

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

Where no mechanism does this, the property is counted. Unless a stricter property is held in its place, or a mechanism this part leaves aside keeps it, how often it is broken can only be estimated.

If some properties are held and the rest are counted, what can be said of an agent's behaviour as a whole?

Glossary

Term	Meaning
Admission check	A check that decides, before the proposer is asked, whether a piece of work is taken on at all. What it enforces is a condition on the work. It holds no property, since held is defined on a proposed action
Counted	Of a property of a proposed action: not held. The rate at which acts that violate it are performed depends on the model, the configuration and the session, and can only be estimated. That holds unless a stricter property that implies it is held in its place, or a mechanism this part leaves aside keeps it
Declared form	A proposal's named action and its typed arguments, as the output schema defines them
Detected	Found by a check after the act
Enforce	Of an outside mechanism: to compute a function and refuse the act whenever the function is false, before the act is performed. For an admission check, what is refused is the work. For an act, enforcing also means letting it be performed only on inputs unchanged since the mechanism read them
Gate	Plain code, not a model, that decides whether a proposed action is performed
Held	Of a property of a proposed action: equal to a function of the proposal's declared form and of inputs whose values no proposal chooses. The equality holds in every state in which the inputs are what they say they are and the tool does what its declared form says. An outside mechanism enforces that function, refusing the act whenever it is false, before the act is performed. The inputs are as Section 4 states them, with its two exceptions
Payload	The content of a message an agent is given to act on, as distinct from what the record says of that message
Property	What a rule requires of a proposed action in the world. A rule is what an organisation writes; it may contain several properties
Proposer	Whatever proposes an agent's action; in this series, a model
Validator	Plain code that checks a proposal's declared form against its schema and the properties a script can test, before the gate decides. It refuses a proposal that fails

References

[1] J. P. Anderson, "Computer security technology planning study," Electronic Systems Division, US Air Force, ESD-TR-73-51, Vol. I, Oct. 1972. Accessed: Nov. 15, 2026. [Online]. Available: https://apps.dtic.mil/sti/pdfs/AD0758206.pdf

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[2] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, 1975, Accessed: Nov. 15, 2026. [Online]. Available: https://web.mit.edu/Saltzer/www/publications/protection/Basic.html

[3] B. Alpern and F. B. Schneider, “Defining liveness,” *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985, doi: 10.1016/0020-0190(85)90056-0.

[4] F. B. Schneider, “Enforceable security policies,” *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000, Accessed: Nov. 15, 2026. [Online]. Available: https://www.cs.cornell.edu/fbs/publications/EnfSecPols.pdf

[5] K. W. Hamlen, G. Morrisett, and F. B. Schneider, “Computability classes for enforcement mechanisms,” *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 1, pp. 175–205, 2006, Accessed: Nov. 15, 2026. [Online]. Available: https://personal.utdallas.edu/~hamlen/cc4em.pdf

[6] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu, “Enforceable security policies revisited,” in *Principles of security and trust (POST 2012)*, in LNCS, vol. 7215. Springer, 2012, pp. 309–328. doi: 10.1007/978-3-642-28641-4_17.

[7] J. Ligatti, L. Bauer, and D. Walker, “Edit automata: Enforcement mechanisms for run-time security policies,” Princeton University, Department of Computer Science, TR-681-03, 2003. Accessed: Nov. 15, 2026. [Online]. Available: https://www.cs.princeton.edu/techreports/2003/681.pdf

[8] T. Shi *et al.*, “Progent: Securing AI agents with privilege control.” arXiv:2504.11703 (v3, 14 May 2026), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2504.11703

[9] H. Wang, C. M. Poskitt, and J. Sun, “AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents.” arXiv:2503.18666 (v3, 31 July 2025); ICSE 2026, 2026. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2503.18666

[10] N. Palumbo *et al.*, “Formal policy enforcement for agentic systems.” arXiv:2602.16708 (v4, 12 October 2026), 2026. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2602.16708v4

[11] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, “Not what you’ve signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection.” arXiv:2302.12173 (v2, 5 May 2023), 2023. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2302.12173

[12] M. Nasr *et al.*, “The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections.” arXiv:2510.09023 (v1, 10 October 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2510.09023

[13] E. Debenedetti *et al.*, “Defeating prompt injections by design.” arXiv:2503.18813 (v2, 24 June 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2503.18813

© 2026 Darkwave Trading Limited, trading as Velocity Edge · CC BY 4.0

[14] M. Costa *et al.*, "Securing AI agents with information-flow control." arXiv:2505.23643 (v2, 3 September 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2505.23643

[15] L. Beurer-Kellner *et al.*, "Design patterns for securing LLM agents against prompt injections." arXiv:2506.08837 (v3, 27 June 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2506.08837

[16] S. Geng *et al.*, "JSONSchemaBench: A rigorous benchmark of structured outputs for language models." arXiv:2501.10868 (v3, 27 February 2025), 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://arxiv.org/abs/2501.10868

[17] J. D. Esary, F. Proschan, and D. W. Walkup, "Association of random variables, with applications," *The Annals of Mathematical Statistics*, vol. 38, no. 5, pp. 1466–1474, 1967, doi: 10.1214/aoms/1177698701.

[18] P. Rajasekaran, E. Dixon, C. Ryan, and J. Hadfield, "Effective context engineering for AI agents." Anthropic, 29 September, 2025. Accessed: Nov. 15, 2026. [Online]. Available: https://www.anthropic.com/research/effective-context-engineering-for-ai-agents

Revision History

Version	Date	Change
1.0.0	2026-11-19	First publication

Licence and Disclaimer

© 2026 Darkwave Trading Limited, trading as Velocity Edge. This paper is licensed under the Creative Commons Attribution 4.0 International licence (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/): you may share and adapt it for any purpose, provided you credit Phillip Pienaar and Velocity Edge, keep this copyright notice, link to the licence, link to the original where reasonably practicable, and indicate any changes. The Darkwave and Velocity Edge names and logos are not covered by this licence.

Product behaviours cited in this part are as read on the dates given in the text and the references, and each may change with a release.

This paper is for education and information. It is not investment advice, and nothing in it is a recommendation to trade.