property-based-testing

property-based-testing

> *Catalogue généré le 2026-05-11*

En une phrase

Le property-based testing teste des **propriétés universelles** ("encoder puis décoder doit redonner la valeur d'origine") au lieu de cas spécifiques — la machine génère elle-même des milliers d'exemples pour traquer un contre-exemple.

Quand l'utiliser

- Tester une paire encode/decode, serialize/deserialize, toJSON/fromJSON (propriété "roundtrip").
- Tester un parseur (URL, config, protocole, format custom).
- Tester une fonction de normalisation (idempotence : appliquer 2 fois doit donner le même résultat).
- Tester une fonction pure (mathématique, tri, comparaison).
- Tester un smart contract Solidity/Vyper (invariants sur les balances, supply, ACL).

Comment l'invoquer

- **Slash command** : `/property-based-testing`
- **Phrases déclencheurs (texte)** : "property-based testing", "PBT", "Hypothesis", "QuickCheck", "invariant"
- **Auto-invocation** : Détection automatique sur patterns encode/decode, parsers, validators

Description détaillée

Le testing classique, c'est : "je liste 5 cas spécifiques (string vide, string longue, caractère bizarre…) et je vérifie chacun à la main". Le problème : tu testes ce à quoi tu penses, donc tu rates les cas auxquels tu ne penses pas. Le property-based testing renverse l'approche : tu décris une **propriété** que ton code doit respecter pour **toute** entrée valide, et la bibliothèque génère pour toi des centaines d'entrées aléatoires (et tordues : valeurs limites, Unicode bizarre, listes vides, négatifs…) pour vérifier la propriété.

Exemples concrets de propriétés :

- **Roundtrip** : `decode(encode(x)) == x` pour tout `x`.
- **Idempotence** : `normalize(normalize(x)) == normalize(x)`.
- **Commutativité** : `add(a, b) == add(b, a)`.
- **Invariant** : "Après transfert, la somme des balances reste constante" (smart contract).

Quand la lib trouve un contre-exemple qui casse ta propriété, elle fait du "shrinking" : elle simplifie automatiquement l'entrée pour te donner le contre-exemple minimal (par exemple `[1, 0]` au lieu de `[42, 17, 99, 0, -5]`). C'est ultra utile pour comprendre le bug. Outils populaires : Hypothesis (Python), QuickCheck (Haskell), fast-check (JS/TS), proptest (Rust), Echidna/Medusa (Solidity). Cette skill couvre les patterns pour tous ces langages.

Pour aller plus loin

Pour les exemples concrets, options de configuration et patterns avancés, voir le SKILL.md original.

Source

- **Plugin** : `trailofbits/property-based-testing`
- **Nom interne** : `property-based-testing`
- **Fichier** : `/home/thymon/.claude/plugins/cache/trailofbits/property-based-testing/1.1.0/skills/property-based-testing/SKILL.md`

Revision #2
Created 2026-05-11 21:20:00 UTC by thymon
Updated 2026-05-11 21:37:33 UTC by thymon