algorithmische Pseudocode-Routine ableiten.
Beides ist **streng am Minimal-Resonator orientiert**, ohne Semantik, Ontologie oder externe Annahmen.

I. Algorithmische Pseudocode-Routine

Minimal-Resonator mit Bruchdetektion

Ziel

Automatische Prüfung eines Quellenensembles auf:

- Resonanz
- Stabilität
- strukturelle Brüche
- resultierende Trägheit

---

1. Datenstrukturen

```
Source s_i:
    phi_i        // ordinale Phase
    R_i          // Menge relationaler Anforderungen
    sigma_i      // stabile innere Komponente (opaque, nicht benutzt)

Input:
    S = {s_1, s_2, ..., s_n}
    epsilon      // Toleranz für Phasennähe

Output:
    ResonanceCluster : Boolean
    Stability        : Boolean
    Breaks           : Set of (i, j)
    Inertia          : {high, low}
```

---

2. Hilfsfunktionen

Phasendistanz

```
PhaseDistance(phi_i, phi_j):
    return ordinal_distance(phi_i, phi_j)
```

> **Kommentar:**
> `ordinal_distance` ist **keine numerische Metrik**,
> sondern eine Ordnungsrelation („nah“ vs. „fern“).

Verträglichkeitsprüfung

```
Compatible(R_i, R_j):
    if exists configuration satisfying (R_i ∪ R_j):
        return true
    else:
        return false
```

Kommentar:
Keine Interpretation der Anforderungen,
nur Prüfung auf **gleichzeitige Erfüllbarkeit**.

---

3. Schritt 1: Resonanzprüfung (Definition 1)

```
CheckResonance(S, epsilon):
    for each pair (i, j) in S:
        if PhaseDistance(phi_i, phi_j) >= epsilon:
            return false
    return true
```

---

4. Schritt 2: Stabilitätstest (Definition 2)

```
CheckStability(S, epsilon):
    for each source s_k in S:
        S_prime = small_perturbation(S, s_k)
        if not CheckResonance(S_prime, epsilon):
            return false
    return true
```

Kommentar:
`small_perturbation` verändert **Form, nicht Rolle**
(z. B. Kürzung, Umformulierung, Detailverlust).

---

5. Schritt 3: Verträglichkeitsmatrix κ

```
ComputeCompatibilityMatrix(S):
    for each pair (i, j) in S:
        if Compatible(R_i, R_j):
            kappa[i][j] = 1
        else:
            kappa[i][j] = 0
    return kappa
```

---

6. Schritt 4: Bruchdetektion (Definition 3)

```
DetectBreaks(S, epsilon, kappa):
    Breaks = empty set
    for each pair (i, j) in S:
        if PhaseDistance(phi_i, phi_j) < epsilon
           and kappa[i][j] == 0:
            add (i, j) to Breaks
    return Breaks
```

---

7. Gesamtalgorithmus

```
MinimalResonator(S, epsilon):

    ResonanceCluster = CheckResonance(S, epsilon)

    if ResonanceCluster == false:
        return {
            ResonanceCluster: false,
            Stability: false,
            Breaks: all,
            Inertia: low
        }

    Stability = CheckStability(S, epsilon)

    kappa = ComputeCompatibilityMatrix(S)

    Breaks = DetectBreaks(S, epsilon, kappa)

    if Stability == true and Breaks is empty:
        Inertia = high
    else:
        Inertia = low

    return {
        ResonanceCluster: true,
        Stability: Stability,
        Breaks: Breaks,
        Inertia: Inertia
    }
```

---

8. Algorithmische Kernaussage

```
Resonanz ≠ Tragfähigkeit
Tragfähigkeit = Resonanz ∧ Stabilität ∧ (keine Brüche)
```