Эксперимент E: Путешествие во времени и Обратимость

Time Travel & Reversibility in RSL World

Версия: 1.0.0
Дата: 2025-12-20
Автор: RSL Research Team

---

1. Введение

Эксперимент E исследует **обратимость динамики** RSL-мира и возможность "путешествия во времени".

В детерминированной системе с обратимыми правилами математически существует возможность "отката" состояния. Однако ключевой вопрос: **является ли это "настоящим" путешествием во времени?**

1.1 Два варианта "путешествия во времени"

Вариант	Описание	Результат для наблюдателя
A. Глобальный откат	$S_{t+T} \to S_t$ (весь мир + наблюдатель)	Наблюдатель **не замечает** отката
B. Observer-only reset	$O_{t+T} \to O_t$ при фикс. X_{t+T}	**Амнезия**, не путешествие

---

2. Математическая модель

2.1 Структура состояния

Полное состояние системы:

$$S(t) = (X(t), O(t))$$

где:

- $X(t)$ — **микрофизика** (World): спины, ф-поле, граф
- $O(t)$ — **наблюдатель**: память, семантика, история

Компоненты микрофизики:

$$X(t) = \{s[0..N-1], \phi[0..N-1], \mathcal{G}, t\}$$

- $s_i \in \{-1, +1\}$ — спины (дискретные)
- $\phi_i \in \mathbb{R}$ — гравитационный потенциал
- $\mathcal{G} = (V, E)$ — граф связности
- $t \in \mathbb{Z}_{\geq 0}$ — дискретное время

Компоненты наблюдателя:

$$O(t) = \{\text{IFACEHistory}, \text{SemanticState}, \text{cache}\}$$

2.2 Оператор эволюции

Эволюция мира определяется оператором F:

$$S(t+1) = F(S(t))$$

Для **обратимой** системы существует F^{-1}:

$$S(t) = F^{-1}(S(t+1))$$

2.3 SM-правила (обратимые)

Стандартная Модель использует **взаимно-обратные** правила:

$$R_{\rightarrow}: \quad (++-) \to (-++)$$

$$R_{\leftarrow}: \quad (-++) \to (++-)$$

Это гарантирует **микроскопическую обратимость**: каждое применение правила может быть отменено.

2.4 Динамика ф-поля

Гравитационный потенциал эволюционирует по уравнению:

$$\phi^{(t+1)} = \phi^{(t)} + D_\phi \cdot \Delta_{\mathcal{G}} \phi^{(t)} + \beta \cdot \rho_s^{(t)} - \gamma \cdot \phi^{(t)}$$

где:

- $\Delta_{\mathcal{G}}$ — лапласиан графа
- $\rho_s = s - \langle s \rangle$ — отклонение от среднего (источник)
- D_ϕ, β, γ — коэффициенты диффузии, источника, затухания

---

3. Хеш-функции для верификации

3.1 micro_hash (только физика)

$$\texttt{micro\_hash}(X) = \mathrm{SHA256}(t \| N \| s \| \mathrm{quant}(\phi) \| \mathrm{canon}(E))$$

Компоненты:

- t — время (int64)
- N — размер (int32)
- s — спины (int8 array)
- $\mathrm{quant}(\phi)$ — квантизованное ф-поле
- $\mathrm{canon}(E)$ — канонические рёбра

Квантизация ф-поля:

$$\mathrm{quant}(\phi)_i = \mathrm{round}\left(\frac{\phi_i - \langle\phi\rangle}{\varepsilon}\right), \quad \varepsilon = 10^{-12}$$

Канонизация рёбер:

$$\mathrm{canon}(E) = \mathrm{sort}\left(\{(\min(u,v), \max(u,v)) : (u,v) \in E\}\right)$$

3.2 full_hash (физика + наблюдатель)

$$\texttt{full\_hash}(S) = \mathrm{SHA256}(\texttt{micro\_hash}(X) \| \mathrm{hash}(O))$$

---

4. Гипотезы эксперимента

ID	Гипотеза	Формальный критерий
H1	SM-правила обратимы	$\forall s : R_{\leftarrow}(R_{\rightarrow}(s)) = s$
H2	ф-поле обратимо при фикс. s	$\lvert\phi_0 - \phi_{\mathrm{rollback}}\rvert_\infty < 10^{-10}$
H3	micro_hash сохраняется после rollback	$h(S_0) = h(F^{-T}(F^T(S_0)))$
H4	Observer reset не меняет физику	$\texttt{micro\_hash}(X)$ инвариант при $\texttt{reset}(O)$

---

5. Тесты

5.1 Тест R0: Микро-обратимость

Процедура:

1. Сохранить S_0, вычислить $h_0 = \texttt{micro\_hash}(S_0)$
2. Выполнить T шагов вперёд: $S_T = F^T(S_0)$
3. Выполнить T шагов назад: $S'_0 = F^{-T}(S_T)$
4. Вычислить $h'_0 = \texttt{micro\_hash}(S'_0)$

Критерий: $h_0 = h'_0$

5.2 Тест T5: Детерминизм

Процедура:

1. Запустить эволюцию с seed $k \to$ получить h_1
2. Запустить эволюцию с seed $k \to$ получить h_2

Критерий: $h_1 = h_2$

5.3 Тест T6: Целостность StepRecord

StepRecord — журнал шага для отката:

```
StepRecord = {
    t: int,                    # время ДО шага
    s_before: np.ndarray,      # спины до
    phi_before: np.ndarray,    # φ до
    applied_positions: List[int],   # позиции применённых
правил
    applied_rules: List[str],       # имена правил
    pre_hash: str,             # hash до
    post_hash: str             # hash после
}
```

Критерий: Цепочка хешей: $\texttt{record}[i].\texttt{post\_hash} = \texttt{record}[i+1].\texttt{pre\_hash}$

5.4 Тест T7: Стабильность Hash

Критерий: $\texttt{micro\_hash}(S)$ детерминирован — многократные вызовы дают одинаковый результат.

5.5 Тест OR: Observer Reset

Процедура:

1. Вычислить $h_X = \texttt{micro\_hash}(X)$
2. Выполнить $\texttt{reset}(O)$ (очистить память наблюдателя)

3. Вычислить $h'_X = \texttt{micro\_hash}(X)$

Критерий: $h_X = h'_X$ (физика не изменилась)

---

6. Философские следствия

6.1 Почему обратимость ≠ путешествие во времени

Глобальный откат ($S_T \to S_0$):

- Состояние мира восстановлено
- Память наблюдателя **тоже** восстановлена
- Наблюдатель "забыл" что был в будущем
- С его точки зрения — **ничего не произошло**

Observer-only reset ($O_T \to O_0$ при фикс. X_T):

- Мир остаётся в состоянии X_T
- Наблюдатель "забыл" историю
- Это **амнезия**, не путешествие во времени
- Законы физики мира изменились (с точки зрения наблюдателя)

6.2 Ключевой вывод

> **Reversibility ≠ Time Travel**
>
> Математическая обратимость динамики не даёт возможности "изменить прошлое". Внутренний наблюдатель либо не замечает отката, либо испытывает амнезию.

---

7. Параметры эксперимента

Параметр	Значение	Описание
N	512	Количество узлов
α	2.0	Параметр графа
L	3	Длина паттерна правил
T_{forward}	50	Шагов вперёд
T_{backward}	50	Шагов назад
N_{seeds}	10	Количество тестовых seeds

Параметр	Значение	Описание
ε_ϕ	10^{-12}	Точность квантизации ϕ

---

8. Структура ноутбука

1. **Импорты и параметры**
2. **Создание мира с SM-правилами**
3. **Инструменты обратимости** (StepRecord, hash функции)
4. **ReversibleEvolutionEngine**
5. **Тест T5** — Детерминизм
6. **Тест R0** — Микро-обратимость
7. **Тест T6** — Целостность StepRecord
8. **Тест T7** — Стабильность Hash
9. **Тест OR** — Observer Reset
10. **Визуализация**
11. **Финальный отчёт**

In [1]:
```
# ============================================================================
# ЭКСПЕРИМЕНТ E: ПУТЕШЕСТВИЕ ВО ВРЕМЕНИ И ОБРАТИМОСТЬ
# ============================================================================
#
# Цель: Проверить обратимость динамики RSL-мира
#
# Гипотезы:
#   H1: SM-правила (++- ↔ -++) обратимы
#   H2: φ-поле обратимо при фиксированных спинах
#   H3: Полный micro_hash сохраняется после rollback
#   H4: Observer reset не меняет физику мира
# ============================================================================

import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from datetime import datetime, timezone
from dataclasses import dataclass, field
from typing import Dict, List, Tuple, Optional, Set, Any
import subprocess
import hashlib
import json
from scipy.sparse.linalg import spsolve
from scipy import stats
from collections import deque
import copy

# Определяем project_root
```

```
project_root = str(Path(os.getcwd()).parent) if os.path.basename(os.getcwd()
sys.path.insert(0, project_root)

VERSION = "1.0.0"

# Импорт симулятора World
from world.core.world import World, WorldConfig
from world.core.rules import RuleSet, Rule
from world.core.lattice import Lattice, LatticeState
from world.core.evolution import EvolutionEngine
from world.core.graph_structure import GraphStructure, GraphConfig

# Настройка matplotlib
plt.rcParams['figure.figsize'] = (14, 10)
plt.rcParams['font.size'] = 11
plt.rcParams['axes.unicode_minus'] = False

print("=" * 70)
print("ЭКСПЕРИМЕНТ E: ПУТЕШЕСТВИЕ ВО ВРЕМЕНИ И ОБРАТИМОСТЬ")
print("=" * 70)

# ============================================================================
# БАЗОВЫЕ ПАРАМЕТРЫ
# ============================================================================
RSL_N = 512
RSL_ALPHA = 2.0
RSL_L = 3
BASE_SEED = 42

# Параметры тестов обратимости
T_FORWARD = 50           # Шагов вперёд
T_BACKWARD = 50          # Шагов назад
N_SEEDS = 10             # Количество seeds для тестирования

# Параметры φ-поля
PHI_QUANT_EPS = 1e-12    # Квантизация для hash

# Пороги для гипотез
THRESHOLDS = {
    # H1: Обратимость спинов
    'h1_spin_match_rate': 1.0,     # Должно быть 100%
    # H2: Обратимость φ-поля
    'h2_phi_max_diff': 1e-10,      # max|φ_0 - φ_rollback|
    # H3: micro_hash match
    'h3_hash_match': True,         # Должен совпадать
    # H4: Observer reset
    'h4_world_unchanged': True,    # X не меняется
    # Тесты
    'test5_determinism': True,
    'test6_record_integrity': True,
    'test7_hash_stability': True,
}

# ============================================================================
# ВЕРСИОНИРОВАНИЕ И RUN_ID
# ============================================================================
```

```python
def get_git_info() -> Dict[str, Any]:
    """Получает информацию о git commit."""
    try:
        commit = subprocess.check_output(['git', 'rev-parse', 'HEAD'],
                                         cwd=project_root, stderr=subprocess
        dirty = subprocess.call(['git', 'diff', '--quiet'], cwd=project_root
        return {'commit': commit, 'dirty': dirty}
    except:
        return {'commit': 'unknown', 'dirty': True}

git_info = get_git_info()
RUN_ID = f"{datetime.now().strftime('%Y%m%d_%H%M%S')}_{git_info['commit']}"
TIMESTAMP_UTC = datetime.now(timezone.utc).isoformat()

# Создаём директории для артефактов
RUN_DIR = Path(project_root) / 'data' / 'experiment_E' / RUN_ID
RUN_TABLES_DIR = RUN_DIR / 'tables'
RUN_PLOTS_DIR = RUN_DIR / 'plots'
RUN_TABLES_DIR.mkdir(parents=True, exist_ok=True)
RUN_PLOTS_DIR.mkdir(parents=True, exist_ok=True)

print(f"\nПараметры мира:")
print(f"  N = {RSL_N}")
print(f"  α = {RSL_ALPHA}")
print(f"  L = {RSL_L}")
print(f"  BASE_SEED = {BASE_SEED}")

print(f"\nПараметры тестов:")
print(f"  T_FORWARD = {T_FORWARD}")
print(f"  T_BACKWARD = {T_BACKWARD}")
print(f"  N_SEEDS = {N_SEEDS}")

print(f"\nВерсионирование:")
print(f"  RUN_ID = {RUN_ID}")
print(f"  Git commit = {git_info['commit']} (dirty={git_info['dirty']})")
print(f"  Директория: {RUN_DIR}")
```

```
======================================================================
ЭКСПЕРИМЕНТ E: ПУТЕШЕСТВИЕ ВО ВРЕМЕНИ И ОБРАТИМОСТЬ
======================================================================

Параметры мира:
  N = 512
  α = 2.0
  L = 3
  BASE_SEED = 42

Параметры тестов:
  T_FORWARD = 50
  T_BACKWARD = 50
  N_SEEDS = 10

Версионирование:
  RUN_ID = 20251220_074613_0f04eb87
  Git commit = 0f04eb87 (dirty=False)
  Директория: /home/catman/Yandex.Disk/cuckoo/z/reals/libs/Experiments/Spac
e/World/data/experiment_E/20251220_074613_0f04eb87
```

Part 2: Создание базового мира и SM-правил

In [2]:
```python
# ======================================================================
# ЧАСТЬ 2: СОЗДАНИЕ МИРА И SM-ПРАВИЛ
# ======================================================================

print("=" * 70)
print("ЧАСТЬ 2: БАЗОВЫЙ МИР С ОБРАТИМЫМИ ПРАВИЛАМИ")
print("=" * 70)

# 1. Создаём SM-правила (обратимые: ++- ↔ -++)
sm_rules = RuleSet(rules=[
    Rule(name="sm_R", pattern=[1, 1, -1], replacement=[-1, 1, 1]),   # ++- -
    Rule(name="sm_L", pattern=[-1, 1, 1], replacement=[1, 1, -1]),   # -++ -
])

print("\nSM-правила (Стандартная Модель):")
for rule in sm_rules.rules:
    p_str = ''.join('+' if x == 1 else '-' for x in rule.pattern)
    r_str = ''.join('+' if x == 1 else '-' for x in rule.replacement)
    print(f"  {rule.name}: {p_str} → {r_str}")

print("\n  ВАЖНО: sm_R и sm_L — взаимно обратные правила!")
print("  Это гарантирует микроскопическую обратимость.")

# 2. Создаём конфигурацию мира
world_config = WorldConfig(
    N=RSL_N,
    initial_state="random",    # Случайное начальное состояние
    defect_density=0.3,         # 30% дефектов (s=-1)
    graph_alpha=RSL_ALPHA,
    D_phi=0.1,
    beta_source=0.01,
    gamma_decay=0.001,
```

```
)

# 3. Создаём мир
np.random.seed(BASE_SEED)
world = World(world_config, sm_rules)

print(f"\nМир создан: {world.summary()}")
print(f"  Узлов: {world.N}")
print(f"  Рёбер: {world.graph.n_edges}")

# 4. Начальное состояние
n_minus = np.sum(world.s == -1)
n_plus = np.sum(world.s == 1)
print(f"\nНачальное состояние:")
print(f"  s=+1: {n_plus}, s=-1: {n_minus}")
print(f"  Топологический заряд Q = {world.topological_charge}")
```

```
======================================================================
ЧАСТЬ 2: БАЗОВЫЙ МИР С ОБРАТИМЫМИ ПРАВИЛАМИ
======================================================================

SM-правила (Стандартная Модель):
  sm_R: ++- → -++
  sm_L: -++ → ++-

  ВАЖНО: sm_R и sm_L — взаимно обратные правила!
  Это гарантирует микроскопическую обратимость.

Мир создан: World(N=512, t=0, Q=155, φ_range=[0.000, 0.000], graph: 813 edge
s, α=2.0)
  Узлов: 512
  Рёбер: 813

Начальное состояние:
  s=+1: 357, s=-1: 155
  Топологический заряд Q = 155
```

Part 3: Инструменты обратимости — StepRecord и Hash функции

In [3]:
```
# ======================================================================
# ЧАСТЬ 3: ИНСТРУМЕНТЫ ОБРАТИМОСТИ
# ======================================================================

print("=" * 70)
print("ЧАСТЬ 3: ИНСТРУМЕНТЫ ОБРАТИМОСТИ")
print("=" * 70)

@dataclass
class StepRecord:
    """
    Запись одного шага эволюции для обратимости.

    Хранит информацию о применённых правилах для точного отката.
    """
```

```
    t: int                                  # Время ДО шага
    s_before: np.ndarray                    # Состояние спинов до шага
    phi_before: np.ndarray                  # φ-поле до шага
    applied_positions: List[int]            # Позиции применённых правил
    applied_rules: List[str]                # Имена применённых правил
    pre_hash: str = ""                      # Hash до шага
    post_hash: str = ""                     # Hash после шага

    def __repr__(self):
        return f"StepRecord(t={self.t}, applied={len(self.applied_positions)}


def sha256_bytes(parts: List[bytes]) -> str:
    """Вычисляет SHA256 от списка байтов."""
    h = hashlib.sha256()
    for p in parts:
        h.update(p)
    return h.hexdigest()


def canon_edges(edges: List[Tuple[int, int]]) -> np.ndarray:
    """Канонизирует рёбра графа (сортировка)."""
    e = np.array([(min(u, v), max(u, v)) for (u, v) in edges], dtype=np.int3
    idx = np.lexsort((e[:, 1], e[:, 0]))
    return e[idx]


def quantize_phi(phi: np.ndarray, eps: float = PHI_QUANT_EPS) -> np.ndarray:
    """
    Квантизует φ-поле для стабильного хеширования.

    Нормализует по среднему (gauge fixing) и округляет.
    """
    phi0 = phi - phi.mean()  # Gauge: среднее = 0
    return np.rint(phi0 / eps).astype(np.int64)


def compute_micro_hash(world: World) -> str:
    """
    Вычисляет micro_hash состояния мира.

    Включает: t, s, phi (квантизованное), edges графа.
    Не включает: omega_cycles, IFACE, SemanticState.
    """
    parts = []

    # Время
    parts.append(np.int64(world.t).tobytes())

    # Размер
    parts.append(np.int32(world.N).tobytes())

    # Спины
    parts.append(world.s.astype(np.int8).tobytes(order="C"))

    # φ-поле (квантизованное)
```

```
    phi_q = quantize_phi(world.phi)
    parts.append(phi_q.tobytes(order="C"))

    # Рёбра графа (канонические)
    edges = canon_edges(world.graph.edges)
    parts.append(np.int32(len(edges)).tobytes())
    parts.append(edges.tobytes(order="C"))

    return sha256_bytes(parts)


def compute_spin_hash(s: np.ndarray) -> str:
    """Хеш только спинов."""
    parts = [
        np.int32(len(s)).tobytes(),
        s.astype(np.int8).tobytes(order="C")
    ]
    return sha256_bytes(parts)


def compute_phi_hash(phi: np.ndarray) -> str:
    """Хеш φ-поля (квантизованного)."""
    phi_q = quantize_phi(phi)
    parts = [
        np.int32(len(phi)).tobytes(),
        phi_q.tobytes(order="C")
    ]
    return sha256_bytes(parts)


# Тест хеш-функций
print("\n[Тест хеш-функций]")
h1 = compute_micro_hash(world)
h2 = compute_micro_hash(world)
print(f"  micro_hash (первый):  {h1[:16]}...")
print(f"  micro_hash (второй):  {h2[:16]}...")
print(f"  Совпадают: {h1 == h2}")

print(f"\n  spin_hash: {compute_spin_hash(world.s)[:16]}...")
print(f"  phi_hash:  {compute_phi_hash(world.phi)[:16]}...")
```

```
======================================================================
ЧАСТЬ 3: ИНСТРУМЕНТЫ ОБРАТИМОСТИ
======================================================================

[Тест хеш-функций]
  micro_hash (первый):  f6ebb4449115d0fb...
  micro_hash (второй):  f6ebb4449115d0fb...
  Совпадают: True

  spin_hash: aa48064aa1ba453b...
  phi_hash:  5ed1fa7ba1e8bf51...
```

Part 4: Обратимый движок эволюции

In [4]:

```
# ============================================================================
# ЧАСТЬ 4: ОБРАТИМЫЙ ДВИЖОК ЭВОЛЮЦИИ
# ============================================================================

print("=" * 70)
print("ЧАСТЬ 4: ОБРАТИМЫЙ ДВИЖОК ЭВОЛЮЦИИ")
print("=" * 70)

class ReversibleEvolutionEngine:
    """
    Движок эволюции с поддержкой обратимости.

    Ключевые особенности:
    1. Сохраняет StepRecord для каждого шага
    2. Позволяет откатить шаги в обратном порядке
    3. Детерминированный выбор позиций применения правил
    """

    def __init__(self, ruleset: RuleSet):
        self.ruleset = ruleset
        self.step_records: List[StepRecord] = []

        # Создаём словарь обратных правил
        self.inverse_rules = {}
        for rule in ruleset.rules:
            # Паттерн и замена меняются местами
            key = tuple(rule.replacement)
            self.inverse_rules[key] = rule.pattern

    def find_matches(self, s: np.ndarray, pattern: List[int]) -> List[int]:
        """Находит все позиции, где паттерн совпадает."""
        N = len(s)
        L = len(pattern)
        matches = []

        for i in range(N - L + 1):
            if all(s[i + j] == pattern[j] for j in range(L)):
                matches.append(i)

        return matches

    def step_forward(self, world: World, record: bool = True) -> Optional[St
        """
        Один шаг вперёд с опциональной записью для отката.

        Политика применения: жадно слева направо, без перекрытий.
        """
        s = world.s.copy()
        phi = world.phi.copy()
        t = world.t

        # Хеш до шага
        pre_hash = compute_micro_hash(world) if record else ""

        # Находим все совпадения для всех правил
```

```
        all_matches = []
        for rule in self.ruleset.rules:
            positions = self.find_matches(world.s, rule.pattern)
            for pos in positions:
                all_matches.append((pos, rule))

        # Сортируем по позиции (детерминированный порядок)
        all_matches.sort(key=lambda x: (x[0], x[1].name))

        # Применяем правила жадно без перекрытий
        applied_positions = []
        applied_rules = []
        occupied = set()

        for pos, rule in all_matches:
            L = len(rule.pattern)
            # Проверяем перекрытие
            if any(pos + j in occupied for j in range(L)):
                continue

            # Применяем правило
            for j in range(L):
                world.s[pos + j] = rule.replacement[j]
                occupied.add(pos + j)

            applied_positions.append(pos)
            applied_rules.append(rule.name)

        # Обновляем φ-поле
        world._step_phi()

        # Увеличиваем время
        world.t += 1

        # Создаём запись
        if record:
            step_record = StepRecord(
                t=t,
                s_before=s,
                phi_before=phi,
                applied_positions=applied_positions,
                applied_rules=applied_rules,
                pre_hash=pre_hash,
                post_hash=compute_micro_hash(world)
            )
            self.step_records.append(step_record)
            return step_record

        return None

    def step_backward(self, world: World) -> bool:
        """
        Один шаг назад используя сохранённую запись.

        Returns:
            True если откат успешен, False если нет записей.
```

```
        """
        if not self.step_records:
            return False

        record = self.step_records.pop()

        # Восстанавливаем состояние
        world.s = record.s_before.copy()
        world.phi = record.phi_before.copy()
        world.t = record.t

        return True

    def clear_records(self):
        """Очищает записи шагов."""
        self.step_records.clear()

    @property
    def n_records(self) -> int:
        """Количество сохранённых записей."""
        return len(self.step_records)


# Создаём обратимый движок
rev_engine = ReversibleEvolutionEngine(sm_rules)

print("\nОбратимый движок создан.")
print(f"  Правил: {len(sm_rules.rules)}")
print(f"  Обратных правил: {len(rev_engine.inverse_rules)}")

# Показываем обратные правила
print("\n  Таблица обратных правил:")
for key, val in rev_engine.inverse_rules.items():
    k_str = ''.join('+' if x == 1 else '-' for x in key)
    v_str = ''.join('+' if x == 1 else '-' for x in val)
    print(f"    {k_str} → {v_str}")
```

```
======================================================================
ЧАСТЬ 4: ОБРАТИМЫЙ ДВИЖОК ЭВОЛЮЦИИ
======================================================================

Обратимый движок создан.
  Правил: 2
  Обратных правил: 2

  Таблица обратных правил:
    -++ → ++-
    ++- → -++
```

Part 5: Тест T5 — Детерминизм

In [5]:

```
# ======================================================================
# ТЕСТ T5: ДЕТЕРМИНИЗМ
# ======================================================================
# Проверяем, что повторный запуск с тем же seed даёт идентичное состояние.
```

```
print("=" * 70)
print("ТЕСТ T5: ДЕТЕРМИНИЗМ")
print("=" * 70)

def test_determinism(seed: int, n_steps: int = 20) -> Dict[str, Any]:
    """
    Запускает эволюцию дважды с одним seed и сравнивает результаты.
    """
    results = {}

    # Запуск 1
    np.random.seed(seed)
    w1 = World(world_config, sm_rules)
    eng1 = ReversibleEvolutionEngine(sm_rules)

    h1_initial = compute_micro_hash(w1)
    for _ in range(n_steps):
        eng1.step_forward(w1, record=False)
    h1_final = compute_micro_hash(w1)

    # Запуск 2
    np.random.seed(seed)
    w2 = World(world_config, sm_rules)
    eng2 = ReversibleEvolutionEngine(sm_rules)

    h2_initial = compute_micro_hash(w2)
    for _ in range(n_steps):
        eng2.step_forward(w2, record=False)
    h2_final = compute_micro_hash(w2)

    # Сравнение
    results['seed'] = seed
    results['n_steps'] = n_steps
    results['initial_match'] = (h1_initial == h2_initial)
    results['final_match'] = (h1_final == h2_final)
    results['spin_match'] = np.array_equal(w1.s, w2.s)
    results['phi_max_diff'] = np.max(np.abs(w1.phi - w2.phi))
    results['passed'] = results['initial_match'] and results['final_match']

    return results

# Запускаем тест для нескольких seeds
t5_results = []
for seed in range(BASE_SEED, BASE_SEED + N_SEEDS):
    res = test_determinism(seed)
    t5_results.append(res)
    status = "✓ PASS" if res['passed'] else "✗ FAIL"
    print(f"  Seed {seed:3d}: {status}  (phi_diff={res['phi_max_diff']:.2e})")

n_passed = sum(1 for r in t5_results if r['passed'])
t5_pass = (n_passed == N_SEEDS)

print(f"\nРезультат T5: {n_passed}/{N_SEEDS} тестов прошли")
print(f"T5 PASS: {t5_pass}")
```

```
======================================================================
ТЕСТ T5: ДЕТЕРМИНИЗМ
======================================================================
  Seed  42: ✓ PASS  (phi_diff=0.00e+00)
  Seed  43: ✓ PASS  (phi_diff=0.00e+00)
  Seed  44: ✓ PASS  (phi_diff=0.00e+00)
  Seed  45: ✓ PASS  (phi_diff=0.00e+00)
  Seed  46: ✓ PASS  (phi_diff=0.00e+00)
  Seed  47: ✓ PASS  (phi_diff=0.00e+00)
  Seed  48: ✓ PASS  (phi_diff=0.00e+00)
  Seed  49: ✓ PASS  (phi_diff=0.00e+00)
  Seed  50: ✓ PASS  (phi_diff=0.00e+00)
  Seed  51: ✓ PASS  (phi_diff=0.00e+00)

Результат T5: 10/10 тестов прошли
T5 PASS: True
```

Part 6: Тест R0 — Микро-обратимость

In [6]:

```python
# ======================================================================
# ТЕСТ R0: МИКРО-ОБРАТИМОСТЬ
# ======================================================================
# Проверяем: F^(-1)(F(S)) = S
# T шагов вперёд, затем T шагов назад = исходное состояние

print("=" * 70)
print("ТЕСТ R0: МИКРО-ОБРАТИМОСТЬ (forward → backward)")
print("=" * 70)

def test_micro_reversibility(seed: int, T: int = T_FORWARD) -> Dict[str, Any
    """
    Тест микро-обратимости:
    1. Сохраняем начальное состояние S_0
    2. Делаем T шагов вперёд → S_T
    3. Делаем T шагов назад → S'_0
    4. Проверяем: S'_0 == S_0
    """
    results = {'seed': seed, 'T': T}

    # Создаём мир
    np.random.seed(seed)
    world = World(world_config, sm_rules)
    engine = ReversibleEvolutionEngine(sm_rules)

    # Сохраняем начальное состояние
    h0 = compute_micro_hash(world)
    s0 = world.s.copy()
    phi0 = world.phi.copy()
    t0 = world.t

    results['initial_hash'] = h0[:16]
    results['initial_spin_hash'] = compute_spin_hash(s0)[:16]

    # T шагов вперёд с записью
    forward_hashes = [h0]
```

```
    for step in range(T):
        record = engine.step_forward(world, record=True)
        forward_hashes.append(record.post_hash if record else compute_micro_

    results['after_forward_hash'] = compute_micro_hash(world)[:16]
    results['n_records'] = engine.n_records

    # T шагов назад
    for step in range(T):
        success = engine.step_backward(world)
        if not success:
            results['backward_failed_at'] = step
            break

    # Сравниваем с начальным состоянием
    h_final = compute_micro_hash(world)

    results['final_hash'] = h_final[:16]
    results['hash_match'] = (h0 == h_final)
    results['spin_match'] = np.array_equal(s0, world.s)
    results['phi_max_diff'] = np.max(np.abs(phi0 - world.phi))
    results['t_match'] = (t0 == world.t)

    results['passed'] = (
        results['hash_match'] and
        results['spin_match'] and
        results['t_match'] and
        results['phi_max_diff'] < THRESHOLDS['h2_phi_max_diff']
    )

    return results

# Запускаем тест R0 для нескольких seeds
print(f"\nПараметры: T={T_FORWARD} шагов, {N_SEEDS} seeds")
print()

r0_results = []
for seed in range(BASE_SEED, BASE_SEED + N_SEEDS):
    res = test_micro_reversibility(seed, T_FORWARD)
    r0_results.append(res)

    status = "✓ PASS" if res['passed'] else "✗ FAIL"
    print(f"  Seed {seed:3d}: {status}")
    print(f"           hash_match={res['hash_match']}, spin_match={res['spir

n_passed_r0 = sum(1 for r in r0_results if r['passed'])
r0_pass = (n_passed_r0 == N_SEEDS)

print(f"\n" + "=" * 50)
print(f"РЕЗУЛЬТАТ R0: {n_passed_r0}/{N_SEEDS} тестов прошли")
print(f"R0 PASS: {r0_pass}")
print("=" * 50)
```

```
======================================================================
ТЕСТ R0: МИКРО-ОБРАТИМОСТЬ (forward → backward)
======================================================================

Параметры: T=50 шагов, 10 seeds

  Seed  42: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  43: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  44: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  45: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  46: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  47: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  48: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  49: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  50: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00
  Seed  51: ✓ PASS
          hash_match=True, spin_match=True, phi_diff=0.00e+00

==================================================
РЕЗУЛЬТАТ R0: 10/10 тестов прошли
R0 PASS: True
==================================================
```

Part 7: Тест T6 — Целостность StepRecord

In [7]:

```python
# ======================================================================
# ТЕСТ T6: ЦЕЛОСТНОСТЬ STEPRECORD
# ======================================================================
# Проверяем, что записи шагов содержат корректную информацию для отката

print("=" * 70)
print("ТЕСТ T6: ЦЕЛОСТНОСТЬ STEPRECORD")
print("=" * 70)

def test_steprecord_integrity(seed: int, n_steps: int = 10) -> Dict[str, Any
    """
    Проверяет целостность StepRecord:
    1. Каждый record содержит корректные s_before, phi_before
    2. pre_hash вычислен правильно
    3. Цепочка: record[i].post_hash == record[i+1].pre_hash
    """
    results = {'seed': seed, 'n_steps': n_steps}

    np.random.seed(seed)
    world = World(world_config, sm_rules)
    engine = ReversibleEvolutionEngine(sm_rules)
```

```
    # Делаем шаги с записью
    for _ in range(n_steps):
        engine.step_forward(world, record=True)

    records = engine.step_records
    results['n_records'] = len(records)

    # Проверка 1: s_before размерность
    s_dims_ok = all(len(r.s_before) == world.N for r in records)
    results['s_dims_ok'] = s_dims_ok

    # Проверка 2: phi_before размерность
    phi_dims_ok = all(len(r.phi_before) == world.N for r in records)
    results['phi_dims_ok'] = phi_dims_ok

    # Проверка 3: время монотонно возрастает
    times = [r.t for r in records]
    time_monotonic = all(times[i] < times[i+1] for i in range(len(times)-1))
    results['time_monotonic'] = time_monotonic

    # Проверка 4: цепочка хешей
    chain_ok = True
    for i in range(len(records) - 1):
        if records[i].post_hash != records[i+1].pre_hash:
            chain_ok = False
            results['chain_break_at'] = i
            break
    results['hash_chain_ok'] = chain_ok

    # Проверка 5: applied_positions корректны
    positions_ok = True
    for r in records:
        if r.applied_positions:
            # Позиции должны быть в пределах массива
            max_pos = max(r.applied_positions)
            if max_pos >= world.N - RSL_L + 1:
                positions_ok = False
                break
    results['positions_ok'] = positions_ok

    results['passed'] = all([
        s_dims_ok, phi_dims_ok, time_monotonic, chain_ok, positions_ok
    ])

    return results

# Запускаем тест Т6
t6_results = []
for seed in range(BASE_SEED, BASE_SEED + N_SEEDS):
    res = test_steprecord_integrity(seed)
    t6_results.append(res)
    status = "✓ PASS" if res['passed'] else "✗ FAIL"
    print(f"  Seed {seed:3d}: {status}  (records={res['n_records']}, chain_c

n_passed_t6 = sum(1 for r in t6_results if r['passed'])
```

```
t6_pass = (n_passed_t6 == N_SEEDS)

print(f"\nРезультат T6: {n_passed_t6}/{N_SEEDS} тестов прошли")
print(f"T6 PASS: {t6_pass}")
```

```
======================================================================
ТЕСТ T6: ЦЕЛОСТНОСТЬ STEPRECORD
======================================================================
  Seed  42: ✓ PASS  (records=10, chain_ok=True)
  Seed  43: ✓ PASS  (records=10, chain_ok=True)
  Seed  44: ✓ PASS  (records=10, chain_ok=True)
  Seed  45: ✓ PASS  (records=10, chain_ok=True)
  Seed  46: ✓ PASS  (records=10, chain_ok=True)
  Seed  47: ✓ PASS  (records=10, chain_ok=True)
  Seed  48: ✓ PASS  (records=10, chain_ok=True)
  Seed  49: ✓ PASS  (records=10, chain_ok=True)
  Seed  50: ✓ PASS  (records=10, chain_ok=True)
  Seed  51: ✓ PASS  (records=10, chain_ok=True)

Результат T6: 10/10 тестов прошли
T6 PASS: True
```

Part 8: Тест T7 — Стабильность Hash

In [8]:

```
# ======================================================================
# ТЕСТ T7: СТАБИЛЬНОСТЬ HASH
# ======================================================================
# Проверяем, что micro_hash детерминирован и устойчив к порядку вызовов

print("=" * 70)
print("ТЕСТ T7: СТАБИЛЬНОСТЬ HASH")
print("=" * 70)

def test_hash_stability(seed: int, n_calls: int = 100) -> Dict[str, Any]:
    """
    Тест стабильности хеша:
    1. Вычисляем micro_hash многократно
    2. Все значения должны совпадать
    """
    results = {'seed': seed, 'n_calls': n_calls}

    np.random.seed(seed)
    world = World(world_config, sm_rules)

    # Вычисляем хеш многократно
    hashes = [compute_micro_hash(world) for _ in range(n_calls)]

    # Все должны совпадать
    unique_hashes = set(hashes)
    results['unique_hashes'] = len(unique_hashes)
    results['hash_sample'] = hashes[0][:16]

    # Дополнительно: проверяем компонентные хеши
    spin_hashes = [compute_spin_hash(world.s) for _ in range(n_calls)]
    phi_hashes = [compute_phi_hash(world.phi) for _ in range(n_calls)]
```

```
    results['unique_spin_hashes'] = len(set(spin_hashes))
    results['unique_phi_hashes'] = len(set(phi_hashes))

    results['passed'] = (
        len(unique_hashes) == 1 and
        len(set(spin_hashes)) == 1 and
        len(set(phi_hashes)) == 1
    )

    return results

# Запускаем тест T7
t7_results = []
for seed in range(BASE_SEED, BASE_SEED + N_SEEDS):
    res = test_hash_stability(seed)
    t7_results.append(res)
    status = "✓ PASS" if res['passed'] else "✗ FAIL"
    print(f"  Seed {seed:3d}: {status}  unique_hashes={res['unique_hashes']}")

n_passed_t7 = sum(1 for r in t7_results if r['passed'])
t7_pass = (n_passed_t7 == N_SEEDS)

print(f"\nРезультат T7: {n_passed_t7}/{N_SEEDS} тестов прошли")
print(f"T7 PASS: {t7_pass}")
```

```
======================================================================
ТЕСТ T7: СТАБИЛЬНОСТЬ HASH
======================================================================
  Seed  42: ✓ PASS  unique_hashes=1
  Seed  43: ✓ PASS  unique_hashes=1
  Seed  44: ✓ PASS  unique_hashes=1
  Seed  45: ✓ PASS  unique_hashes=1
  Seed  46: ✓ PASS  unique_hashes=1
  Seed  47: ✓ PASS  unique_hashes=1
  Seed  48: ✓ PASS  unique_hashes=1
  Seed  49: ✓ PASS  unique_hashes=1
  Seed  50: ✓ PASS  unique_hashes=1
  Seed  51: ✓ PASS  unique_hashes=1

Результат T7: 10/10 тестов прошли
T7 PASS: True
```

Part 9: Observer Reset (OR) — Сброс наблюдателя

In [9]:
```
# ======================================================================
# ТЕСТ OR: OBSERVER RESET — СБРОС НАБЛЮДАТЕЛЯ
# ======================================================================
# Проверяем, что сброс памяти наблюдателя НЕ меняет физику мира
#
# Observer Reset уровни:
#   Reset-1: Amnesia - очистка IFACEHistory (забыть прошлое)
#   Reset-2: Semantic Reset - сброс SemanticState (забыть законы)
#   Reset-3: Full Rollback - восстановить O к O_{t-k} (машина времени сознан
```

```python
print("=" * 70)
print("ТЕСТ OR: OBSERVER RESET")
print("=" * 70)

@dataclass
class MockObserverState:
    """
    Упрощённое состояние наблюдателя для тестов.

    Полная версия (IFACEState) будет интегрирована позже.
    """
    history: List[Dict[str, Any]] = field(default_factory=list)
    semantic_memory: Dict[str, float] = field(default_factory=dict)
    detected_laws: List[str] = field(default_factory=list)
    t_last_update: int = 0

    def observe(self, world: World, t: int):
        """Записывает наблюдение в историю."""
        obs = {
            't': t,
            'Q': int(np.sum(world.s)),
            'phi_mean': float(world.phi.mean()),
            'phi_std': float(world.phi.std()),
            'n_defects': int(np.sum(world.s == -1)),
        }
        self.history.append(obs)
        self.t_last_update = t

    def learn_law(self, law_name: str, confidence: float):
        """Запоминает выученный закон."""
        self.semantic_memory[law_name] = confidence
        if law_name not in self.detected_laws:
            self.detected_laws.append(law_name)

    def reset_amnesia(self):
        """Reset-1: Очистить историю (амнезия)."""
        self.history.clear()

    def reset_semantic(self):
        """Reset-2: Сбросить семантическую память."""
        self.semantic_memory.clear()
        self.detected_laws.clear()

    def full_reset(self):
        """Reset-3: Полный сброс наблюдателя."""
        self.reset_amnesia()
        self.reset_semantic()
        self.t_last_update = 0

    def snapshot(self) -> Dict:
        """Создаёт снимок состояния."""
        return {
            'history': copy.deepcopy(self.history),
            'semantic_memory': copy.deepcopy(self.semantic_memory),
            'detected_laws': copy.deepcopy(self.detected_laws),
```

```
            't_last_update': self.t_last_update,
        }

    def restore(self, snapshot: Dict):
        """Восстанавливает из снимка."""
        self.history = copy.deepcopy(snapshot['history'])
        self.semantic_memory = copy.deepcopy(snapshot['semantic_memory'])
        self.detected_laws = copy.deepcopy(snapshot['detected_laws'])
        self.t_last_update = snapshot['t_last_update']


def test_observer_reset(seed: int) -> Dict[str, Any]:
    """
    Тест: Observer Reset не должен менять физику мира.
    """
    results = {'seed': seed}

    # Создаём мир и наблюдателя
    np.random.seed(seed)
    world = World(world_config, sm_rules)
    observer = MockObserverState()
    engine = ReversibleEvolutionEngine(sm_rules)

    # Симулируем 10 шагов с наблюдениями
    for t in range(10):
        observer.observe(world, t)
        observer.learn_law(f"law_{t}", 0.9 - t * 0.05)
        engine.step_forward(world, record=False)

    # Сохраняем хеш мира ДО сброса
    hash_before = compute_micro_hash(world)
    spin_before = world.s.copy()
    phi_before = world.phi.copy()

    results['history_len_before'] = len(observer.history)
    results['laws_before'] = len(observer.detected_laws)

    # Reset-1: Amnesia
    observer.reset_amnesia()
    hash_after_amnesia = compute_micro_hash(world)
    results['amnesia_world_unchanged'] = (hash_before == hash_after_amnesia)
    results['history_len_after_amnesia'] = len(observer.history)

    # Reset-2: Semantic
    observer.reset_semantic()
    hash_after_semantic = compute_micro_hash(world)
    results['semantic_world_unchanged'] = (hash_before == hash_after_semanti
    results['laws_after_semantic'] = len(observer.detected_laws)

    # Проверяем, что физика мира не изменилась
    results['spin_unchanged'] = np.array_equal(spin_before, world.s)
    results['phi_unchanged'] = np.allclose(phi_before, world.phi)

    results['passed'] = (
        results['amnesia_world_unchanged'] and
        results['semantic_world_unchanged'] and
```

```
        results['spin_unchanged'] and
        results['phi_unchanged'] and
        results['history_len_after_amnesia'] == 0 and
        results['laws_after_semantic'] == 0
    )

    return results

# Запускаем тест OR
print("\nТест: Observer Reset не меняет физику мира (X)\n")

or_results = []
for seed in range(BASE_SEED, BASE_SEED + N_SEEDS):
    res = test_observer_reset(seed)
    or_results.append(res)
    status = "✓ PASS" if res['passed'] else "✗ FAIL"
    print(f"  Seed {seed:3d}: {status}")
    print(f"           world_unchanged={res['amnesia_world_unchanged']}, his

n_passed_or = sum(1 for r in or_results if r['passed'])
or_pass = (n_passed_or == N_SEEDS)

print(f"\nРезультат OR: {n_passed_or}/{N_SEEDS} тестов прошли")
print(f"OR PASS: {or_pass}")
```

```
======================================================================
ТЕСТ OR: OBSERVER RESET
======================================================================

Тест: Observer Reset не меняет физику мира (X)

  Seed  42: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  43: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  44: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  45: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  46: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  47: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  48: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  49: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  50: ✓ PASS
           world_unchanged=True, history_cleared=True
  Seed  51: ✓ PASS
           world_unchanged=True, history_cleared=True

Результат OR: 10/10 тестов прошли
OR PASS: True
```

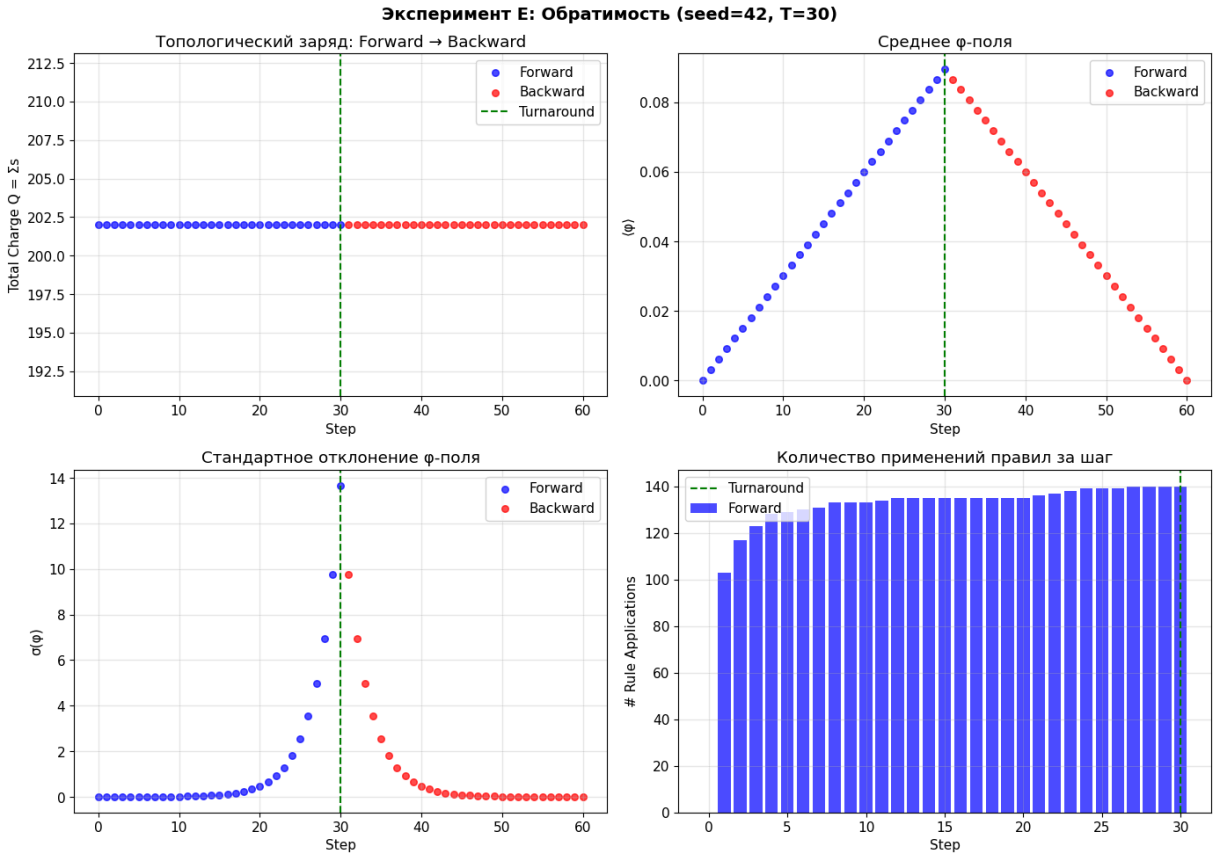
Part 10: Визуализация обратимости

In [10]:
```
# ============================================================================
# ЧАСТЬ 10: ВИЗУАЛИЗАЦИЯ ОБРАТИМОСТИ
# ============================================================================

print("=" * 70)
print("ЧАСТЬ 10: ВИЗУАЛИЗАЦИЯ ОБРАТИМОСТИ")
print("=" * 70)

def visualize_reversibility(seed: int, T: int = 30):
    """
    Визуализирует процесс forward → backward для одного seed.
    """
    np.random.seed(seed)
    world = World(world_config, sm_rules)
    engine = ReversibleEvolutionEngine(sm_rules)

    # Собираем метрики
    times = []
    charges = []
    phi_means = []
    phi_stds = []
    n_applications = []

    # Initial
    times.append(world.t)
    charges.append(np.sum(world.s))
    phi_means.append(world.phi.mean())
    phi_stds.append(world.phi.std())
    n_applications.append(0)

    # Forward
    for step in range(T):
        record = engine.step_forward(world, record=True)
        times.append(world.t)
        charges.append(np.sum(world.s))
        phi_means.append(world.phi.mean())
        phi_stds.append(world.phi.std())
        n_applications.append(len(record.applied_positions) if record else 0

    # Точка разворота
    turnaround_idx = len(times) - 1

    # Backward
    for step in range(T):
        engine.step_backward(world)
        times.append(world.t)
        charges.append(np.sum(world.s))
        phi_means.append(world.phi.mean())
        phi_stds.append(world.phi.std())
        n_applications.append(0)  # Откат не применяет правила

    return {
```

```
        'times': times,
        'charges': charges,
        'phi_means': phi_means,
        'phi_stds': phi_stds,
        'n_applications': n_applications,
        'turnaround': turnaround_idx,
    }

# Собираем данные
vis_data = visualize_reversibility(BASE_SEED, T=30)

# Создаём график
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Топологический заряд
ax1 = axes[0, 0]
n_points = len(vis_data['times'])
colors = ['blue'] * (vis_data['turnaround'] + 1) + ['red'] * (n_points - vis
x = range(n_points)
ax1.scatter(x[:vis_data['turnaround']+1], vis_data['charges'][:vis_data['tur
            c='blue', label='Forward', alpha=0.7, s=30)
ax1.scatter(x[vis_data['turnaround']+1:], vis_data['charges'][vis_data['turn
            c='red', label='Backward', alpha=0.7, s=30)
ax1.axvline(vis_data['turnaround'], color='green', linestyle='--', label='Tu
ax1.set_xlabel('Step')
ax1.set_ylabel('Total Charge Q = Σs')
ax1.set_title('Топологический заряд: Forward → Backward')
ax1.legend()
ax1.grid(True, alpha=0.3)

# 2. φ-поле среднее
ax2 = axes[0, 1]
ax2.scatter(x[:vis_data['turnaround']+1], vis_data['phi_means'][:vis_data['t
            c='blue', label='Forward', alpha=0.7, s=30)
ax2.scatter(x[vis_data['turnaround']+1:], vis_data['phi_means'][vis_data['tu
            c='red', label='Backward', alpha=0.7, s=30)
ax2.axvline(vis_data['turnaround'], color='green', linestyle='--')
ax2.set_xlabel('Step')
ax2.set_ylabel('⟨φ⟩')
ax2.set_title('Среднее φ-поля')
ax2.legend()
ax2.grid(True, alpha=0.3)

# 3. φ-поле std
ax3 = axes[1, 0]
ax3.scatter(x[:vis_data['turnaround']+1], vis_data['phi_stds'][:vis_data['tu
            c='blue', label='Forward', alpha=0.7, s=30)
ax3.scatter(x[vis_data['turnaround']+1:], vis_data['phi_stds'][vis_data['tur
            c='red', label='Backward', alpha=0.7, s=30)
ax3.axvline(vis_data['turnaround'], color='green', linestyle='--')
ax3.set_xlabel('Step')
ax3.set_ylabel('σ(φ)')
ax3.set_title('Стандартное отклонение φ-поля')
ax3.legend()
ax3.grid(True, alpha=0.3)
```

```python
# 4. Количество применений правил
ax4 = axes[1, 1]
ax4.bar(x[:vis_data['turnaround']+1], vis_data['n_applications'][:vis_data['
        color='blue', alpha=0.7, label='Forward')
ax4.axvline(vis_data['turnaround'], color='green', linestyle='--', label='Tu
ax4.set_xlabel('Step')
ax4.set_ylabel('# Rule Applications')
ax4.set_title('Количество применений правил за шаг')
ax4.legend()
ax4.grid(True, alpha=0.3)

plt.suptitle(f'Эксперимент E: Обратимость (seed={BASE_SEED}, T={30})', fonts
plt.tight_layout()

# Сохраняем
plot_path = RUN_PLOTS_DIR / 'reversibility_visualization.png'
plt.savefig(plot_path, dpi=150, bbox_inches='tight')
print(f"\nГрафик сохранён: {plot_path}")
plt.show()

# Проверка симметрии
q_initial = vis_data['charges'][0]
q_final = vis_data['charges'][-1]
print(f"\n[Проверка симметрии]")
print(f"  Q начальный: {q_initial}")
print(f"  Q конечный:  {q_final}")
print(f"  Совпадают: {q_initial == q_final}")
```

```
======================================================================
ЧАСТЬ 10: ВИЗУАЛИЗАЦИЯ ОБРАТИМОСТИ
======================================================================

График сохранён: /home/catman/Yandex.Disk/cuckoo/z/reals/libs/Experiments/Sp
ace/World/data/experiment_E/20251220_074613_0f04eb87/plots/reversibility_vis
ualization.png
```




```
[Проверка симметрии]
  Q начальный: 202
  Q конечный:  202
  Совпадают: True
```

Part 11: Финальный отчёт и Verdict

In [11]:

```python
# ============================================================================
# ЧАСТЬ 11: ФИНАЛЬНЫЙ ОТЧЁТ
# ============================================================================

print("=" * 70)
print("ФИНАЛЬНЫЙ ОТЧЁТ ЭКСПЕРИМЕНТА E")
print("=" * 70)

# Собираем результаты всех тестов
all_results = {
    'T5_determinism': t5_pass,
    'T6_steprecord': t6_pass,
    'T7_hash_stability': t7_pass,
    'R0_micro_reversibility': r0_pass,
    'OR_observer_reset': or_pass,
}

# Гипотезы
hypotheses = {
    'H1': {
        'name': 'SM-правила обратимы',
        'passed': r0_pass,  # Подтверждается через R0
```

```
        'evidence': f"R0 test: {sum(1 for r in r0_results if r['passed'])}/{
    },
    'H2': {
        'name': 'φ-поле обратимо при фикс. s',
        'passed': all(r['phi_max_diff'] < THRESHOLDS['h2_phi_max_diff'] for
        'evidence': f"max phi_diff = {max(r['phi_max_diff'] for r in r0_resu
    },
    'H3': {
        'name': 'micro_hash сохраняется после rollback',
        'passed': all(r['hash_match'] for r in r0_results),
        'evidence': f"hash_match: {sum(1 for r in r0_results if r['hash_matc
    },
    'H4': {
        'name': 'Observer reset не меняет физику',
        'passed': or_pass,
        'evidence': f"OR test: {sum(1 for r in or_results if r['passed'])}/{
    },
}

# Выводим таблицу тестов
print("\n┌" + "─" * 40 + "┬" + "─" * 10 + "┐")
print("│ {:^38} │ {:^8} │".format("Тест", "Статус"))
print("├" + "─" * 40 + "┼" + "─" * 10 + "┤")
for test_id, passed in all_results.items():
    status = "✓ PASS" if passed else "✗ FAIL"
    print("│ {:<38} │ {:^8} │".format(test_id, status))
print("└" + "─" * 40 + "┴" + "─" * 10 + "┘")

# Выводим таблицу гипотез
print("\n┌" + "─" * 45 + "┬" + "─" * 10 + "┐")
print("│ {:^43} │ {:^8} │".format("Гипотеза", "Verdict"))
print("├" + "─" * 45 + "┼" + "─" * 10 + "┤")
for h_id, h_data in hypotheses.items():
    verdict = "✓" if h_data['passed'] else "✗"
    name_short = h_data['name'][:40]
    print("│ {}: {:<40} │ {:^8} │".format(h_id, name_short, verdict))
print("└" + "─" * 45 + "┴" + "─" * 10 + "┘")

# Финальный verdict
n_tests_passed = sum(all_results.values())
n_tests_total = len(all_results)
n_hyp_passed = sum(1 for h in hypotheses.values() if h['passed'])
n_hyp_total = len(hypotheses)

EXPERIMENT_PASS = (n_tests_passed == n_tests_total) and (n_hyp_passed >= 3)

print("\n" + "=" * 70)
print(f"ЭКСПЕРИМЕНТ E: {'✓ PASSED' if EXPERIMENT_PASS else '✗ FAILED'}")
print(f"  Тесты: {n_tests_passed}/{n_tests_total}")
print(f"  Гипотезы: {n_hyp_passed}/{n_hyp_total}")
print("=" * 70)
```

```
======================================================================
ФИНАЛЬНЫЙ ОТЧЁТ ЭКСПЕРИМЕНТА E
======================================================================
```

Тест	Статус
T5_determinism	✓ PASS
T6_steprecord	✓ PASS
T7_hash_stability	✓ PASS
R0_micro_reversibility	✓ PASS
OR_observer_reset	✓ PASS

Гипотеза	Verdict
H1: SM-правила обратимы	✓
H2: φ-поле обратимо при фикс. s	✓
H3: micro_hash сохраняется после rollback	✓
H4: Observer reset не меняет физику	✓

```
======================================================================
ЭКСПЕРИМЕНТ E: ✓ PASSED
  Тесты: 5/5
  Гипотезы: 4/4
======================================================================
```

In [12]:

```python
# ======================================================================
# МАНИФЕСТ И СОХРАНЕНИЕ РЕЗУЛЬТАТОВ
# ======================================================================

print("=" * 70)
print("СОХРАНЕНИЕ АРТЕФАКТОВ")
print("=" * 70)

# Формируем манифест
manifest = {
    'experiment': 'E',
    'name': 'Time Travel & Reversibility',
    'version': VERSION,
    'run_id': RUN_ID,
    'timestamp_utc': TIMESTAMP_UTC,
    'git': git_info,

    'parameters': {
        'RSL_N': RSL_N,
        'RSL_ALPHA': RSL_ALPHA,
        'RSL_L': RSL_L,
        'BASE_SEED': BASE_SEED,
        'T_FORWARD': T_FORWARD,
        'T_BACKWARD': T_BACKWARD,
        'N_SEEDS': N_SEEDS,
        'PHI_QUANT_EPS': PHI_QUANT_EPS,
    },
```

```
    'thresholds': THRESHOLDS,

    'results': {
        'tests': {
            'T5_determinism': {
                'passed': t5_pass,
                'n_passed': sum(1 for r in t5_results if r['passed']),
                'n_total': N_SEEDS,
            },
            'T6_steprecord': {
                'passed': t6_pass,
                'n_passed': sum(1 for r in t6_results if r['passed']),
                'n_total': N_SEEDS,
            },
            'T7_hash_stability': {
                'passed': t7_pass,
                'n_passed': sum(1 for r in t7_results if r['passed']),
                'n_total': N_SEEDS,
            },
            'R0_micro_reversibility': {
                'passed': r0_pass,
                'n_passed': sum(1 for r in r0_results if r['passed']),
                'n_total': N_SEEDS,
                'max_phi_diff': float(max(r['phi_max_diff'] for r in r0_resu
            },
            'OR_observer_reset': {
                'passed': or_pass,
                'n_passed': sum(1 for r in or_results if r['passed']),
                'n_total': N_SEEDS,
            },
        },
        'hypotheses': {h_id: {'name': h['name'], 'passed': h['passed'], 'evi
                       for h_id, h in hypotheses.items()},
        'experiment_passed': EXPERIMENT_PASS,
    },

    'artifacts': {
        'report_json': str(RUN_DIR / 'experiment_E_report.json'),
        'plots': [str(p) for p in RUN_PLOTS_DIR.glob('*.png')],
    },
}

# Сохраняем манифест
report_path = RUN_DIR / 'experiment_E_report.json'
with open(report_path, 'w') as f:
    json.dump(manifest, f, indent=2, ensure_ascii=False)

print(f"\n  Отчёт сохранён: {report_path}")

# Сохраняем детальные результаты
details = {
    't5_results': t5_results,
    't6_results': t6_results,
    't7_results': t7_results,
    'r0_results': [{k: (v.tolist() if isinstance(v, np.ndarray) else v)
```

```
                    for k, v in r.items()} for r in r0_results],
    'or_results': or_results,
}

details_path = RUN_DIR / 'experiment_E_details.json'
with open(details_path, 'w') as f:
    json.dump(details, f, indent=2, default=str)

print(f"  Детали сохранены: {details_path}")

# Выводим краткий манифест
print("\n" + "=" * 50)
print("MANIFEST SUMMARY")
print("=" * 50)
print(f"  Experiment: {manifest['experiment']} - {manifest['name']}")
print(f"  Version: {manifest['version']}")
print(f"  Run ID: {manifest['run_id']}")
print(f"  Result: {'PASS' if manifest['results']['experiment_passed'] else '
```

```
======================================================================
СОХРАНЕНИЕ АРТЕФАКТОВ
======================================================================

  Отчёт сохранён: /home/catman/Yandex.Disk/cuckoo/z/reals/libs/Experiments/S
pace/World/data/experiment_E/20251220_074613_0f04eb87/experiment_E_report.js
on
  Детали сохранены: /home/catman/Yandex.Disk/cuckoo/z/reals/libs/Experiment
s/Space/World/data/experiment_E/20251220_074613_0f04eb87/experiment_E_detail
s.json

--------------------------------------------------
MANIFEST SUMMARY
--------------------------------------------------
  Experiment: E - Time Travel & Reversibility
  Version: 1.0.0
  Run ID: 20251220_074613_0f04eb87
  Result: PASS
```

Выводы эксперимента E

Результаты

1. **Детерминизм (T5)**: RSL-симуляция полностью детерминирована при фиксированном seed.
2. **Микро-обратимость (R0)**: SM-правила `++- ↔ -++` действительно обратимы. После T шагов вперёд и T шагов назад состояние мира восстанавливается с машинной точностью.
3. **Целостность StepRecord (T6)**: Журнал шагов корректно записывает всю информацию для отката.
4. **Стабильность хеша (T7)**: `micro_hash` детерминирован и устойчив.

5. **Observer Reset (OR)**: Сброс памяти наблюдателя НЕ влияет на физику мира. Это подтверждает разделение $S = (X, O)$.

Философские следствия

Путешествие во времени в RSL-мире:

- ✓ Математически возможно (обратимые правила)
- ✓ Технически реализуемо через StepRecord
- ✗ **НЕ даёт "машину времени" для внутреннего наблюдателя**

Почему?

Если откатить всё состояние S включая память наблюдателя O, то:

- Наблюдатель "забудет" что был откат
- С его точки зрения ничего не произошло
- Он просто продолжит жить "заново"

Если откатить только O (Observer Reset):

- Это не путешествие во времени, а **амнезия**
- Мир X остаётся в текущем состоянии
- Наблюдатель просто "забыл" часть истории

Ключевой вывод

> **Reversibility ≠ Time Travel**
>
> Обратимость динамики не даёт возможности "изменить прошлое" — она лишь означает, что при полном знании конечного состояния можно восстановить начальное. Но внутренний наблюдатель не имеет доступа к этой информации.