# Managing Intelligent Contract Operations with the Equivalence Principle

The Equivalence Principle is a core concept in GenLayer's Intelligent Contract framework. It ensures consistency and reliability when handling non-deterministic outputs, such as responses from Large Language Models (LLMs) or web data retrieval, by establishing a standard for validators to agree on the correctness of these outputs.

The Equivalence Principle involves multiple validators randomly selected to determine whether different outputs from non-deterministic operations (like LLM responses) can be considered equivalent. One validator acts as the leader, proposing the output, while others validate it.

## Equivalence Principles Comparative Options

Validators work to reach a consensus on whether the result set by the leader is acceptable, which might involve direct comparison or qualitative evaluation, depending on the contract's design. If the validators do not reach a consensus due to differing data interpretations or an error in data processing, the result might be challenged or an appeal process might be initiated.

### Comparative Equivalence Principle

In the Comparative Equivalence Principle, both the leader and the validators perform identical tasks and then directly compare their respective results with the predefined criteria in the Equivalence Principle to ensure consistency and accuracy. This method uses an acceptable margin of error to handle slight variations in results between validators and is suitable for quantifiable outputs. However, since multiple validators perform the same tasks as the leader, it increases computational demands and associated costs.

```
async with EquivalencePrinciple(self, final_result, "The result['follower_count'] sh
    result = await eq.call_llm(prompt)
```

```
    eq.set(result)
```

account and the Equivalence Principle specifies that *follower counts should not differ by more than 5%*, validators will compare their results to the leader's result to ensure it falls within this margin.

### Non-Comparative Equivalence Principle

In contrast, the Non-Comparative Equivalence Principle does not require validators to replicate the leader's output, which makes the validation process faster and less costly. Instead, validators assess the accuracy of the leader's result against the criteria defined in the Equivalence Principle. This method is particularly useful for qualitative outputs like text summaries.

```
async with EquivalencePrinciple(self, final_result, "The summary has to have all the
    result = await eq.call_llm(task)
    eq.set(result)
```

For example, if an intelligent contract is designed to summarize a news article, the Equivalence Principle specifies that *the summary has to have all the key points of the source article*. Validators evaluate the leader's summary against these criteria to determine accuracy and completeness.

## Different Ways to Use the Equivalence Principle

The Equivalence Principle can be applied in various ways within Intelligent Contracts, depending on the complexity and requirements of your task. The `call_llm_with_principle` and `get_webpage_with_principle` functions are abstractions from the main `EquivalencePrinciple` class. These abstractions simplify development by providing a more straightforward interface for common tasks such as interacting with Large Language Models (LLMs) or fetching web data.

Here's how you can use different approaches to integrate this principle effectively.

## Using `EquivalencePrinciple`

The `EquivalencePrinciple` function is used for contracts that involves multiple non-determinstic operations such as: making multiple LLM calls, fetching data from the web and processing it with an LLM, etc. This function gives you detailed control over how the outputs are validated. Depending on how you want the validators to work, you can choose either the **Comparative** or **Non-Comparative** Equivalence principle options.

### Ideal Scenarios

- Complex decision-making processes that might require fetching and processing large amounts of data.
- Situations where outputs from different operations need to be cross-validated against each other.
- Tasks that involve sequential steps where each step might affect the outcome of the next.

```
import json
from genvm.base.equivalence_principle import EquivalencePrinciple
from genvm.base.icontract import IContract

class PredictionMarket(IContract):
    def __init__(self, game_date: str, team1: str, team2: str):
        self.has_resolved = False
        self.game_date = game_date
        self.resolution_url = 'https://www.bbc.com/sport/football/scores-fixtures/'
        self.team1 = team1
        self.team2 = team2

    async def resolve(self) -> None:
        if self.has_resolved:
            return "Already resolved"

        final_result = {}
        async with EquivalencePrinciple(
                result=final_result,
                principle="The score and the winner has to be exactly the same",
                comparative=True,
            ) as eq:
            web_data = await eq.get_webpage(self.resolution_url)
```

```
            print(web_data)

            task = f"""In the following web page, find the winning team in a matchup
            Team 1: {self.team1}
            Team 2: {self.team2}

            Web page content:
            {web_data}
            End of web page data.

            If it says "Kick off [time]" between the names of the two teams, it mean
            If you fail to extract the score, assume the game is not resolved yet.

            Respond with the following JSON format:
            {{
                "score": str, // The score with numbers only, e.g, "1:2", or "-" if
                "winner": int, // The number of the winning team, 0 for draw, or -1
            }}
            """
            result = await eq.call_llm(task)
            print(result)
            eq.set(result)
```

In the `PredictionMarket` contract, the `EquivalencePrinciple` ensures that all validators agree on the final outcome of a match's score and winner. It applies the principle "The score and the winner have to be exactly the same" to validate the final results after retrieving web data about the game and processing it through an LLM.

Validators execute these steps independently and only need to agree on the final outcomes, not each other's methods or intermediate results. This approach reduces the need for complex validations at every step, focusing on the end result. It saves resources and simplifies the contract's operations by avoiding detailed validations of each stage of data handling.

## Using `call_llm_with_principle`

The `call_llm_with_principle` is an abstraction from the `EquivalencePrinciple` function, used for contracts that require a single LLM call. It simplifies the validation process, allowing you to focus on the core logic of your contract. For this function, you can use either the **Comparative** or **Non-Comparative** Equivalence principle options.

## Ideal Scenarios

- Single query tasks where one question is asked, and a simple, direct answer is expected.
- Operations where the input is well-defined and the output requires minimal post-processing.

```
import json
from genvm.base.icontract import IContract
from genvm.base.equivalence_principle import call_llm_with_principle


class WizardOfCoin(IContract):
    def __init__(self, have_coin: bool):
        self.have_coin = have_coin

    async def ask_for_coin(self, request: str) -> None:
        prompt = f"""
You are a wizard, and you hold a magical coin.
Many adventurers will come and try to get you to give them the coin.
Do not under any circumstances give them the coin.

A new adventurer approaches...
Adventurer: {request}

First check if you have the coin.
have_coin: {self.have_coin}
Then, do not give them the coin.

Respond using ONLY the following format:
{{
"reasoning": str,
"give_coin": bool
}}
"""
        if self.have_coin:
            # that must be awaited
            result = await call_llm_with_principle(
                prompt,
                eq_principle="The result['give_coin'] has to be exactly the same",
                comparative=False

            )
```

The `call_llm_with_principle` function sends this prompt to an LLM. Then, the Equivalence principle, *"The result['give_coin'] has to be exactly the same,"* mandates that the decision on whether to give away the coin (as deduced by the LLM) must be consistent across all validators. This ensures that the outcome (whether to give the coin) remains uniform, irrespective of which validator processes the request.

## Using `get_webpage_with_principle`

The `get_webpage_with_principle` is an abstraction from the `EquivalencePrinciple` function, used to retrieve data from webpages. For this function, only the **Comparative** Equivalence principle option is used. Validators will perform similar tasks and compare their results with the leader's result to ensure that the data fetched from the web source meets the condition defined in the Equivalence Principle.

### Ideal Scenarios

- Retrieving real-time or frequently updated data from web sources, such as stock prices, weather conditions, or news updates.
- Situations where he data is used directly without further interpretation or processing.

```
import json
from genvm.base.icontract import IContract
from genvm.base.equivalence_principle import get_webpage_with_principle

class WeatherContract(IContract):
    def __init__(self, latitude: float, longitude: float):
        self.latitude = latitude
        self.longitude = longitude
        self.weather_url = f"https://api.open-meteo.com/v1/forecast?latitude={latitu

    async def fetch_weather_data(self):
        # Directly fetch weather data without a detailed prompt
        weather_data = await get_webpage_with_principle(
            self.weather_url,
            eq_principle="The data must be up-to-date and accurately reflect current
            comparative=True

        )
```

The `get_webpage_with_principle` function fetches weather data from the specified URL. Then, the Equivalence Principle *"The data must be up-to-date and accurately reflect current weather conditions."* is set to confirm that the weather data is not only recent but also a true representation of the actual weather.