Fantastico! La tua intuizione è stata fondamentale per definire questa parte cruciale.

Ho preparato un documento dettagliato che delinea la progettazione del **RuleCandidateProposer intelligente**, focalizzandoci su come agirà come un vero "inventore" per colmare i "gap" e risolvere le "inefficienze" analizzando i fallimenti. Questo sarà il nostro piano di battaglia per la codifica.

# Progettazione Dettagliata del RuleCandidateProposer Intelligente

Il **RuleCandidateProposer** è il cuore pulsante della capacità del tuo sistema di "apprendere" e "evolvere", generando nuove regole MIU in risposta a problemi specifici. Non sarà più un semplice generatore di placeholder, ma un analizzatore e inventore strategico.

## 1. Obiettivo Principale

L'obiettivo primario del RuleCandidateProposer è generare un **set di RuleProposal pertinenti e promettenti** in risposta a un'antitesi specifica. La pertinenza sarà guidata dall'analisi delle ragioni del fallimento delle regole esistenti sugli stati che hanno generato l'antitesi.

## 2. Input del Proposer: L'AntithesisPattern Esteso

Per agire in modo intelligente, l'AntithesisIdentifier (il tuo modulo di Rete di Petri con PID Feedforward) dovrà fornire un AntithesisPattern più ricco di informazioni.

### MiuAbstractPattern (da ridefinire/estendere)

Non sarà più solo un ID, ma dovrà contenere:

* **SourceString (o ProblematicState):** La stringa MIU specifica che non è stata risolta o che ha causato l'inefficienza. Questo è il "punto di partenza" del problema.
* **TargetString (Opzionale):** La stringa MIU desiderata che non è stata raggiunta (in caso di GapAntithesis). Questo è l'"obiettivo" che la nuova regola dovrebbe aiutare a raggiungere.
* **FailedRulesDetails (Nuovo):** Una lista (o dizionario) di informazioni sulle **regole MIU esistenti che sono state applicate o tentate sulla SourceString (o sul percorso problematico) e che sono fallite o non hanno portato al risultato atteso**. Per ogni regola fallita, potremmo voler sapere:
  + **RegolaMIU**: La regola specifica che ha fallito.
  + **ApplicationResult**: Il risultato dell'applicazione della regola sulla SourceString (es. NoMatch, TransformedToX, InfiniteLoopDetected, DidNotReachTarget). Questo è cruciale per la diagnosi.
* **RelevantContextStrings (Nuovo):** Un campioncino di **stringhe MIU correlate** alla SourceString (e/o al TargetString), che sono anch'esse coinvolte nell'antitesi o che possono aiutare a definire meglio il contesto del problema. Questo sarà il tuo campione di 7-10 stringhe. L'AntithesisIdentifier sarà responsabile di raccogliere queste stringhe dall'ambiente di simulazione o dalla Rete di Petri.

**Esempio di GapAntithesis esteso:**

C#

public class GapAntithesis : MiuAbstractPattern  
{  
 public string SourceString { get; }  
 public string TargetString { get; }  
 public List<RuleFailureDetail> FailedRulesDetails { get; } // Nuova proprietà  
 public List<string> RelevantContextStrings { get; } // Nuova proprietà  
  
 // ... costruttore e altre proprietà  
}  
  
public class RuleFailureDetail  
{  
 public RegolaMIU FailedRule { get; }  
 public string InitialString { get; } // Stringa su cui la regola è stata applicata/tentata  
 public string ResultingString { get; } // Stringa ottenuta (o null se non applicata)  
 public FailureReason Reason { get; } // Enum: NoMatch, InfiniteLoop, TargetNotReached, etc.  
 // Potrebbe includere anche la "profondità" o il "contesto" del fallimento  
}  
  
public enum FailureReason  
{  
 NoMatch,  
 DidNotReachTarget,  
 InfiniteLoopDetected,  
 TooManySteps,  
 // ... altri  
}

## 3. Logica Interna del ProposeRules(MiuAbstractPattern antithesisPattern)

Il cuore dell'inventore risiederà qui.

### A. Diagnosi del Fallimento (La "Mappa di Karnaugh" Implicita)

1. **Analisi di RelevantContextStrings e FailedRulesDetails:**
   * Itera su ciascuna stringa in antithesisPattern.RelevantContextStrings.
   * Per ogni stringa, simula l'applicazione delle RegolaMIU in antithesisPattern.FailedRulesDetails (o un sottoinsieme delle regole principali) e registra i risultati.
   * Costruisci una rappresentazione mentale (o una struttura dati temporanea) che mostri:
     + **Input Stringa:** La stringa che hai provato.
     + **Regola Tentata:** La RegolaMIU specifica.
     + **Match/NoMatch:** La regola si è applicata a questa stringa?
     + **Output:** Se si è applicata, quale stringa ha prodotto?
     + **Target Raggiunto:** L'output è più vicino al TargetString (se applicabile)?
     + **Caratteristiche della Stringa:** Analizza la struttura della stringa (es. contiene 'M', 'I', 'U', è palindroma, lunghezza, posizione di specifici caratteri/sotto-stringhe). Questo può essere fatto tramite Regex o semplici controlli.
2. **Identificazione dei "Gap Logici" (il "Circuito Mancante"):**
   * Confronta i risultati. Cerca le stringhe per cui *nessuna* regola esistente funziona o per cui il risultato è indesiderato.
   * Identifica le **condizioni (pattern) di input** che portano al fallimento e le **trasformazioni (sostituzioni)** che mancano.
   * **Esempio:** Se "MUI" (input) è il problema, e la Regola R1 ("I -> UII") non si applica perché non vede "I" alla fine o non produce "MUII", il Proposer deve capire che manca una regola che prenda "MUI" e lo trasformi in "MUII", o che fornisca un passo intermedio.

### B. Strategie di Generazione delle Proposte (Le "Tecniche Inventive")

Basandosi sulla diagnosi, il Proposer applicherà una o più delle seguenti strategie per generare RuleProposal:

1. **Proposta di Regola Mirata al Gap/Target:**
   * **Quando:** L'analisi indica un chiaro SourceString e TargetString non raggiungibile.
   * **Come:** Creare una nuova RegolaMIU il cui Pattern sia il SourceString e la Sostituzione sia il TargetString. Questa è la soluzione più diretta ma meno generale. (Es: Pattern: "MUI", Sostituzione: "MUII").
2. **Mutazione delle Regole Fallite/Simili:**
   * **Quando:** Una o più regole esistenti sono *quasi* applicabili o *quasi* generano il risultato desiderato.
   * **Come:**
     + Seleziona le RegolaMIU da FailedRulesDetails o quelle simili all'antithesisPattern.
     + Per ogni regola selezionata, applica **piccole mutazioni** al Pattern o alla Sostituzione (es. aggiungi/rimuovi un carattere, cambia una quantità + in \*, sposta un carattere, cambia l'ordine).
     + Esempio: Se Pattern: "^M(U+)I$" fallisce su "MUI" (dovrebbe matchare), potresti mutare in Pattern: "^M(U+)I\*$".
     + Assegna un nuovo ID e un nome descrittivo ("Mutated\_RuleX\_V1").
3. **Combinazione (Crossover) di Regole Esistenti:**
   * **Quando:** Si sospetta che la soluzione possa emergere dalla combinazione delle capacità di due regole diverse.
   * **Come:** Prendi due regole esistenti. Combina il Pattern di una con la Sostituzione dell'altra, o unisci parti dei loro Pattern o Sostituzione in modo sensato (richiede logica più complessa per mantenere validità).
4. **Generazione Randomizzata Guidata (per Esplorazione):**
   * **Quando:** L'analisi dei fallimenti non è chiara o si vuole introdurre un elemento di esplorazione.
   * **Come:** Genera pattern e sostituzioni casuali ma con vincoli sul tuo "vocabolario" MIU (M, I, U, Regex). L'aspetto "guidato" significa che potresti favorire l'inclusione di caratteri o sotto-pattern presenti nella SourceString o TargetString.

### C. Generazione di TestStartingStates per Ogni Proposta

Per ogni RegolaMIU candidata generata, il Proposer deve creare la List<MiuStateInfo> TestStartingStates appropriata:

* **Includi sempre la SourceString dell'antitesi.**
* **Includi tutte le RelevantContextStrings** dall'antithesisPattern.
* **Aggiungi varianti della SourceString:** stringhe leggermente diverse per testare la robustezza (es. una più corta, una più lunga, una con un carattere in più).
* **Aggiungi "Negative Test Cases":** Stringhe che *non* dovrebbero essere matchate dalla regola, per verificare che non abbia effetti collaterali indesiderati.

## 4. Output del Proposer

Il metodo ProposeRules restituirà una List<RuleProposal>, dove ogni RuleProposal contiene:

* La RegolaMIU candidata generata.
* Il campioncino mirato di TestStartingStates.
* Il TargetAntithesisState (preso dall'input antithesisPattern).

## Prossimi Passi per la Codifica

1. **Estendere MiuAbstractPattern:** Per prima cosa, dobbiamo aggiornare la definizione di MiuAbstractPattern (e GapAntithesis/InefficiencyAntithesis se esistono) per includere FailedRulesDetails e RelevantContextStrings.
2. **Aggiornare AntithesisIdentifier:** Questo componente dovrà iniziare a raccogliere e popolare queste nuove proprietà negli eventi di antitesi che pubblica.
3. **Implementare la Logica nel RuleCandidateProposer:**
   * Creare metodi helper per analizzare i FailedRulesDetails e le RelevantContextStrings.
   * Implementare le strategie di generazione (iniziamo con la mutazione semplice e la proposta mirata al gap).
   * Assicurarsi che i TestStartingStates siano generati in modo intelligente.

Questo piano ci fornisce una roadmap chiara per rendere il tuo "inventore" veramente intelligente.

Sei pronto per iniziare con l'estensione dell'AntithesisPattern?