# NRES: un sistema esperto per la diagnosi di patologie della vite

Cicerello Simone  Fina Pio Raffaele  Nicolì Mauro
Sabato Simone

Gennaio 2018

## Contents

# 1 Introduzione

Naive Raspelli Expert System (NRES) rappresenta un Rule Based System (RBS) Expert System (ES) per la diagnosi di avversità della vite sviluppato in C Language Integrated Production System (CLIPS). Il seguente documento offre un overview ad alto livello sul processo di progettazione e sviluppo del sistema, per specifiche tecniche riguardo l'implementazione si consiglia la lettura dei commenti presenti nel codice. Il documento si articola in 3 sezioni principali: **acquisizione e ingegnerizzazione della conoscenza** nella quale viene illustrato il processo di estrazione della conoscenza dell'esperto e la modellazione del dominio; **Diagonal Computation Model** illustra il framework teorico utilizzato per modellare la computazione del sistema; infine **System Analysis** analizza i dettagli implementativi del sistema realizzato. Durante la lettura vengono illustrati i trade-off e le relative scelte effettuate con le opportune motivazioni.

## 1.1 Scheda riassuntiva

Viene presentata di seguito, una scheda riassuntiva del dominio e delle relative caratteristiche considerate per lo sviluppo di NRES:

| | |
|---|---|
| Dominio | Viticultura |
| Dominio Esteso | Arboricoltura |
| Finalità | Diagnosi patologie della vite |
| Classe S.E. | Diagnosi |
| Tool di sviluppo | FUZZYClips v6.31 (versione custom ricompilata) |
| Conoscenza/Reasoning presente nel dominio | Judgemental Knowledge,<br>Uncertain Knowledge,<br>Temporal reasoning |
| Conoscenza/Inference utilizzata nel sistema | Uncertain Reasoning (limitata)<br>Fuzzy Resoning,<br>Certainty Factor Model,<br>Common sense knowledge (molto limitata) |

## 1.2 Trattamento dell'incertezza

Durante il processo di *knowledge acquisition* abbiamo notato come il dominio presentasse intrinsecamente delle componenti **incerte**. Per modellare naturalmente queste componenti e i relativi meccanismi sottostanti, riducendo il gap tra il modello mentale dell'utente e quello del sistema, abbiamo scelto di utilizzare un *uncertain inference schema* basato sulla *Logica Fuzzy*. La scelta di modellare il dominio utilizzando *variabili linguistiche* evita all'esperto, durante l'elicitazione della conoscenza, un processo di "discretizzazione" dei concetti;

infatti quest'ultimo non è forzato ad esprimere la transizione da un concetto ad un altro in termini rigidi ma in termini di **grado di appartenenza** [1].

Tra le varie alternative valutate, abbiamo deciso di utilizzare l'estensione *FuzzyCLIPS* di CLIPS sia per rispettare i vincoli progettuali che per limitare l'impiego di risorse utilizzate per l'apprendimento di un nuovo sistema. FuzzyCLIPS permette il trattamento dell'incertezza utilizzando 2 tecniche: *Certainty Factor Model* e *Fuzzy Rule Based inference* (basata principalmente sul modello proposto da Mamdavi [2]). Nello specifico il Certainty Factor (Model) (CF) viene utilizzato per esprimere un certo grado di certezza delle ipotesi sulla base delle osservazioni acquisite durante l'interazione con il sistema; la logica fuzzy permette di esprimere concetti con "confini poco delineati" (*soft boundaries*) avendo la possibilità di pesare nel processo di inferenza quanto l'appartenenza di un osservazione alla relativa variabile linguistica incida sull'**attivazione di regole** e **asserzione dei fatti** nella Working Memory (WM) e più in generale nella Knowledge Base (KB).

# 2 Acquisizione e Ingegnerizzazione della Conoscenza

Nella prima parte di questa sezione si descrivono le fasi di acquisizione, inerenti agli incontri con l'esperto del dominio; nella seconda è descritto il processo di ingegnerizzazione della conoscenza acquisita con le relative scelte progettuali adottate e i vari modelli utilizzati per rappresentare al meglio la parte "environment" del processo di diagnosi.

## 2.1 Interviste

Il processo di acquisizione della conoscenza è consistito in:

- una **Fase Preliminare** nella quale il team ha chiarito le principali entità del dominio quali: struttura della pianta, fase fenologica, patologia, sintomo, diagnosi e altri termini tecnici che permettessero di individuare univocamente un'entità o un processo nel dominio
- una successiva **Fase di Approfondimento** dalla quale, tramite metodo "bottom-up", si è cercato di focalizzare subito quale fosse il modello mentale dell'esperto circa l'intero processo di diagnosi; evincendo che:
  - risulta complesso eseguire una diagnosi senza "visita su campo" e di conseguenza il processo deve basarsi sull'osservazione dei sintomi sulla struttura della pianta, i quali verranno poi contestualizzati nel periodo dell'anno
  - la diagnosi, nel mondo reale, è spesso accompagnata dal rilevamento di campioni e successive analisi di laboratorio che possano determinare con certezza il patogeno responsabile dei danni sulla coltura

essendo quest'ultimo punto molto complesso da gestire, ci si è limitati a modellare il primo ponendo come **goal** l'inferenza delle strutture "visibili/presenti" sulla pianta in base al periodo dell'anno nel quale ci si trova (vedi Sezioni 2.3.1 e 2.3.2)

- previa validazione delle entità rappresentate, si è infine passati alla **Fase Estensiva** andando ad estrapolare conoscenza strategica; quest'ultima volta ad affinare il processo di diagnosi, mediante l'uso di *euristiche*, al fine di porre all'utente domande giuste e utili. Da qui si è reso necessario che, così come l'esperto, il sistema sia a conoscenza di altri fattori quali estensione del danno/problema, precipitazioni e interventi di "manutenzione" della coltura, significativi nel processo di diagnosi (vedi Sezione 2.3.3).

## 2.2 Ingegnerizzazione della Conoscenza

Una volta acquisita gran parte dei dati rilevanti mediante conoscenza tacita (proveniente dall'esperto), il team ha ritenuto necessario consultare documenti scritti (fonti multimediali e guide cartacee) per acquisire conoscenza strutturata da utilizzare per astrarre sui dati "grezzi". Ne è conseguita la scelta di considerare il livello superiore della singola patologia, ovvero le **classi di patologie**, manipolate come "categorie" nelle quali le euristiche hanno trovato applicazione; inoltre hanno permesso di individuare caratteristiche comuni (sintomi) tra le patologie.

Il seguente schema descrive:

- in verde, le evidenze richieste all'utente
- con frecce piane: l'inferenza del sistema tramite variabili FUZZY (vedi Sezioni 2.3.1 e 2.3.2) per contestualizzare un determinato giorno nel periodo dell'anno e inferire, tramite la fase fenologica della pianta, le strutture presenti su di essa che potrebbero essere attaccabili da patogeni; ad esempio: in inverno quando le temperature sono basse, è certo che la pianta si trovi nella fase fenologica "riposo", ovvero che la struttura sia ridotta a radice, ceppo e tralci (magari potati)
- con frecce ondulate, dati che condizionano l'attivazione di euristiche le quali influenzano le categorie, "portando avanti" le più credibili
- in azzurro, le entità che saranno usate nel processo di diagnosi (vedi Sezione 4).

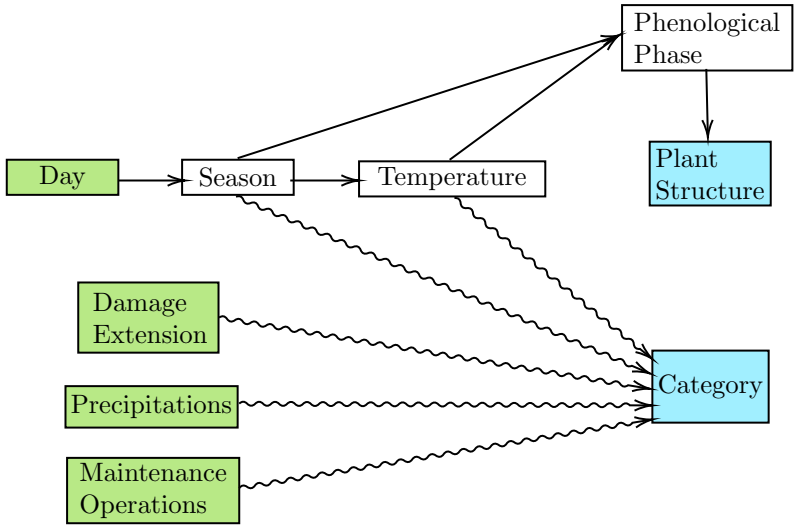


Figure 1: Environment Model

## 2.3 Modelli e Scelte progettuali

In questa sezione vengono esplicitati i modelli utilizzati nella prima parte della diagnosi, denominata **environment** nella quale si acquisiscono informazioni per contestualizzare la pianta nel tempo.

### 2.3.1 Tempo

Durante l'analisi del dominio, il fattore **tempo** è risultato essere di notevole importanza, sia nella parte iniziale della diagnosi che riguardo la sintomatologia; sono state, quindi, individuate 2 relazioni:

- Ciclo di vita: non è stato modellato in quanto difficile da dedurre e manipolare, oltre che dipendente da troppe variabili
- Ciclo annuale: è stato preferito al precedente in quanto sistematico, stabile e attendibile

Data la mancanza di *temporal inference* (Tabella **??**), una prima approssimazione per la gestione del tempo è stata la modellazione dello stesso attraverso la variabile fuzzy **season** la quale ha rappresentato, tramite i suoi range, l'andamento ciclico delle stagioni. Per motivi di implementazione si è scelto di considerare 2 calendari diversi: *real calendar* (stagioni astronomiche) e *system calendar* (stagioni meteorologiche). Il mapping tra i 2 calendari è definito dalla seguente funzione:

$$f(x) = [(x - 15) \bmod 365] + 1$$

con

$$f : D \mapsto D' \qquad D = \{1 \dots 365\}, D' = \{1 \dots 366\}$$

Tale funzione viene implementata dalla *deffunction* `real_to_system_calendar`

### 2.3.2 Fasi Fenologiche

Per mantenere la consistenza con il mondo reale e rimanere in linea con la rappresentazione del modello temporale, si è esplicitata la relazione tra ciclo annuale e struttura della singola pianta. A seconda dei periodi dell'anno (stagioni) e delle relative temperature (range) si sono descritte:

- 3 fasi fenologiche "piene" quali **riposo**, **vegetativa** e **riproduttiva**
- 3 fasi di congiunzione tra le precedenti per coprire ciclicamente tutto l'anno e ottenere un livello di dettaglio maggiore

Complessivamente, tramite le fasi, viene modellato il ciclo di vita di ogni struttura della pianta espresso tramite attributi fuzzy quali `absent`, `growing`, `full` e `decline` i quali valori andranno a influire nel rank descritto in 6.

### 2.3.3 Influenza delle Euristiche

Come illustrato (in generale) nella Figura 1 e in base alla conoscenza acquisita, sono state codificate diverse domande per contestualizzare meglio il processo di diagnosi. In base alle risposte il sistema può modificare l'attributo di *belief* di una determinata categoria; tale meccanismo codifica la conoscenza euristica dell'esperto sui fattori che potrebbero rivelarsi influenti, ovvero:

- **estensione del problema**: in base alla quale si determinano quali categorie di patologie si manifestano in modo localizzato piuttosto che esteso
- **precipitazioni**: possono influire, se presenti, a seconda del tipo di precipitazione, su categorie diverse
- **manutenzione**: operazioni come la potatura potrebbero influire sulla struttura della pianta e dare spazio a determinate avversità

Alcuni esempi:

- Acquisendo l'evidenza che il problema sulla coltura sia localizzato a pochi esemplari saranno più probabili alcune classi di patologie piuttosto che altre; ad esempio, sarà bassa la probabilità di riscontrare un'infezione batterica. Ma i batteri trovano terreno fertile nelle piante che presentano "ferite" (provocate da grandine, potatura), quindi in caso di riscontro di queste evidenze la probabilità dovrà essere rivista (in questo caso aumentata)

- In un periodo primaverile, con temperature medie se si verificano precipitazioni frequenti a carattere piovoso nella coltura, ci saranno condizioni favorevoli per patogeni di origine fungina, di conseguenza la probabilità di riscontrare una malattia appartenente alla categoria funghi sarà maggiore; sempre salvo le influenze di altri fattori sopra citati

Per ulteriori euristiche si rimanda alle relative regole nel codice.

# 3 Diagonal Computation Model

La seguente sezione, illustra un modello semi-formale alla base del S.E., tale modello è da considerarsi come un framework teorico su cui è stata incentrata la computazione del sistema e come "metafora" per l'implementazione dello stesso. Il modello è un estensione del *Conjunctive Classification Model* presente in [3]. Il sistema sviluppato rientra nella categoria dei sistemi esperti per la diagnosi. Nei sistemi più semplici, dove il problema della diagnosi può essere interpretato esclusivamente come problema di classificazione [3], la computazione si svolge visitando un *albero di decisione*; quest' ultimo può essere esplicito, quando la computazione avviene effettivamente su dati strutturati gerarchicamente o implicito. Nel caso del sistema in analisi l'albero è implicito e non viene mai costruito, è il Question Driven Dialog (QDD) a guidare il processo di esplorazione nell'albero di decisione.

La computazione si sviluppa seguendo due dimensioni: il **dialogo**, inteso come la sequenza di domanda-risposta tra il sistema e l'utente; **diagnosi**, inteso come il **gap** che separa le osservazioni dalle ipotesi delle malattia diagnosticate. Lo scopo della computazione, attraverso il QDD e la transizione tra gli stati computazionali, è quello di colmare il gap tra osservazioni e malattie diagnosticate.

**Notazione** siano:

- $Q = \{q_1, \ldots, q_n\}$: l'insieme di domande possibili da porre all'utente durante il QDD. (Spazio delle domande)
- $QDD = (q_s, \ldots, q_f)$: la sequenza di domande poste durante il QDD
- $D = \{d_1, \ldots, d_m\}$ l'insieme di malattie **diagnosticabili** dal sistema
- $D' \subset D$ l'insieme delle patologie **diagnosticate** dal sistema. (Spazio delle patologie)
- $S = \{s_1, \ldots, s_z\}$ l'insieme degli stati computazionali del sistema. (Spazio degli Stati)
- $\oplus$ viene intesa come operazione di "applicazione in sequenza" dei suoi operandi

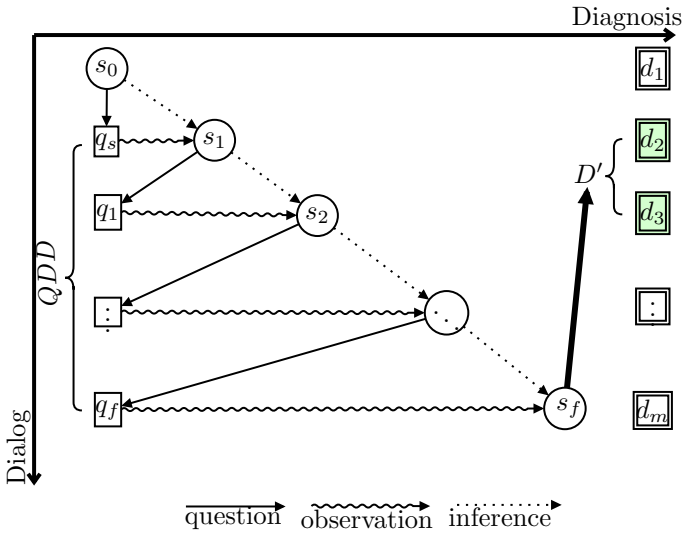


Figure 2: Diagonal Computation Model

### 3.1 Question Generation

Il processo di QDD non è codificato a priori, risulta invece essere parzialmente dinamico. Per ovviare all' *esplosione combinatoria* dovuta dall'enumerazione di tutte le possibili domande, la domanda $q_i \in QDD$ viene generata sulla base dello stato del sistema $s_i$ utilizzando la funzione:

$$\Phi : S \mapsto Q \tag{1}$$

L'avanzamento negli stati computazionali, viene espresso attraverso la funzione:

$$\Delta : S \mapsto S \tag{2}$$

L'acquisizione di nuova informazione dovuta sia dalla risposta (observation nella figura) data dall'utente quando gli viene posta la domanda $q_i$ che dall'inferenza "interna" (*firing* di regole non legate alle osservazioni ricevute) porta l'avanzamento tra gli stati computazionali.

La sequenza $QDD$ è il risultato dell' applicazione iterativa delle funzioni $\Phi(s_i)$ e $\Delta(s_i)$:

$$q_{i+1} = \Phi(s_{i+1}) = \Phi(s_i) \oplus \Delta(s_i) = q_i \oplus \Delta(s_i) \qquad q_i, q_{i+1} \in QDD \tag{3}$$

### 3.2 Ranking Strategy

Ad ogni stato computazionale $s_i$ il task di generazione di $q_{i+1}$, può essere riformulato come un *problema di decisione*. A questo proposito, framework teorici come Multiple Criteria Decision Making (MCDM) e più nello specifico Multiple

Attribute Utility Theory (MAUT), possono risultare di notevole efficacia nella definizione di $\Phi$. Alle possibili domande $q \in Q$, viene applicata una funzione di utilità $u : Q \to \mathbb{R}$, utilizzata per la definizione di un ranking; la domanda generata sarà la domanda che massimizza la funzione di utilità, in formula:

$$q* = \arg\max_{q \in Q} u(q) \tag{4}$$

Dalle considerazioni appena fatte e dalla (3) otteniamo:

$$q_{i+1} = q* \tag{5}$$

Per ulteriori approfondimenti sulle metriche utilizzate per il ranking si rimanda alla sezione 4.2.1.

# 4 System Analysis

La progettazione e lo sviluppo di NRES è stata modellata come un *ambiente multi-agente collaborativo* inducendo a livello strutturale una architettura con topologia di *rete a stella* (vedi 3). Il sistema è stato decomposto in **sub-agenti** (i nodi host della rete), la comunicazione e la coordinazione tra essi, avviene seguendo il *spooke and hub paradigm*: il sub-agente SYS (hub centrale della rete) comunica con gli altri sub-agenti attraverso lo scambio di messaggi (fatti di tipo `deftemplate system_status`). Nella maggior parte dei casi un sub-agente è in relazione 1 : 1 con un modulo definito dal costrutto `defmodule`[4]. NRES presenta tre modalità d'interazione/esecuzione:

1. **diagnosys mode**: l'utente attraverso il QDD interagisce con il sistema fornendo informazioni per la diagnosi della patologia.

2. **engineering mode**: l'ingegnere della conoscenza attraverso un processo guidato può estendere la KB inserendo nuove patologie e i relativi sintomi collegati.

3. **debug mode**: abilita regole utili allo sviluppatore per la fase di valutazione del sistema.

A differenza delle ultime due, le quali risultano modalità accessorie, la prima è sicuramente quella di principale interesse.

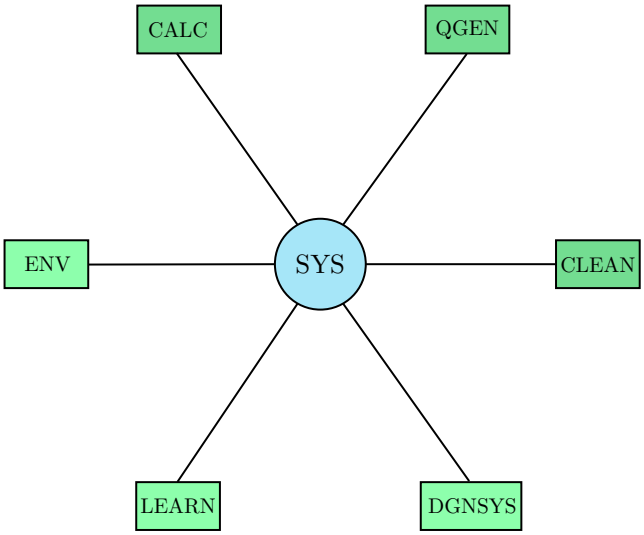


Figure 3: Architettura NRES.

## 4.1 Knowledge Representation

Come tipologia principale per la rappresentazione della conoscenza si è deciso di utilizzare le triplette Object Attribute Value (OAV), poichè permettono di modellare il dominio in maniera generica e flessibile, semplificano il pattern matching e risultano di facile implementazione attraverso il costrutto `deftemplate` [5]. In alcuni casi (come `deftemplate symptom`) per semplificare le operazioni di matching, si è preferito aggiungere un ulteriore slot (`slot desease`) ed utilizzarlo come "valore di join" per collegare fatti di `deftemplate` diversi (`deftemplate symptom` e `deftemplate desease`).

## 4.2 Inference: RAD cycle

Il Diagonal Computation Model (DCM) presentato nella sezione 3 è stato implementato nel sistema seguendo la metafora del ciclo:

*rank-ask-delete*

Le operazioni effettuate durante il Rank Ask Delete (Cycle) (RAD), con le dovute differenze, ricordano le stesse utilizzate nell'algoritmo di classificazione *ID3*[6].

Per controllare il flusso di esecuzione del RAD, il quale s'integra e adatta al *recognize-act cycle* effettuato da CLIPS, vengono utilizzati i moduli `CALC, QGEN, CLEAN` (vedi figura 3 i nodi in verde scuro) . Come esposto precedentemente il modulo `defmodule SYS` contiene `deftemplate` e `defrule` che permettono il coordinamento delle operazioni di sistema. Di notevole importanza è la `defrule next_phase`, la quale, iterativamente, attiva il focus del modulo specificato nello slot `sequence`.

### 4.2.1 Rank

Anche se basato su metriche differenti da quelle proposte in *ID3*, durante la **fase di rank** avviene comunque una operazione di *"best feature selection"* per la generazione della domanda da porre all' utente. Le informazioni riguardo i sintomi e le malattie presenti nella KB vengono condensate nella struttura `deftemplate damaged_struct`, a cui può essere data la seguente interpretazione semantica:

"ogni fatto damaged_struct rappresenta **quanti** e **quali sintomi** delle malattie di una determinata **categoria** presenti nella KB colpiscono una specifica **struttura** della pianta."

Ad ogni fatto `damaged_struct` è associato un **valore di ranking** con il quale viene definito un ordinamento totale. Per generare i suddetti valori, dalle motivazioni evinte in [7], viene utilizzato un Weighted Product Model (WPM) per definire la **funzione di rank**:

$$r(s,c) = c_{belief}^{\alpha} \cdot s_{lifetime}^{\beta} \cdot \log_{10}(f_s + 1)^{\gamma} \tag{6}$$

Dove:

- $rank : S \times C \to \mathbb{R}$
- $s$: rappresenta la struttura della pianta (grappolo, foglia, ecc..)
- $c$: rappresenta la categoria della malattia (funghi, batteri, ecc..)
- $c_{belief}$: rappresenta il grado di *belief* assegnato alla categoria $c$ durante l'inferenza.
- $s_{lifetime}$: rappresenta il grado di *belief* assegnato alla struttura $s$ durante l'inferenza.
- $f_s$: rappresenta la *frequenza cumulativa* dei sintomi che colpiscono la struttura $s$
- $\alpha, \beta, \gamma \in [0, 1]$: rappresentano i pesi del WPM

Si noti che la funzione *rank*, implementata da `deffunction calculate_rank` e `defrule update_rank`, è stata definita sulla base della conoscenza dei progettisti e dell'esperto dopo una fase di sperimentazione empirica per validare la stessa. Il logaritmo presente in (6), considerando l'estensibilità del sistema, viene utilizzato per imporre una crescita *sub lineare* ad $f_s$ in quanto dipendente dal numero di sintomi presenti nella KB.

### 4.2.2 Ask

Un problema di diagnosi è generalmente formulabile in termini di concorrenza tra ipotesi mutualmente esclusive, nel quale il task principale per il problem solving rimane la scelta dei dati da acquisire per verificare le ipotesi effettuate o formularne delle nuove [3]. Per motivazioni quali, **dinamicità** ed **estensibilità** del sistema, la sequenza di domande poste all'utente non è codificata a priori; ad ogni iterazione, il fatto `damaged_struct` con valore di ranking più alto $r^*$ viene utilizzato per generare una sequenza di domande, una per ogni valore dello slot `symptoms`:

$$\left.\begin{matrix} [c_1, s_1, f_{s_1}, \overbrace{\{sm_{11}, sm_{12}, \ldots, sm_{1m}\}}^{\text{slot symptoms}}, r^*(c_1, s_1)] \\ \ldots \\ [c_n, s_n, f_{s_n}, \{sm_{n1}, sm_{n2}, \ldots, sm_{nm}\}, r(c_n, s_n)] \end{matrix}\right\} \to \overbrace{\left(q_{sm_{11}}, q_{sm_{12}}, \ldots, q_{sm_{1m}}\right)}^{\text{sequenza di domande}}$$

Il meccanismo di generazione delle domande viene implementato attraverso: `deffunction get_rank_pos, get_allowed_values, ask_question` e `defrule generate_question`

**Question Template** Ogni domanda $q_{sm_{ij}}$ viene generata utilizzando un template predefinito, strutturalmente il template può essere espresso dalla seguente grammatica:

⟨*domanda*⟩ ::= `La` ⟨*struttura*⟩ `presenta` ⟨*sintomo*⟩ `?` (⟨*risposta consentita*⟩)

⟨*risposta consentita*⟩ ::= `si,no` | `risposte possibili:` ⟨*value*⟩+ altro

⟨*struttura*⟩ ::= `radice`|`ceppo`|`tralcio`|`foglia`|`infiorescenza`|`grappolo`

⟨*sintomo*⟩ ::= ⟨*attributo binario*⟩ | ⟨*attributo*⟩ ⟨*valore*⟩ | ⟨*attributo*⟩

⟨*attributo binario*⟩ ::= deformazione|marciume|disseccamento|caduta|tacche|escrescenze|puntura

Dalla grammatica presentata, è possibile creare 3 principali tipologie di domanda:

- Domanda binaria: "La radice presenta muffa? (si, no)"
- Domanda specifica: "La foglia presenta macchia gialla? (si, no)"
- Domanda generica: "La foglia presenta macchia di forma? (regolare, irregolare, puntiforme, mosaico, **altro**)"

Si osservi che per mantenere un certo grado di semplicità nella procedura di generazione, non sono state considerate tecniche di Natural Language Processing (NLP) e Natural Language Generation (NLG) ne tanto meno sono state codificate *meta informazioni* riguardo la correttezza sintattico-linguistica della domanda, per tanto come *side-effect* si potrebbero generare delle domande del tipo:

"La grappolo presenta macchia colore giallo?"

La caratteristica più interessante consiste, attraverso la scelta del valore *altro*, nel lasciare all'utente la possibilità di rispondere con un valore diverso da quelli presenti nella KB. Ovviamente tale scelta porterà inevitabilmente al fallimento della diagnosi poiché le regole che codificano le patologie non "matcheranno" con i valori specificati. La scelta effettuata può essere rivista in un contesto di sviluppi futuri per aumentare la **flessibilità** e più in generale migliorare le prestazioni del sistema.

### 4.2.3 Delete

Citando Stefik [3]:

> When is there enough evidence to establish a fault or disease in diagnosis? Roughly speaking, for most diagnostic programs the answer is "when all of the other plausible alternatives are ruled out." Diagnosis proceeds by elimination rather than by seeking more and more evidence for a favorite hypothesis.

Le operazioni effettuate nella fase di delete sono basate sulla considerazione appena espressa. Dopo aver acquisito l'evidenza espressa dall'utente, come risposta alla domanda effettuata nella fase precedente, il sistema attraverso la regola `defrule clean_symptoms_by_evidence` procede ad eliminare dalla KB i relativi ai sintomi, nello specifico elimina:

- i fatti relativi ai sintomi che **confermano** l'evidenza acquisita, in quanto la domanda relativa ad esso è stata già effettuata e il sintomo non dovrà incidere, nella successiva iterazione, nel calcolo del ranking.
- i fatti relativi ai sintomi che **negano** l'evidenza acquisita, in quanto data la mutua esclusività di essi, non possono essere considerati per la diagnosi finale.

## 4.3 Learning

NRES prevede una componente di **learning** basata su aggiornamento della conoscenza semi-automatico. L'apprendimento viene attivato, dal modulo `defmodule LEARN`, qualora il sistema non non riesca a diagnosticare una patologia presente nella KB in base alle informazioni acquisite. Il QDD fornisce al sistema esattamente le asserzioni di tipo `deftemplate oav` corrispondenti alla nuova malattia, a questo punto, viene fatto inserire all'utente il nome e la classe di malattia alla quale appartiene. Attraverso le funzioni `create_patologia_rule` e `create_patologia_deffacts` le informazioni acquisite verranno memorizzate in due distinti file:

- `learned_deffacts.clp`: contenente i `deffacts` che arricchiscono la base di conoscenza.
- `learned_rule.clp`: contenente le regole che codificano la nuova patologia per le successive diagnosi.

I 2 file sopra citati verranno caricati dinamicamente al termine della fase di learning (o engineering mode). Il modulo di learning è altresì pensato per essere riutilizzato per arricchire la KB tramite la **modalità engineering**.

## 4.4 Explanation

Per quanto riguarda il **Modulo Spiegazioni** un approccio possibile potrebbe essere quello del *why-how* ovvero che, ad ogni interazione con l'utente il sistema offre la possibilità di chiedere il perchè della domanda e il come si è arrivati ad una risposta, seppur essa intermedia nell'intero processo di diagnosi. Questa strategia può essere implementata con:

- un testo predefinito associato ad ogni domanda
- l'esplicitazione delle esecuzioni del sistema rendendo trasparenti all'utente tutti i dati che si elaborano

comune denominatore è il fatto di informare l'utente sullo stato del sistema e giustificare delle azioni in base ai risultati di interazione, per dare coerenza e credibilità.

Si è scelto di implementare un approccio del primo tipo, nel quale l'utente ha la possibilità, tramite il carattere `[x]` (explanation), di sapere il perchè gli è stata posta quella determinata domanda in quel punto del processo ed eventualmente le possibili strade che il sistema intraprenderà a seconda dell'input.

# 5 Considerazioni e Sviluppi Futuri

L'utilizzo di euristiche, rispetto ad una soluzione puramente procedurale, ha permesso di ridurre notevolmente l'impegno per lo sviluppo. In alcuni casi, come l'implementazione del RAD dove veniva richiesta una conoscenza di tipo procedurale, il paradigma è risultato abbastanza limitativo e ha prodotto una soluzione sicuramente meno coincisa di quella procedurale. Dopo un'analisi retrospettiva, siamo giunti alle seguenti considerazioni :

- Utilizzare fuzzy inference e più in generale l'incertezza durante tutto il processo di diagnosi, non solo nella prima fase di inferenza sull'ambiente.
- Refactoring del codice che interessa le modalità d'interazione con l'utente, attraverso un controllo sintattico dell'input o più in generale lo svilluppo di un utility Read Eval Print Loop (REPL).
- Migliorare il QDD modificando il RAD, basandosi su un algoritmo di ricerca generalizzato come $A*$ invece che uno *best-first*.
- Rendere la utility di spiegazione un "cittadino di prima classe", introdurre conoscenza di supporto ed euristiche che migliorino il processo di spiegazione.
- Introdurre tecniche NLP e NLG per il processamento dell'input e più in generale nell'inferenza. Un agente intelligente, per essere considerato tale, non può non presentare tali caratteristiche.
- Sostituire la componente di learning semi-automatica con una basata su Machine Learning (ML) e più in genereale su un framework teorico come *Statistical Learning Theory*.

Il team NRES ringrazia.

# References


[1] Earl Cox. *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[2] E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1 – 13, 1975.

[3] Mark Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[4] Gary Riley. Clips reference manual. volume 1-basic programming guide, nov 2017.

[5] Joseph C. Giarratano and Gary Riley. *Expert Systems: Principles and Programming*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1994.

[6] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.

[7] Chris Tofallis. Add or multiply? a tutorial on ranking and choosing with multiple criteria. *INFORMS Transactions on Education*, 14(3):109–119, 2014.


# Acronyms

**CF** Certainty Factor (Model).

**CLIPS** C Language Integrated Production System.

**DCM** Diagonal Computation Model.

**ES** Expert System.

**KB** Knowledge Base.

**KE** Knowledge Engineering.

**MAUT** Multiple Attribute Utility Theory.

**MCDM** Multiple Criteria Decision Making.

**ML** Machine Learning.

**NLG** Natural Language Generation.

**NLP** Natural Language Processing.

**NRES** Naive Raspelli Expert System.

**OAV** Object Attribute Value.

**QDD** Question Driven Dialog.

**RAD** Rank Ask Delete (Cycle).

**RBS** Rule Based System.

**REPL** Read Eval Print Loop.

**SE** Sistema Esperto.

**WM** Working Memory.

**WPM** Weighted Product Model.