Uma Lição Sucinta de Sistemas Periciais com PROLOG e KAN

Dulce Sá[1], Victor Santos[2], João Negreiros[4]

dulce@isegi.unl.pt, vitors@microsoft.com, c8057@isegi.unl.pt

Universidade Lusófona, Campo Grande 375, 1749-024, Lisboa, Portugal

Resumo: Um dos problemas mais relevantes da quinta geração de computadores é a Inteligência Artificial e, consequentemente, os Sistemas Periciais. A capacidade de fornecer inteligência inferencial e conclusiva (*deductive reasoning*) a uma máquina e a capacidade de aprender novos factos (*rule-base programmes on a database heuristic facts*) é deveras um desafio estimulante. Rever o *state-of-the-art* desta área de conhecimento através da exemplificação de um programa em PROLOG e a utilização do Expert System KAN (ver http://paginas.ulusofona.pt/2203) apresenta-se como o propósito principal deste artigo.

Palavras-chave: Inteligência artificial (IA), Sistemas Periciais, PROLOG, KAN.

1. Introdução

> *"Estou profundamente sensibilizado pela notícia que um programa de computador tenha finalmente ultrapassado o campeão do mundo de xadrez num jogo completo. Para todos aqueles que como nós assistiram ao nascimento do campo da IA, este era sem dúvida o problema do grande desafio."* ***Edward Feigenbaum***

Uma plausível definição de *Expert System* (ES) consistiria num sistema (interface do utilizador, base do conhecimento, motor de inferência) que realiza uma tarefa desempenhada por um especialista humano numa específica área de conhecimento tal como a comercial (*Xcon*, *Idea*, *Screenio*, *ExperTax*), financeira (*Authorizer Assistant*), medicina (*Mycin*, *Internist*, *Dendral*, *PathFinder*), matemática (*Macsyma*), geologia (*Prospector*, *Drilling Advisor*) ou educação (*Excheck*). Contudo e independentemente do âmbito de estudo, é fundamental que o problema em causa seja bem estruturado e com um domínio perfeitamente definido.

O processo eficiente de fabrico de um produto, por exemplo, tem sido uma das análises de maior utilização deste tipo de tecnologia, isto é, no controlo de processos artificiais. Assim, o objectivo do controle de processos é permitir que um conjunto de variáveis dinâmicas fiquem fixas ou perto do valor especifico desejado. Como estas podem conter um comportamento aleatório, a sua correcção constante é deveras uma necessidade. A figura seguinte representa um sistema de controlo de processos genérico no qual existe um sistema de aquisição de dados com base num computador especializado para o efeito. Assim, o computador realimenta o processo através da regulação automática dos pontos de ajustamento da malha. Naturalmente, á medida

que aumenta o número de variáveis, mais difícil é manter a realimentação de controlo de forma eficiente, isto é, mantendo os valores dentro de certos limites pré-estabelecidos pelo operador. Acontece que o ponto de funcionamento de processo industrial depende de muitos parâmetros e, por vezes, a decisão de alterar um ponto de ajustamento pode implicar a alteração de muitos outros indicadores de ajustamento nas malhas de modo a ter em conta os efeitos de interacções numa linha de produção. Este controlo de processos poderá ainda ser englobado num outro ES e com uma função de análise do funcionamento global de todo o sistema. Os resultados são avaliados por testes, também estes programados para supervisionar a operação do processo global do qual são tomadas as entradas. As previsões dos rendimentos futuros, a avaliação da eficiência, as tendências para os desvios e muitas outras operações podem, portanto, ficar ao alcance do operador do processo.

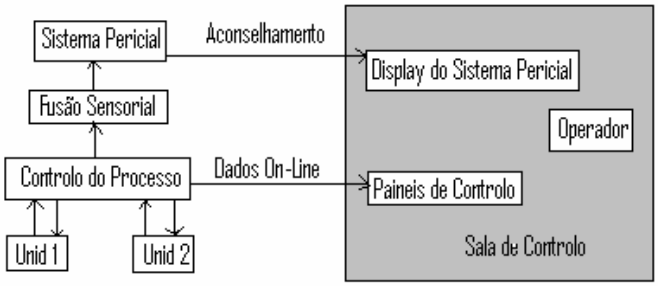

Figura 1 – Integração do sistema pericial como supervisor.

De uma forma abstracta, a configuração de um ES no processo de tomada de decisão compreende, em geral, os seguintes elementos:

- *Knowledge*: Conjunto de factos que têm interesse para a resolução do problema. Dele faz parte o conjunto de regras ou proposições estruturadas numa linguagem formal que pode ser concebida sob a forma de condições lógicas ou de uma rede semântica (expressão de relações que ocorrem num contexto de significados idênticos, utilizando conceitos de memória humana) designadas por crenças do sistema.
- Planeamento: Específica a ordem cujas regras devem ser aplicadas às bases dados e a forma de resolver eventuais conflitos que possam surgir na interacção das regras (determinísticos e heurísticos). Deve-se, no entanto, tentar limitar o espaço de busca da solução à mais pequena das regiões que a contém. Nesta busca da solução, admite-se um estado inicial ao qual se aplicam determinadas funções para produzir um novo estado e, num processo sucessivo, chegar à solução desejada.

Este problema do espaço de busca parece bastante simples quando o número de nós é reduzido. No entanto, a maioria dos problemas necessita de uma *mainframe* para que se possa realizar uma busca total, pois o número de nós do espaço de busca pode ser enorme produzindo um elevado conjunto de diferentes caminhos. De facto, o número de caminhos aumenta

abruptamente à medida que se adiciona nós. Pode-se associar esta ideia às permutações matemáticas. Por exemplo: Quatro nós podem ser combinados em 4×3×2×1=24 maneiras diferentes mas se adicionarmos mais três fica-se com 7×6×5×24=5040 combinações. Por este motivo, desenvolveram-se técnicas de busca das quais se destacam a busca em profundidade e a busca em amplitude.

2. Lógica de Predicados no PROLOG

Inferência é o processo pelo qual se obtêm conclusões com base no conhecimento anterior. Sumariamente, existem três tipos:

- Indução - Se os conhecimentos são representados na forma de factos, então regras de dedução a partir da combinação dos conhecimentos já existente podem ser accionadas (*modus ponens*).
- Abdução - Inferência dita não legal ou meramente plausível (um processo de gerar explicações).
- Dedução - Processo a que se refere como sendo a generalização de leis a partir de exemplos particulares. Assim, esta resolução de provas por refutação e encapsulada por variados procedimentos necessários incorpora as seguintes fases:
  1) Conversão de todas as frases para forma causal.
  2) Negar o teorema a demonstrar e adicioná-lo à base de conhecimentos gerado em 1.
  3) Repetir os passos seguintes até se encontrar uma inconsistência ou não se conseguir progredir mais:
     1) Seleccionar duas cláusulas.
     2) Resolver essas cláusulas, substituindo/unificando como apropriado até se produzir um resolvente
     3) Caso o resolvente não seja uma cláusula vazia, adicione-se ao conjunto de cláusulas. Caso contrário, encontrou-se uma substituição e ficou demonstrado o teorema.

No exemplo seguinte relativo a graus de parentesco, é possível reconhecer várias definições declarativas recursivas e factos com/sem parâmetros de *input*. Mais, persiste um conjunto de axiomas e uma computação como prova construtiva de uma afirmação a partir dos axiomas. Note, também, que o raciocínio é realizado sob o princípio do mundo fechado.

```
pai(Zé, Ana)
pai(Zé, João)
pai(Zé, Isabel)
pai(Zé, Zé Miguel)
pai(João, Miguel)
pai(João, Marta)
```

```
mãe(Rita, Miguel)
mãe(Rita, Marta)
homem(Zé)
homem(João)
mulher(Ana)
mulher(Rita)
mulher(Marta)
pai((Zé, Teresa) (Ana, João))
pai((João, Rita) (Miguel, Marta))
pai((Zé Miguel, Sónia) ())
avo(x,y)          if pai(x,z) and pai(z,y)
antecessor(x,y)   if pai(x,y)
antecessor(x,y)   if pai(x,z) and pai(z,y)
irmão(x,y)        if homem(x) and pai(z,x) and pai(z,y) and noteq(x,y)
bizavo(x,y)       if pai(x,z) and avo(z,y)
```

Através deste código PROLOG (PROgramming in LOGic, linguagem baseada nas cláusulas de Horn da lógica de predicados de 1ª ordem), é possível questionar o seguinte:

- ?- pai(X,Ana) 'Quem é o pai da Ana?
- ?- mãe(X,Miguel) 'Quem é a mãe do Miguel?
- ?- mulher(X) 'Quem é mulher?
- ?- pai((X,Y)(Miguel,Marta)) 'Quais são os pais do Miguel e da Marta?
- ?- pai((X,Y)()) 'Que casais não tem filhos?
- ?- avo(X,Isabel) 'É possível indicar o avô da Isabel?
- ?- irmão(X,Y) 'Quem é irmão de quem?

3. A Capacidade de Aprendizagem no PROLOG

Uma outra técnica da programação em lógica é a sua capacidade de aprendizagem baseada num depósito de factos e adição dos mesmos, caso ainda esse conhecimento ainda não exista. O exemplo seguinte (ver anexo A) consiste num jogo em que mediante diversas perguntas do tipo sim/não, o programa adivinha o animal pensado pelo utilizador desde que este exista na base de conhecimento. Caso o programa não obtenha a solução correcta, este apresenta uma solução próxima ao animal pensado pelo utilizador (ver tabela 1). Posteriormente, pede-se ao utilizador que introduza o nome do animal em questão bem como uma das características que o distinga da solução a que o ES tenha chegado. Assim, este adquire novo conhecimento dada a nova capacidade de identificar mais um animal. De um modo sucinto, esta solução baseia-se no uso de árvores binárias ordenada e em recursividade (capacidade de executar o mesmo procedimento N vezes, sendo N igual ao número de nós a percorrer até ser alcançando uma solução).

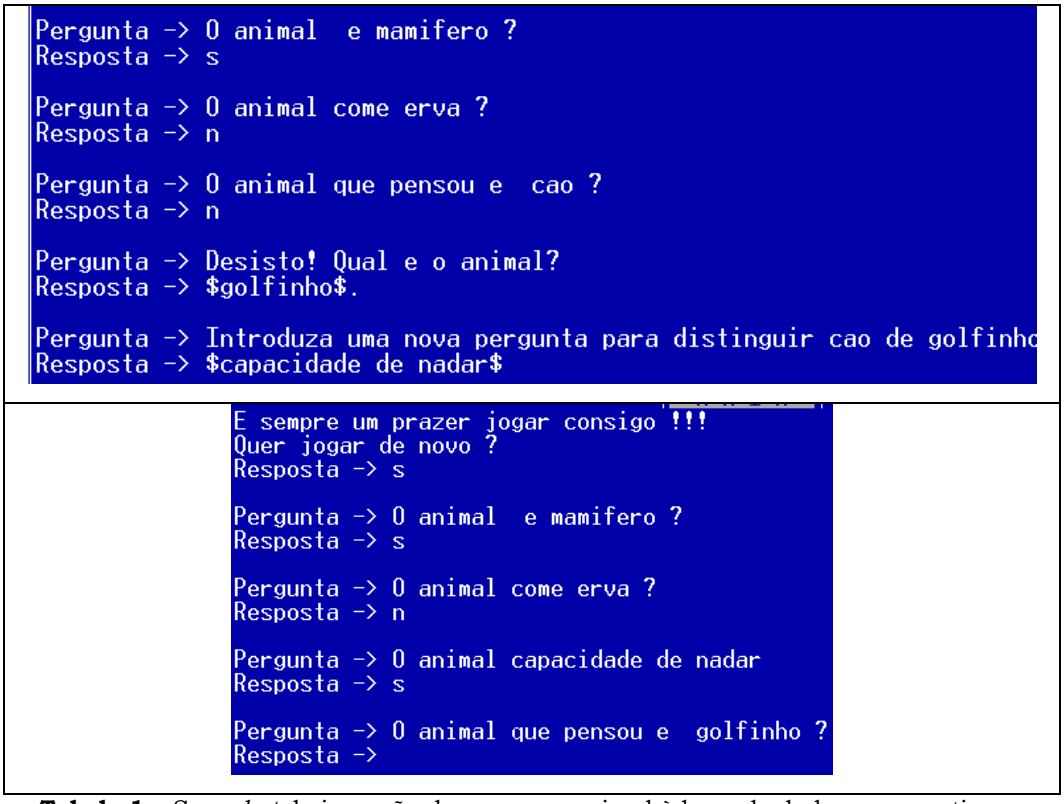

Tabela 1 - *Snapshot* de inserção de um novo animal à base de dados e respectivas características (superior) e posterior adivinha da aplicação (inferior).

5. KAN

Uma dos campos de estudo de maior sucesso de implementação dos ES tem sido na medicina como é o caso particular do KAN. Escrito em linguagem C com 26000 linhas (105 ficheiros), este ES foi criado para Macintosh®, UNIX® workstations e DOS® (*a\>KANv01.exe*) pelo *VUB AI Laboratory, Brussels*. Genericamente, a sua representação interna de conhecimento na resolução de problemas é realizada através de regras de produção (se..então…) e baseada em casos e experiências passadas. Tecnicamente, os objectos utilizados são vocabulário (descrição de características de objectos genéricos), tipo de objectos (por exemplo: pacientes, carros, fruta…), relações entre objectos (*roles*), regras (constituída por um conjunto de condições e acções), resolução de problemas (*problem-solver*, isto, definição do objectivo do ES), propriedades (descrição booleana de uma determinada característica de um objecto), atributos (valores possíveis de um descritor), descritores (expressões que permitem criar factos e formular condições nas regras), factos (especificações sobre o conhecimento de objectos) e perfis. Identificados univocamente, um dos objectos mais relevantes é a definição de regras como a seguinte:

```
(define (rule infection-rules) german-measles-rule
        (if fever running-nose coughing
                pink-spots enlarged-glands
```

```
                ear-inflammation)
        (then
                (conclude (infection-disease german-measles)))
    (documentation "Taken from Gothenburg(1980), p. 426"))
```

Com se verifica por este exemplo, esta expressão e subdividida em três secções: *if*, *then* e *documentation*. Paralelamente, a criação de um vocabulário subdivide-se na documentação de apoio e nos conteúdos dos atributos afectados. Presumivelmente, na utilização dos objectos KAN, apenas devem ser referenciadas se já definidos anteriormente. Não faria sentido definir *rule-sets*, por exemplo, sem definir factos ou tipos de objectos.

Contudo, esta secção não pretende apenas indagar sobre a faceta técnica do ES em causa mas exteriorizar o simples funcionamento, segundo a óptica de um mero paciente que tem uma doença e procura saber qual o seu diagnóstico de cura (tal como quando consulta um médico). Assim, os passos iniciais correspondem à leitura do ficheiro que contém a base de conhecimentos sobre doenças contagiosas (ver figura seguinte, cima) e indicação dos sintomas do paciente (baixo).

```
THERAPY defined
START-RULES defined
INFECTION-RULE defined
NO-DISEASE-RULE defined
INFECTION-THERAPY-RULE defined
NO-INFECTION-THERAPY-RULE defined
INFECTION-DIAGNOSIS-RULES defined
GERMAN-MEASLES-RULE defined
MUMPS-RULE defined
DIPHTERIA-RULE defined
SCARLET-FEVER-RULE defined
POLIOMYELITIS-RULE defined
NO-INFECTION-RULE defined
INFECTION-THERAPY-RULES defined
GERMAN-MEASLES-IMMUNIZATION-RULE defined
GERMAN-MEASLES-NO-IMMUNIZATION-RULE defined
SCARLET-FEVER-IMMUNIZATION-RULE defined
INFECTION-WITH-NO-THERAPY-RULE defined
 #.(PATHNAME "A:\\CONDIS.KAN")  loaded
 ==== KAN ====

  Q or 0 means Quit. ? asks for help.
   1. Select Case.    2. New case.    3. Select Rule-set.
   4. Reason.    5. Discuss.    6. Inspect.
   7. Load.    8. Parameters.>>
```

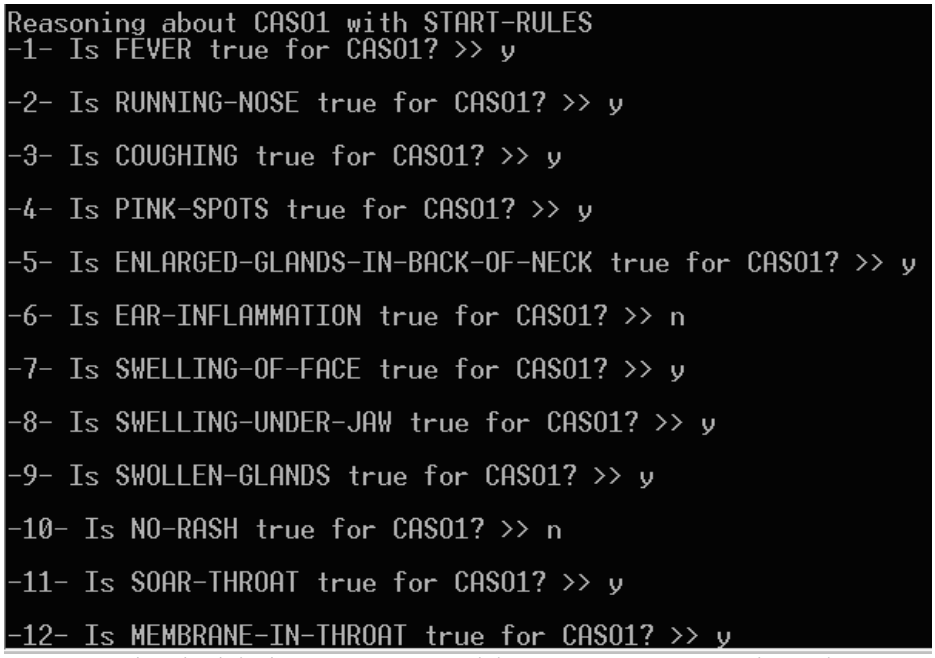

Figura 2 – *Snapshot* de dois *layouts*. Repare o leitor que as respostas do paciente CASO1 ao KAN são do tipo booleano (*y* ou *n*).

Certamente, a discussão dos sintomas, diagnóstico, infecção e terapia para o problema apresenta-se como uma etapa fundamental. Neste caso particular, o paciente CASO1 apresenta a doença SCARLET-FEVER (escarlatina) pois manifesta as cinco características básicas desta doença (ver figura seguinte, cima). A confirmação do tipo de terapia a aplicar implica uma injecção de penicilina (baixo).

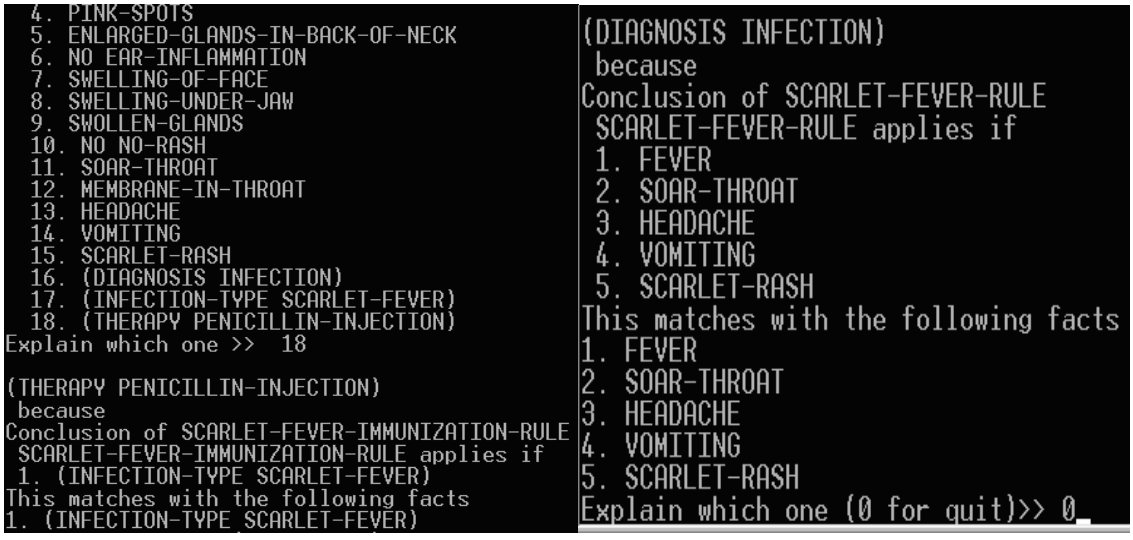

Figura 3 – Novo *snapshot* do *KAN layout*.

6. Conclusão

Tendo em conta a progressiva complexidade dos sistemas tecnológicos, económico e social, a capacidade cognitiva de criar software que não adopte a forma determinística e procedural

começa a ser uma necessidade a ser preenchida por ES e redes neuronais (dendrites, axónios). Este último conceito tem sido especialmente útil em tarefas de classificação e *Datamining*, por exemplo, embora neste *paper* apenas o primeiro ponto seja focado. Formalmente e baseado nos diversos tipos de procuras heurísticas, é identicamente possível procurar soluções num universo bastante grande, se associado a custos de procura e probabilidades (uma outra situação também não abrangida). De facto, a única solução aqui anunciada é a da restrição do universo de soluções.

Como concretização da lógica de predicados de 1ª ordem utilizada no PROLOG, foi apresentado um simples programa de aprendizagem de conhecimento em Arity Prolog® e com a possibilidade de questionar o sistema sobre conhecimentos já adquiridos. Finalmente, é analisado uma referência ao ES KAN, um *Rule-Based Knowledge System software* desenvolvido numa base estratégica de procura em profundidade (*forward chaining*).

Anexo A

zoo.ari

```
zoo:-
    cls,
    open(Fp,'zoo.txt',rw),
    read(Fp,List),
    prompt_pergunta,
    write('Pense num animal e carregue numa tecla...'),
    get0_noecho(Readkey),
    cls,
    adivinha_animal(List).

adivinha_animal(List):-
    localiza_animal(List,NewList),
    prompt_pergunta,
    cls,
    write('E sempre um prazer jogar consigo !!!'),
    nl,
    write('Quer jogar de novo ?'),
    prompt_resposta,
    get(Resp),
    if then((Resp=:=115;Resp=:=83),adivinha_animal(NewList)),!,
    cls,
    create(Fp1,'zoo2.txt'),
    writeq(Fp1,NewList),
    close(Fp),close(Fp1),
    cls,
    write('adeus !').

localiza_animal(Aux,NewAux):-
    folha(Aux),
    nl,
    prompt_pergunta,
    write('O animal que pensou e  '),
    write(Aux),
    write(' ?'),
    prompt_resposta,
    get(Resp),
ifthenelse((Resp=:=115;Resp=:=83),NewAux=Aux,insere_animal(Aux,NewAux)
)
localiza_animal([Arvesq,No,Arvdir],NewAux):-
    nl,
    prompt_pergunta,
    write('O animal '),
    write(No),
    prompt_resposta,
    get(Resp),
    ifthenelse((Resp=:=110;Resp=:=78),
      (localiza_animal(Arvesq,NewAux2),NewAux=[NewAux2,No,Arvdir]),
      (localiza_animal(Arvdir,NewAux2),NewAux=[Arvesq,No,NewAux2]))

insere_animal(Aux,NewAux):-
    nl,
    prompt_pergunta,
    write('Desisto! Qual e o animal?'),
    prompt_resposta,
    read(Animal),
    prompt_pergunta,
    write('Introduza uma nova pergunta para distinguir '),
    write(Aux),
```

```
    write(' de '),
    write(Animal),
    prompt_resposta,
    read(Pergunta),
    NewAux=[Aux,Pergunta,Animal].

folha(Aux):-
    string(Aux).
    prompt_pergunta:-
    nl,
    write('Pergunta -> ').
    prompt_resposta:-
    nl,
    write('Resposta -> ').
```

zoo.txt

```
[[$tartaruga$,$muda a pele ?$,$cobra$],$ e mamifero ?$,[$cao$,$come
erva ?$,$vaca$]]
```