Um Processo de Classificação aplicado a Objectos Gráficos

Manuel João Próspero

Departamento de Informática
Universidade Nova de Lisboa

RESUMO

Esta comunicação aborda um problema que surge com as interfaces gráficas de determinadas aplicações. Trata-se do reconhecimento de objectos cujas imagens são mostradas ao utilizador e por este podem ser interactivamente manipuladas. Como dado do problema apresenta-se uma determinada descrição, a partir da qual os objectos serão classificados no sentido de serem usados posteriormente numa certa operação ou apenas para caracterizarem um estado da computação. O uso da programação em lógica, concretizado na linguagem Prolog, permitiu a implementação protótipo de um sistema de modelação (MGL) capaz de lidar, a alto nível, com a entrada gráfica produzida por um utilizador e relacioná-la com o modelo da aplicação. É aqui mostrado um exemplo de como este sistema poderá complementar um processo de classificação baseado em aspectos gráficos estruturais. A modularidade da forma clausal, sob a qual se representou o conhecimento do domínio, traduz-se numa vantagem adicional para a manipulação de regras num elevado grau de independência.

1. Descrição de Objectos ao Sistema Gráfico

A primeira finalidade de um Sistema Gráfico é, sem dúvida, a de permitir a tradução de certos conjuntos de dados em imagens inteligíveis por seres humanos. No domínio da síntese de imagens e na grande maioria dos casos, essas descrições pictóricas são preferidas, em termos de análise, à informação (textual ou numérica) que esteve na base da sua criação. Na verdade, há conceitos que são muito mais facilmente por nós interpretados quando apresentados sob a forma gráfica, quer se trate da sintetização de objectos reais que nos são familiares (como casas, mesas, aviões, etc.), quer abstracções do foro de uma larga variedade de áreas específicas do conhecimento (caso da visualização de determinados fenómenos físicos ou da representação de resultados estatísticos, por exemplo).

Tal como acontece com as próprias linguagens de programação, um Sistema Gráfico é concebido de molde a ser usado por uma vasta gama de aplicações. Nesse sentido, os dados com que um modelo pode ser construído, permitindo a sua interpretação por aquele sistema, não deverão ser demasiadamente complexos. Caso contrário envereda-se por uma solução fortemente restritiva, impossível de aplicar nalguns casos e compreensivelmente ineficiente para outros. O compromisso tomado na concepção dos actuais sistemas gráficos tem conduzido sempre a

estruturas de dados bastante simples e, portanto, de aplicação muito geral.

Não é suficiente, porém, a existência de determinadas estruturas de dados para que um programa de aplicação possa descrever ao Sistema Gráfico os objectos do seu interesse. A tais estruturas terá que, naturalmente, associar-se uma funcionalidade específica. Esta é concretizada na forma de primitivas gráficas. Sistemas concebidos para se tornarem normas, como GKS [ISO85, ISO87] e PHIGS [ISO86], possuem primitivas denominadas *Polyline*, *Polymarker*, *Fill Area*, *Text* e *Cell Array*.

Para permitir modificações selectivas, bem como facilitar a cópia de informação gráfica, as normas actuais (já em vigor ou ainda em estudo para aprovação) incluem o conceito de segmentação. Um «segmento» é essencialmente uma sequência de primitivas de saída gráfica que o Sistema Gráfico pode manipular como um todo. A introdução, em sistemas como PHIGS, da possibilidade de segmentação a vários níveis e com edição da informação levou à necessidade de um novo nome, dito «estrutura», no sentido de distinguir aquelas particularidades na semântica das designações em causa. Com a utilização de «estruturas» o Sistema Gráfico fica com um conhecimento em geral mais próximo do modelo da aplicação do que teria usando uma segmentação linear (mais apropriada a uma normal *display-file*). Como se verá na secção 2, tal uso não é de forma alguma restritivo.

Quando, por outro lado, falamos em Computação Gráfica Interactiva, queremo-nos referir ao caso especial em que o utilizador pode controlar dinamicamente as imagens produzidas no contexto da execução de um programa. De igual modo, é tarefa do Sistema Gráfico a gestão do uso dos dispositivos de entrada no sentido de providenciar o acesso às estruturas de dados que fazem parte do modelo da aplicação.

De entre os seis dispositivos lógicos de entrada que GKS (ou PHIGS) colocam à disposição do programador (ie, *Choice*, *Pick*, *Locator*, *Stroke*, *Valuator* e *String*), é *Pick* aquele que permite aceder ao segmento (ou estrutura) que o utilizador haja seleccionado apontando a respectiva imagem, presente no ecrã da estação de trabalho em uso. Com este mecanismo se relacionam entre si as entradas e as saídas gráficas, embora numa forma extremamente dependente da representação interna própria do Sistema Gráfico e não com a versatilidade desejável do ponto de vista da aplicação.

2. Identificação dos Objectos

Sempre que se trate de uma aplicação em que os objectos são simples, ou seja, na medida em que eles não sejam decomponíveis, a codificação que o Sistema Gráfico use para cada um deles (nesse caso segmento ou estrutura) será igualmente adequada à funcionalidade da aplicação propriamente dita. Para tal bastará existir um mapa de conversão (ie, uma correspondência biunívoca) entre os códigos utilizados, conforme se ilustra na figura 1. Para a algoritmia da aplicação é, pois, irrelevante o tipo de representação dos objectos pela parte do Sistema Gráfico.

Todavia, o problema complica-se quando os objectos em causa são complexos e podem construir-se com qualquer número de instâncias de outros. É este o caso mais geral, pois raras são as entidades de um modelo que se apresentam ao programador como indivisíveis. Torna-se agora

evidente que a segmentação a um único nível, como a que existe em GKS, não é a mais adequada ao modelo da aplicação.

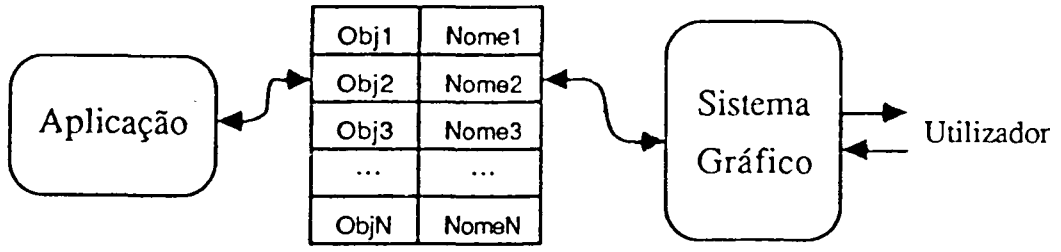

Figura 1 — Correspondência entre nomes externos e internos.

Uma técnica usada frequentemente baseia-se na qualificação dos nomes dos objectos. Assume-se que uma dada composição se pode exprimir por um modelo hierárquico e, portanto, o nome de qualquer elemento dessa hierarquia (excepção obviamente feita à raiz) poderá ser qualificado com o nome do respectivo predecessor. É o que acontece no caso de PHIGS, sistema que já inclui, portanto, facilidades de modelação em que uma aplicação se poderá basear.

Porém, como são possíveis várias ocorrências (instâncias) de uma mesma entidade, este método não é suficiente para identificar univocamente um objecto gráfico apontado pelo utilizador final da aplicação. Atente-se, por exemplo, na impossibilidade de separação das duas instâncias da entidade **e** na figura 2(b), caso que não ocorreria na situação da figura 2(a). Para resolver esta questão, usam-se geralmente artifícios (e.g. inclusão de identificadores de selecção inventados pelo programador e/ou número de ordem dentro da estrutura corrente) difíceis, senão mesmo impossíveis de relacionar com o modelo duma aplicação.

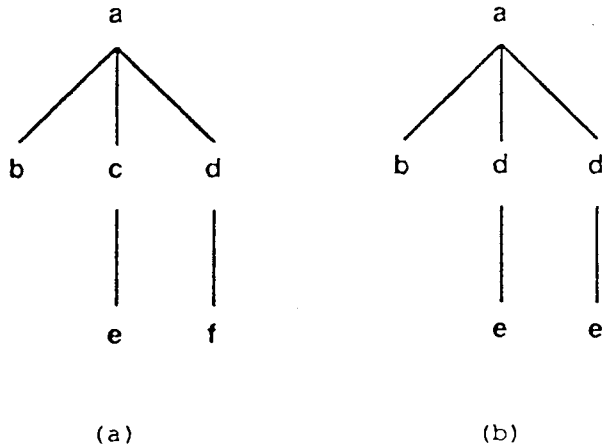

Figura 2 — Hierarquia de entidades por assemblagem.

No sentido de generalizar o processo de identificação, entendido como o relacionamento das entradas com as saídas gráficas, usámos uma estratégia de qualificação baseada nas próprias matrizes de instanciação. Isto significa que a designação de um determinado objecto gráfico, além do nome que foi dado à família de entidades que o gerou e do nível dentro

da hierarquia (dado pelo comprimento da lista de qualificadores), passará a depender também da posição desse mesmo objecto em cada um dos referenciais cartesianos em causa. Assim, mesmo que se apresentem ao utilizador duas instâncias como as da família **e** na figura 2(b), a selecção de uma delas corresponderá a um único nome (na forma *e::T1@T2* em relação a **a**, em que as variáveis *T1* e *T2* representam matrizes de instanciação e os restantes símbolos são dois operadores infixos). É claro que este método só será exequível se tivermos a possibilidade de manipular eficientemente uma matriz, quer como operador, quer como símbolo. Usando Prolog como linguagem de implementação, o desígnio anterior não constitui qualquer problema, uma vez que, além de possuir poder de cálculo, se trata de uma linguagem especialmente indicada para tratamento simbólico [PER86].

Segundo a metodologia do sistema de modelação concebido com base na lógica de primeira ordem e denominado Sistema de MGL, só poderão ser visualizados os objectos que pertençam a uma hierarquia cuja raiz seja uma cena. Esta é, pois, o objecto complexo de mais alto nível e de existência obrigatória. Podem definir-se simultaneamente várias cenas, partilhando ou não as mesmas famílias, sendo a descrição de cada uma representada então numa base de dados lógica por um certo conjunto de cláusulas. Para tal são usados os predicados que a seguir e muito sumariamente se descrevem, identificados com a respectiva aridade: *scene/2*, que relaciona o sistema de coordenadas de modelação com o utilizado pelo sistema gráfico e associa, à relação, o nome duma cena (em GKS, por exemplo, a transformação de normalização); *attribute/2*, que, para uma determinada família, define o valor de um atributo; *instance/4*, que define a associação estrutural entre duas famílias, especificando a matriz de instanciação e, eventualmente, um ou mais valores de atributos próprios da instanciação; *graphics/2*, que declara a forma procedimental que o sistema gráfico usará para a representação gráfica de uma família; *shown/1*, que tem que ser demonstrado para que o sistema gráfico execute o(s) procedimento(s) de visualização sobre uma família de entidades. No âmbito da presente comunicação não é indispensável entrarmos em mais detalhes no que se refere a esta colecção de predicados. Alguns deles aparecerão em exemplos mais adiante no texto. Em todo o caso, para o leitor que deseje uma informação mais completa, aconselha-se a consulta de [PRE87] ou [PRÓ88].

De notar que os nomes dos objectos gráficos não se encontram explícitos na base de dados, sendo obtidos por derivação sempre que necessário. Para esse efeito é calculado o fecho transitivo da relação *instance* usando um predicado *part_of*. Embora não tenha sido programado exactamente como adiante se transcreve, a semântica declarativa deste predicado anti-reflexivo é a mesma das cláusulas indicadas:

```
part_of(Comp::T,Obj) :- instance(Comp,Obj,T,_).

part_of(Comp::T1@T,Obj::T) :- instance(Comp,Obj,T1,_),
                              part_of(Obj::T,X).

part_of(Comp::T1@T,Obj) :- instance(Comp,Med,T1,_),
                           part_of(Med::T,Obj).
```

3. Classificação

Quando se fala em classificação, pensa-se num processo que envolve a atribuição de nomes a determinados objectos e de acordo com certas regras. Uma tal operação corresponde a um agrupamento de objectos com alguma semelhança entre si e no que chamamos uma «classe». É o conjunto de características próprias de uma classe que tornará exequível aquele processo de reconhecimento quando aplicado a dados concretos. As regras que devem reger a elaboração de um programa de classificação terão, pois, que assentar numa descrição fornecida para cada classe.

Actualmente é bem conhecida a utilidade de uma classificação no que diz respeito aos chamados sistemas periciais. Com efeito, imaginemos que se depara a um destes sistemas uma determinada situação cuja resolução dependa da avaliação de certas regras sobre objectos. Para esta última operação haverá certamente conceitos que o programa, não só seja obrigado a conhecer, como deles se servir para identificar, de entre os dados fornecidos, objectos que lhes correspondam. É até muito provável que determinados conceitos resultem de propriedades geométricas que os objectos possuam. Se assim for, poderá ser vantajosa para a comunicação com o utilizador a existência de uma interface gráfica. É então imprescindível que o modelo de uma tal aplicação seja suficientemente flexível para, além da visualização, permitir também um processo de classificação baseado naquelas propriedades.

No sistema de MGL, anteriormente referido, os objectos gráficos têm uma descrição que, à partida, é dotada de uma grande flexibilidade e independência da representação gráfica interna. Por outro lado, é automaticamente exercido um grande controlo sobre as relações entre os objectos e a sua localização (absoluta ou relativa) no espaço. Somos então levados a tentar escrever, com ajuda desse sistema, programas que identifiquem determinadas classes de objectos tendo por base propriedades geométricas estruturais de um modelo. É um exercício deste tipo que nos propomos resolver nas secções que se seguem.

Antes, porém, façamos ainda uma observação que se nos afigura útil. É que, dado o protótipo do sistema de MGL ter sido construído em 2D, será este o espaço dimensional que usaremos na modelação. Contudo, os esquemas de representação em 3D seriam simples extensões dos bidimensionais.

4. A descrição de um arco

Na figura 3 representa-se uma rede semântica para descrever o conceito de «arco» construído a partir de blocos (segundo [WIN75, RIC83]). Uma rede semântica é um mecanismo que permite a representação do conhecimento na forma declarativa e é uma ferramenta considerada fundamental no desenvolvimento de um grande número de aplicações importantes na actualidade, como sejam os já referidos sistemas periciais [SAL87].

No nosso caso, a rede semântica foi aplicada na definição de todas propriedades julgadas necessárias para caracterizar um arco, em termos de relações existentes entre três blocos (*A*, *B* e *C*), os quais, por sua vez, fazem parte da estrutura de um objecto complexo *X*.

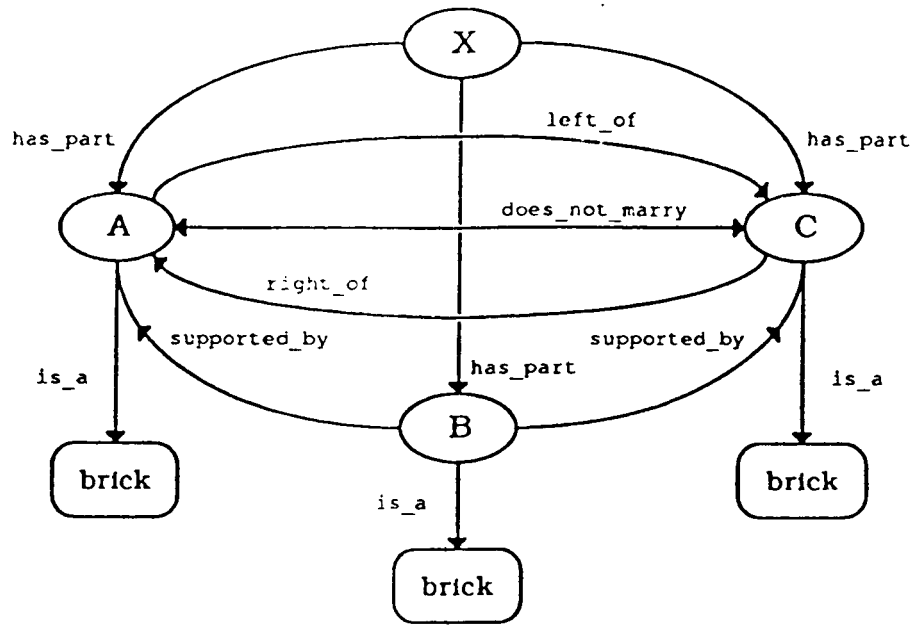

Figura 3 — Descrição de um arco X.

Como o poder expressivo das redes semânticas é equivalente ao da lógica de predicados de primeira ordem [DEL79], facilmente convertemos aquela representação na forma clausal que nos interessa para a programação propriamente dita.

5. Criação de uma Base de Dados Lógica inicial

Da observação da figura 3 concluímos que será necessária a existência de três objectos que possam ser reconhecidos como blocos de construção (*brick*). Na situação mais geral, o reconhecimento de um arco *X* através do lançamento de um golo

```
?- is_a(X,arch).
```

envolveria um segundo nível de reconhecimento, na forma *is_a(Y,brick)*. Por comodidade de apresentação, iremos supor que são impostas três famílias (*p*, *q* e *r*) para as quais, à partida, se definiram as seguintes asserções: .

```
is_a(p,brick).
is_a(q,brick).
is_a(r,brick).
```

Cada instância de qualquer uma destas famílias supomos ter uma representação gráfica que, como nos iremos restringir a 2D, será por hipótese um rectângulo. Em termos dos predicados próprios do sistema de MGL, a base de dados lógica é então acrescida com a cláusula

```
graphics(F,g_fa([0:0,1:0,1:3,0:3])) :- is_a(F,brick).
```

Usou-se a primitiva *Fill Area*, sendo *X:Y* a notação empregue para definir as coordenadas de um ponto num referencial cartesiano. Qualquer bloco é, assim, uma área rectangular de dimensões 1x3.

Para que a regra anterior seja interpretada pelo sistema de modo a construir a respectiva imagem no ecrã da estação de trabalho, terão ainda que existir os factos

```
shown(p).
shown(q).
shown(r).
```

Faltarão agora apenas as relações estruturais entre famílias. No caso de *p*, *q* e *r* constituírem um objecto complexo, por exemplo, da família *obj*, terão que ser criadas cláusulas unitárias para o predicado *instance* que definam tal associação. São do tipo *instance(W,obj,T,L)*, em que o primeiro argumento será *p*, *q* ou *r*, a variável *T* é substituída por uma matriz de instanciação e *L* é uma lista de atributos, eventualmente vazia. Esses atributos poderiam influenciar bastante o aspecto final da imagem, mas, não sendo esse o tema central desta comunicação, não lhe dispensaremos qualquer atenção. Também aqui não são relevantes as facilidades de criação de certo tipo de cláusulas na base de dados lógica, como seja o caso das relações estruturais. Na verdade, e contrariamente ao que poderia parecer à primeira vista, não se obriga o programador a editar matrizes directamente. Uma tal operação pode ser muito facilmente indicada usando alguns dos predicados de mais alto nível implementados no sistema de MGL, ou transformando geometricamente os objectos por métodos interactivos [PRÓ88].

Para que o modelo possa ser visualizado, imaginemos ainda uma instanciação de *obj* numa determinada cena, de acordo com a obrigatoriedade referida anteriormente na secção 2. Ficaremos, então, com um modelo hierárquico a dois níveis, de que a cena é a raiz.

6. Um programa de reconhecimento

Nesta altura existem na base de dados lógica as cláusulas suficientes para que possam ser visualizados no ecrã três rectângulos. Sobre eles poderão ser executadas diversas operações, entre as quais transformações geométricas que se reflectirão nas correspondentes matrizes de instanciação. No entanto, não existe ainda qualquer informação específica que viabilize um processo de reconhecimento, ou seja, a descrição de um conceito.

Consideremos, por isso, a rede semântica da figura 3. Todo o objecto *X* que satisfaça as relações aí presentes é, por hipótese, classificado como arco. O que há a fazer é converter essa representação numa descrição em Prolog. Uma das vantagens do uso da programação em lógica neste caso é que as descrições, como são executáveis, serão elas próprias um programa de classificação.

Começaremos por determinar se um dado objecto gráfico é ou não um bloco de construção, o que se traduz simplesmente pela cláusula

```
is_a(C::_,brick) :- is_a(C,brick).
```

Bastará, em seguida, traduzir as relações da figura 3 para um programa Prolog, tarefa que, a esse nível, é praticamente imediata:

```
is_a(X,arch) :- has_part(X,A),
                is_a(A,brick),
                has_part(X,B),
                B \== A,is_a(B,brick),
                has_part(X,C),
                C \== A,C \== B,is_a(C,brick),
                supported_by(B,C),
                supported_by(B,A),
                left_of(A,C),
                right_of(C,A),
                does_not_marry(A,C).
```

Em linguagem corrente isto significa que um objecto complexo *X* é classificado como arco se for composto por três blocos distintos *A*, *B* e *C* de tal modo que *B* seja suportado pelos outros dois, encontrando-se *A* à esquerda e *C* à direita, mas não demasiadamente perto um do outro.

Como *A*, *B* e *C* correspondem a variáveis, houve necessidade de incluir restrições que se encontravam implícitas na figura 3, ou seja, providenciar que essas variáveis fiquem ligadas a objectos distintos.

Repare-se que a descrição obtida pode ser vista de forma completamente independente duma representação gráfica. Esta será apenas introduzida nos procedimentos para os predicados encontrados em tal descrição.

No caso concreto e como foi já exemplificado, o primeiro argumento de *is_a* passará agora a ser um objecto gráfico, cujo nome está portanto sujeito à sintaxe que foi exposta na secção 2.

Quanto à relação *has_part* é uma forma particular de *part_of*, pelo que a sua programação origina a cláusula

```
has_part(F::T,C::tm(M1,M2,M3,M4,M5,M6)) :-
                part_of(C::tm(M1,M2,M3,M4,M5,M6)@T,F::T).
```

Em 2D linearizamos uma matriz de instanciação 3x3 incluindo somente os seus seis elementos característicos das transformações geométricas possíveis [ISO85]. O resultado é apresentado pelos argumentos do símbolo funcional *tm*.

As outras relações que figuram na descrição de um arco não encontram, porém, expressão directa em predicados do sistema de MGL. Mas este é suficientemente flexível e geral para permitir a sua fácil implementação. A solução mais simples será o recurso à noção de *extent*, a construir com base na cláusula de definição do predicado *graphics*. O acesso a este predicado, lançando-o como golo com vista à extracção do segundo argumento totalmente instanciado, é um processo usual no sistema de MGL para se obterem as primitivas gráficas responsáveis pelo traçado de uma determinada imagem e que difere bastante duma aproximação estritamente procedimental.

Um predicado *extent* possuirá então três argumentos, sendo o primeiro o nome do objecto e os outros dois o ponto inferior esquerdo e o superior direito da área rectangular abrangida, por esta mesma ordem. Assim, para

conhecermos a lateralidade entre dois objectos L e R servimo-nos das cláusulas

```
left_of(L,R) :- extent(L,_,LXmax:_),
                extent(R,RXmin:_,_),
                LXmax<RXmin.

right_of(R,L) :- left_of(L,R).
```

Por outro lado, para se programar a relação *supported_by* é necessário fazer referência à precisão com que as imagens dos objectos são dispostas no ecrã. O predicado *almost_equal* estabelecerá essa condição, pelo que se tem:

```
supported_by(Top,Bot) :- extent(Top,TXmin:TYmin,TXmax:_),
                         extent(Bot,BXmin:_,BXmax:BYmax),
                         almost_equal(TYmin,BYmax),
                         (TXmin=<BXmax,BXmin=<TXmax,!;
                          TXmax>=BXmin,BXmax>=TXmin).
```

Assim, pode não ser importante uma separação visivelmente existente entre os objectos *Top* e *Bot*, como acontece no caso da figura 4(a).

Uma tolerância também terá que ser definida sempre que se trata de saber em que medida duas imagens se encontram demasiadamente próximas uma da outra, no sentido dado a *does_not_marry*:

```
does_not_marry(D,E) :- extent(D,DXmin:_,DXmax:_),
                       extent(E,EXmin:_,EXmax:_),
                       abs(DXmax-EXmin,K1),K1>0.5,
                       abs(DXmin-EXmax,K2),K2>0.5.

abs(X,P) :- P is X,P>0,!.
abs(X,P) :- P is -X.
```

O programa acabado de descrever pode aplicar-se a qualquer estado da base de dados lógica (a que corresponde um determinado conjunto de informações no ecrã), desde que se lance um golo da forma *is_a(X,arch)* e independentemente da variável se encontrar livre ou instanciada.

Suponhamos, a título de ilustração, que um utilizador, executando um determinado programa que lhe permita a interacção com os objectos presentes no ecrã [PRÓ88], modifica a posição relativa existente entre os blocos mediante translações e rotações arbitrárias. As figuras 4 e 5 dão conta de alguns exemplos possíveis.

A instanciação da família *obj* é efectivamente reconhecida como um arco em qualquer dos casos apresentados na figura 4 (admitindo as tolerâncias usadas), mas já nenhuma das que a figura 5 inclui está de acordo com a descrição inicialmente dada para esse conceito. Por exemplo, na figura 5(a) as partes *A* e *C* estão demasiadamente próximas, enquanto que nas outras duas a parte *B* não é devidamente suportada.

É claro que a descrição de partida poderá evoluir, correspondendo às necessidades da aplicação ou à avaliação experimental. Talvez, por

exemplo, permitindo disposições de blocos idênticas à da figura 5(c) ou excluindo as do género da figura 4(c).

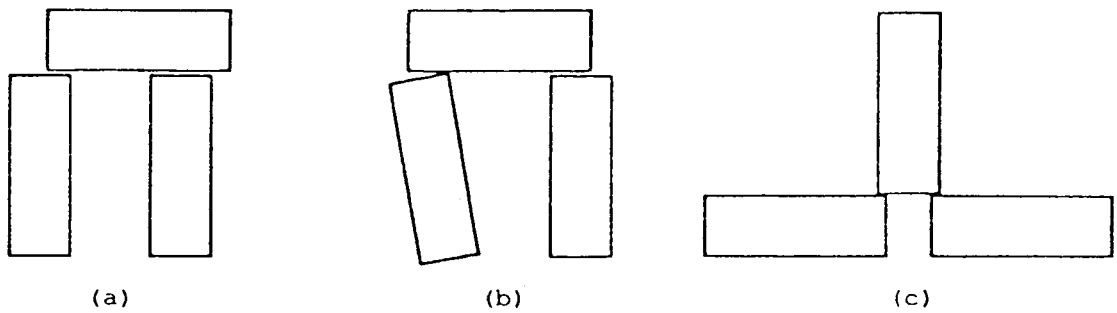

Figura 4 — Objectos complexos reconhecidos como arcos.

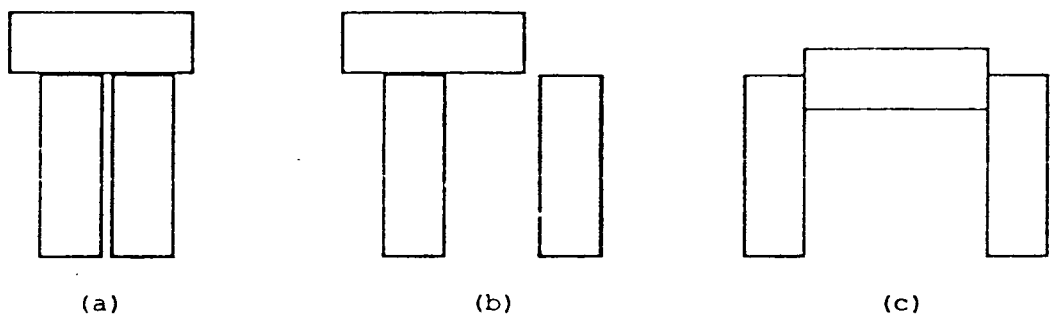

Figura 5 — Objectos complexos não reconhecidos como arcos.

Por observação das figuras 4 e 5 não é possível distinguir, de entre *p*, *q* e *r*, qual a família a que corresponde um dado rectângulo (bloco). Nem isso é importante para o processo de reconhecimento que foi exposto: qualquer permutação dos três blocos daria sempre os mesmos resultados. Em todo o caso, pode ser útil para a aplicação associar a cada um dos blocos valores de atributos (e.g. cor) que alteram a imagem com que eles se apresentam ao utilizador. Esse tipo de actualização do modelo nada tem a ver com a classificação apresentada (apenas estrutural, no nosso exemplo), mas poderia eventualmente interessar a outra aplicação que redefiniria, nesse sentido, o predicado *is_a*.

7. A concluir

A classificação de objectos representados graficamente num ecrã pode ser de extrema importância para determinadas aplicações. Obviamente que as imagens visíveis apenas são úteis como forma de comunicação com o utilizador. Porém, em sistemas interactivos, essas mesmas imagens podem servir como um meio natural de modificação indirecta do modelo duma aplicação. Normalmente o sistema gráfico, mesmo possuindo capacidades de modelação, não tem a flexibilidade suficiente para analisar os dados presentes sob o ponto de vista de um processo de classificação como pretendemos. Este facto resulta principalmente do carácter

estritamente procedimental dos programas que usualmente se constroem para resolver problemas gráficos.

O sistema de MGL, além de responder adequadamente aos aspectos procedimentais exigidos à programação, apresenta características que permitem um estilo predominantemente declarativo. O programador encontra-se então muito mais perto da essência do problema a ser resolvido.

Um outro exemplo da flexibilidade conseguida é mostrado pela seguinte variante do exercício sobre arcos resolvido na secção 6: em vez de se obrigar a que o programa tenha como espaço de pesquisa uma colecção de objectos compostos, é inclusivamente mais geral a procura de objectos gráficos que, independentemente das relações estruturais próprias do modelo, se encontrem numa certa disposição espacial. Na verdade é isso o que o utilizador vê e não a representação interna do modelo, que segue critérios de implementação raras vêzes deixados transparecer pela interface. É óbvio que a contrapartida é dada pelo perigo de explosão combinatória no espaço de pesquisa se se estiver na presença de grande número de objectos candidatos.

No caso do arco bastaria alterar a definição de *has_part* para se poder contemplar qualquer grupo de objectos numa cena, constituindo eles ou não um objecto previamente declarado como complexo:

```
has_part(_,C) :- part_of(C,S),scene(S,_).
```

Naturalmente que o primeiro argumento deixa de ter interesse, agora substituído por uma variável anónima apenas para manter a aridade do predicado. E se fosse importante a determinação de alguma solução, em vez duma resposta em termos de valores de verdade, esta teria que ser dada pelas instanciações das variáveis *A*, *B* e *C* presentes no corpo da cláusula cuja cabeça é *is_a(X,arch)*, na secção 2. Poderia assim, com vantagem, substituir-se este implicando por um termo *shape_of([A,B,C],arch)*, correspondendo a um novo predicado.

O exemplo que serviu de referência a este texto foi efectivamente implementado. O equipamento utilizado consistiu numa estação de trabalho VAXstation II, pertencente ao Grupo de Programação em Lógica e Inteligência Artificial do Departamento de Informática da UNL. Um protótipo do sistema de MGL foi escrito em C-Prolog [PER84] e desenvolvido no âmbito do projecto ALPES [PRE87], usando como Sistema Gráfico de base o pacote VAX/GKS-2b e funcionando com o sistema operacional MicroVMS.

Referências

[DEL79] Deliyanni, A.; Kowalski, R.
Logic and Semantic Networks, *Communications of the ACM*, 22(3) pp 184-192 (March 1979)

[ISO85] International Organization for Standardization
Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description, ISO IS 7942 (July 1985)

[ISO86] International Organization for Standardization
Information processing systems — Computer graphics — Programmer's Hierarchical Interactive System (PHIGS) Part 1 — functional description, ISO dp9592/1 (October 1986)

[ISO87] International Organization for Standardization
Information processing systems — Computer graphics — Graphical Kernel System for Three Dimensions (GKS-3D) Functional Description, ISO DIS 8805 (1987)

[PER84] Pereira, F.C. *et al*
C-Prolog version 1.5 User's Manual, EdCAAD, University of Edinburgh (1984)

[PER86] Pereira, L.M.; Porto, A.
Vantagens da programação em lógica para sistemas de conhecimentos, em *Actas do 4º Congresso Português de Informática*, pp 118-129 (Junho 1986)

[PRE87] Preston, N.; Próspero, M.J.; Gandilhon, T.
Prolog and Graphics — Specification, Deliverable for WP3.1, ESPRIT project ALPES-p973 (September 1987)

[PRÓ88] Próspero, M.J.
Um estilo declarativo na Programação Gráfica Interactiva: análise e avaliação baseadas em sistemas construídos com Prolog, Dissertação — UNL (em vias de publicação) (1988)

[RIC83] Rich, E.
Artificial Intelligence, McGraw-Hill (1983)

[SAL87] Salzberg, S.
Knowledge Representation in the Real World, *AI Expert*, 2(8) pp 32-38 (August 1987)

[WIN75] Winston, P.H.
Learning Structural Descriptions from Examples, in *The Psychology of Computer Vision*, McGraw-Hill (1975)