Pràctica 3: Codificació en SAT

Lògica en la Informàtica

FIB

Antoni Lozano

Q1 2023–2024

Objectius

Aquesta segona pràctica té com a objectius:

- Fer ús d'un *SAT-solver* (**kissat**) per resoldre problemes combinatoris.
- Generar la codificació del problema en SAT mitjançant un programa en Prolog.
- Aprendre a utilitzar la llibreria de Prolog que es proporciona com a ajut per la codificació.

Referències

Com a guia d'estudi per a aquesta pràctica teniu

- El fitxer **miSudoku.pl**, que serveix d'exemples per la resta de problemes
- El **Makefile** que es proporciona
- Aquestes transparències

Codificació en SAT

Sudoku

```
%% 4. Resolució de problemes combinatoris codificant en SAT
%% i usant Prolog com a llenguatge de modelatge

%% Farem servir el problema d'omplir un Sudoku com a exemple: miSudoku.pl

%% Què fa miSudoku.pl?

%% > ./miSudoku
%% Generated 12016 clauses over 729 variables.
%% Calling solver....
%% Solution found:

%% 6 4 3  9 7 5  2 1 8
%% 2 7 8  4 3 1  9 5 6
%% 5 1 9  6 8 2  3 4 7

%% 3 2 5  8 6 4  1 7 9
%% 1 8 7  3 5 9  4 6 2
%% 9 6 4  2 1 7  5 8 3
```

Sudoku

El programa

1. genera i desa en un fitxer `.cnf` les clàusules corresponents a les restriccions del problema,
2. crida al *SAT-solver* kissat amb `infile.cnf` i desa el resultat en un fitxer `model`,
3. obté un model simbòlic M, que conté les variables certes de la solució en format simbòlic i
4. mostra la solució de forma llegible amb `displaySol`.

Codificació

Per codificar un problema només heu de modificar tres punts de l'esquema:

1) definir SAT-variables (moltes vegades ja vénen donades, o donem algunes SAT-variables,
i cal definir-ne alguna altra)

2) generar les clàusules corresponents a les restriccions: completar el predicat
writeClauses/0. *No el confoneu amb writeClause/1*, que afegeix una clàusula al fitxer clauses

3) escriure o completar displaySol(M) on M és la llista de variables simbòliques [SAT-variables]
que són *certes* en el model retornat per kissat (sovint també ve donat)

Codificació en SAT

Exemples

```
%% 2. Predicat between

%% between(L, U, X) és cert quan X és un enter entre els enters L i U.

%% ?- between(1, 3, 2).
%% true.

%% ?- between(1, 3, X), write(X), nl, fail.
%% 1
%% 2
%% 3
%% false.
```

Exemples

```
%% 3. Predicat findall

%% findall(X, G, L) és cert quan L és la llista del X per als que
%% (hi ha manera d'instanciar les altres variables, si n'hi ha)
%% l'objectiu G és cert

% f(a, b, c).
% f(a, b, d).
% f(b, c, e).
% f(b, c, f).
% f(c, c, g).

%% ?- findall(C, f(A, B, C), Cs).
%% Cs = [c, d, e, f, g].

%% ?- findall(C, f(a, B, C), Cs).
%% Cs = [c, d].
```

Exemples

```
%% Hi ha altres predicats semblants al findall: setof, bagof. Però tenen
%% el problema que, si no hi ha solucions per a G, aleshores fallen, en
%% lloc de considerar que L ha de ser la llista buida.
%% Això sol donar problemes, i no en recomanem l'ús.


%% ?- findall(A, f(A, b, C), As).
%% As = [a, a].

%% Pot ser que hi hagi solucions repetides!
%% Si no volem repeticions, podem fer servir el predicat sort,
%% que ordena *i elimina repeticions*:

%% ?- | sort([1,3,2], X).
%% X = [1, 2, 3].

%% ?- sort([1,3,1,3,2,2], X).
%% X = [1, 2, 3].
```

Exemples

```
%% Més exemples, combinant findall i between:

%% ?- findall(X, (between(1,7,X), 1 is X mod 2), L).
%% llista dels senars entre 1 i 7

%% L = [1, 3, 5, 7].

%% ?- findall(Y, (between(1,7,X), 1 is X mod 2, Y is X*X), L).
%% llista dels quadrats dels senars entre 1 i 7

%% L = [1, 9, 25, 49].
```

Codificació en SAT

Ús de la llibreria

Des de l'intèrpret de Prolog, la generació de clàusules s'inicia amb la consulta

```
?- main.
```

Per veure les clàusules generades en format simbòlic, canvieu al fitxer `symbolicOutput(0)` per `symbolicOutput(1).`

(El predicat es troba normalment a l'inici)

Predicats de generació de clàusules

- `writeOneClause(L)`: genera la clàusula donada per la llista `L`
- Cardinalitat. Generació de clàusules equivalents a
  - `exactly(K,L)`: exactament `K` literals de `L` són certs
  - `atMost(K,L)`: com a màxim `K` literals de `L` són certs
  - `atLeast(K,L)`: com a mínim `K` literals de `L` són certs
- Equivalència. Genera clàusules que estableixen que la variable `V` equival a
  - `expressOr(V,L)`: la disjunció de les clàusules de `L`
  - `expressAnd(V,L)`: la conjunció de les clàusules de `L`

Predicat `exactly`

Està definit com:

```
exactly(K,Lits):-
        symbolicOutput(1),
        write( exactly(K,Lits) ), nl, !.

exactly(K,Lits):-
        atLeast(K,Lits),
        atMost(K,Lits), !.
```

És a dir, amb

- `symbolicOutput(1)`, s'escriu la restricció lògica que estem afegint com a entrada del SAT solver i
- `symbolicOutput(0)`, es generen clàusules que asseguren que almenys `K` literals de `Lits` són certs i que com a màxim `K` literals de `Lits` són certs.

Predicat `atMost`

```
atMost(K,Lits):- symbolicOutput(1), write( atMost(K,Lits) ), nl, !.
atMost(K,Lits):-   % l1+...+ln <= k:  in all subsets of size k+1, at least one is false:
    negateAll(Lits,NLits),
    K1 is K+1,    subsetOfSize(K1,NLits,Clause), writeOneClause(Clause),fail.
atMost(_,_).
```

Predicat `atLeast`

```
atLeast(K,Lits):- symbolicOutput(1), write( atLeast(K,Lits) ), nl, !.
atLeast(K,Lits):-  % l1+...+ln >= k: in all subsets of size n-k+1, at least one is true:
    length(Lits,N),
    K1 is N-K+1,  subsetOfSize(K1, Lits,Clause), writeOneClause(Clause),fail.
atLeast(_,_).
```