In the Lecture Series Logic Programming and Constraints

Non Logical

Presented by Stéphane Bressan

References

- Chapters 9, 10, 11 and 21 of the user manual
- Self–study chapter 20

Global Data Structures

Normal Logic Programming does not allow to share information across parallel branches of the SLD tree and to get information from failing branches

To do so, we need global data structures.

Global Data Structures

Facility	Type	Access	See
shelves	array	by handle	shelf_create/2,3
bags	unordered bag	by handle	bag_create/1
stores	hash table	by handle	store_create/1
named shelves	array	by name	shelf/2
named stores	hash table	by name	store/1
non-logical variables	single cell	by name	variable/1
non-logical arrays	array	by name	array/1,2
records	ordered list	by name	record/1,2
dynamic predicates	ordered list	by name	dynamic/1,assert/1

Non-logical Variables

Non-logical variables are global (to a module) named variables (an atom) with destructive assignment

- It is good style to declare variables
  - :- local variable(count).
- A value, a term, is assigned to the variable
  - setval(+ElemSpec, ?Value)
- The value is retrieved
  - getval(+ElemSpec, ?Value)

Non-logical Variables

If the value is a non ground term, the retrieved value is a variant (i.e. different variables)

```
:- V = f(X, Y, X),
   setval(test, V),
   getval(test, V1),
   getval(test, V2),
   writeq([V, V1, V2]).
```

Non-logical Variables

```
count_solutions(Goal, _) :-
  setval(count, 0),
  call(Goal),
  incval(count),
  fail.
count_solutions(_, N) :-
  getval(count, N).
```


Bags

- A bag is an object which can be used to store terms across failures.
  - A bag is created and a hnadle stored in a variable
    - bag_create(-BagHandle)
  - Terms are entered in the bag and remain there upon failure
    - bag_enter(+BagHandle, +Term)
  - The content of the bag can be retrieved
    - bag_retrieve(+BagHandle, ?List)
  - The content of the bag can be retrieved and the bag destroyed
    - bag_dissolve(+BagHandle, ?List)

Bags

```
simple_findall(Goal, Solutions) :-
  bag_create(Bag),
      ( call(Goal),
      bag_enter(Bag, Goal),
      fail ;
      true ),
  bag_dissolve(Bag, Solutions).
```


Dynamic Predicates

Dynamic predicates definitions (part of the program) can be modified at run time

- The dynamic predicate must be declared
  - dynamic(+PredSpec)
- A new clause can be addedd
  - assert(+Clause)
- Existing clauses can be removed
  - retract(+Clause) (Unification!)
- Exiting clauses can be printed
  - listing +SpecList
- See also asserta/1, compile_term/1

Dynamic Predicates

```
:- dynamic a/2.
:- assert(a(c, Y)).
:- listing a/2.
:- assert(a(c, d)).
:- assert(a(X, Y) :- writeln(“hello”)).
:- a(X, Y).
:- listing a/2.
:- retract(a(X, Y)).
:- listing a/2.
:- retract(H:-B).
:- retract(a(X, Y):-B).
:- listing a/2.
```


Input and Output

Input and output in ECL*i*PS*e* is done via communication channels called *streams*. They are usually associated with either a file, a terminal, a socket, a pipe, or in-memory queues and buffers. The streams may be opened for input only (*read mode*), output only (*write mode*), or for both input and output (*update mode*).

Input and Output

- Input: 0 (stdin)
- Output: 1 (stdout)
- warning_output: 1 (stdout)
- log_output: 1 (stdout)
- error: 2 (stderr)
- null 3 (null)
- User: 0 or 1 (stdin or stdout)

Input and Output

- open(++Source, +Mode, ?Stream)
  - Opens the I/O source or sink Source in mode Mode and associates it with the stream identifier Stream.
  - SourceSink can be
    - a file (path and name),
    - a string (string(String)), or
    - a queue (queue(String))
  - Mode can be one of
    - read,
    - write,
    - update,
    - Append
- close(+Stream)
  - Closes the stream specified by Stream.

Input and Output

See also: tyi/1, tyi/2, tyo/1, tyo/2, read_string/3, read_string/4, readvar/3, read_term/2, read_term/3, display / 1, display / 2, writeq/1, writeq/2, write_canonical/1, write_canonical/2, nl/0, nl/1, writeln/1, writeln/2, print/1, print/2, printf/2, printf/3, at_eof/1 and set_stream_property/3.

Input and Output

- get(+Stream, -Ascii), get(-Ascii)
  - Reads the next character from the input stream Stream and unifies its ASCII code with Ascii.
- put(+Stream, +Ascii) put(+Stream, +Ascii)
  - The character represented by the ascii integer code Ascii is put onto the buffered output stream Stream.
- read_token(+Stream, -Token, -Class), read_token(-Token, -Class)
  - Succeeds if the next token from the input stream Stream is successfully read and unified with Token and its token class with Class.
- read(+Stream, -Term), read(-Term)
  - Succeeds if the next term from the input stream Stream is successfully read and unified with Term. read(-Term)
- write(+Stream, ?Term), write(?Term)
  - The term Term is written on the output stream Stream according to the current operator declarations.
- seek(+Stream, +Offset)
  - The pointer in stream Stream is offset Offset from the start of the file .
- at(+Stream, -Pointer)
  - Succeeds if Position is the position of the stream Stream.
- pipe(?StreamIn, ?StreamOut)
  - Creates a pipe and two streams StreamIn and StreamOut to its read and write ends

- See also: tyi/1, tyi/2, tyo/1, tyo/2, read_string/3, read_string/4, readvar/3, read_term/2, read_term/3, display / 1, display / 2, writeq/1, writeq/2, write_canonical/2, nl/0, nl/1, writeln/1, writeln/2, printf/2, printf/3, at_eof/1 and set_stream_property/3.

Internet and Inter-process Communication

- Socket Domains
- Stream Connection (internet domain)
- Datagram Connection (internet domain)
- Stream Connection (unix domain)
- Datagram Connection (unix domain)

Internet and Inter-process Communication

- socket(+Domain, +Type, ?Stream)
  - Creates a socket of a given type and domain and associates a stream with it.
  - Domain is unix or internet.
  - Type is stream or datagram.
- accept(+Stream, -From, ?NewStream)
  - Accepts a connection for a stream socket and creates a new socket which can be used for I/O.
- See also bind / 2, listen / 2, connect / 2

Internet Communication

server:
[eclipse 10]: socket(internet, stream, s), bind(s, X). <HostName/Port >

X = m1 / 3789
yes.
[eclipse 11]: listen(s, 1), accept(s, From, news). <blocks waiting for a connection>

client:
[eclipse 26]: socket(internet, stream, s), connect(s, m1/3789).

yes.
[eclipse 27]: printf(s, "%w. %b", message(client)), read(s, Msg).

server:
From = m2 / 1627
yes.
[eclipse 12]:
read(news, Msg), printf(news, "%w. %b", message(server)).

Msg = message(client)
yes.

client:
Msg = message(server)
yes.

Credits

Clipart and media are licensed from Microsoft Office Online Clipart and Media