NOPT042 Constraint programming: Tutorial 1 – Introduction to Picat

- See the tutorial website for program of classes, credit requirements, and a list of useful resources.
- Check out our GitHub repository for the notebooks, code examples, and solutions to some exercises (will be updated throughout the semester).
- Join our ReCodEx group for homework assignments.

What was in the lecture?

This tutorial does not follow the contents of the lecture. This is by design:

- In the lecture, we learn how constraint solvers work.
- In the tutorial, we learn how to create efficient constraint models, how to think in "declarative", "constraint programming" mode.

While in your future careers you are much more likely to be using an industry-grade constraint solver than to be developing one, being able to write efficient models requires sufficient understanding of the solver's inner workings.

However, at the beginning of each tutorial we will briefly review what was covered in the last lecture.

What was in Lecture 1?

- course overview
- history
- examples (Sudoku, map coloring, N-queens, ...)
- applications
- how CP fits within optimization
- the definition of the CSP, basic terminology
- advantages and limitations
- binarization of constraints

In this tutorial:

- getting started with the Picat programming language
- overview of Picat in general (not specific to constraint programming)
- a demonstration of basic constraint solving in Picat

About Picat

Picat is a logic-based multiparadigm general-purpose programming language.

- **Pattern-matching**: predicates defined with pattern-matching rules
- **Intuitive**: incorporates declarative language syntax, e.g. for scripting, mimics for-loops, ...
- **Constraints**: designed with constraint programming in mind, provides 4 solvers, `cp, sat, smt, mip`
- **Actors**: action rules for event-driven behaviour; constraint propagators are implemented as actors
- **Tabling**: store subresults, dynamic programming, module `planner`

Why Picat?

Installation

You can install Picat like this (check if there's a newer version of Picat):

```
cd ~
wget http://picat-lang.org/download/picat37_linux64.tar.gz
tar -xf picat37_linux64.tar.gz
```

Then add the executable to `$PATH` (assuming we use bash):

```
echo 'export PATH="$HOME/Picat:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Then the command `picat` runs the Picat interpreter.

If you want to execute the notebooks, install Jupyter Notebook with ipicat extension (if you want to install them locally, add `--user`):

```
pip install jupyter
pip install ipicat
```

Then run `jupyter notebook`. Once the extension is loaded you can use `%%picat` cell magic or execute picat files: `%picat -e hello-world.pi`.

To view the slideshow, install the RISE extension:

```
pip install RISE
```

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.7
```

Introductory examples

Hello, World!

In [2]:
```
%%picat
main =>
    println("Hello, World!").
```

```
Hello, World!
```

Execute a picat file via inline magic:

In [3]:
```
%picat -e hello-world.pi
```

```
Hello, World!
```

Alternatively, via a shell comand:

In [4]:
```
!picat hello-world.pi
```

```
Hello, World!
```

Command-line arguments

In [5]:
```
# This doesn't work at the moment
# %picat -e hello-world.pi Alice
!picat hello-world.pi Alice
!picat hello-world.pi Alice Bob Carol Dave
```

```
Hello, Alice! I love your solution to the homework problem.
Hello, Alice and Bob and Carol and Dave! You are my favourite students.
```

In [6]:
```
!cat hello-world.pi
```

```
import util.

main =>
    println("Hello, World!").

main([Name]) =>
    printf("Hello, %s! I love your solution to the homework problem.\n", Nam
e).

main(ARGS) =>
    Names = ARGS.join(" and "),
    printf("Hello, %s! You are my favourite students.\n", Names).
```

Example: Fibonacci sequence

In Jupyter, use `%%picat -n predicate_name` to define a predicate from a cell.

In [7]:
```
%%picat -n fib
fib(N, F) =>
    if (N = 0) then
```

```
        F = 0
    elseif (N = 1) then
        F = 1
    else
        fib(N - 1, F1),
        fib(N - 2, F2),
        F = F1 + F2
    end.
```

Alternative syntax:

In [8]:
```
%%picat -n fib
fib(0, F) => F = 0.
fib(1, F) => F = 1.
fib(N, F), N > 1 => fib(N - 1, F1), fib(N - 2, F2), F = F1 + F2.
```

But of course, we should use **tabling**!

In [9]:
```
%%picat -n fib_tabled
table
fib_tabled(0, F) => F = 0.
fib_tabled(1, F) => F = 1.
fib_tabled(N, F), N > 1 => fib_tabled(N - 1, F1), fib_tabled(N - 2, F2), F =
```

Compare the performance:

In [10]:
```
%%picat
main =>
    time(fib(42, F)),
    println(F),
    time(fib_tabled(42, F)),
    println(F).
```

```
CPU time 30.553 seconds.

267914296

CPU time 0.0 seconds.

267914296
```

Example: Quicksort

Pattern-matching rules:

In [11]:
```
%%picat -n qsort
qsort([]) = [].
qsort([H | T]) = qsort([E : E in T, E =< H]) ++ [H] ++ qsort([E : E in T, E
```

Alternative version:

In [12]:
```
%%picat -n qsort
qsort(L) = Lsorted =>
    if L = [] then
        Lsorted = []
    else
        L = [H | T],
        Lsorted = qsort([E : E in T, E =< H]) ++ [H] ++  qsort([E : E in T,
```

Try it out:

In [13]:
```
%%picat
main => L = qsort([5, 2, 6, 4, 1, 3]), println(L).
```

```
[1,2,3,4,5,6]
```

Source-file, with string interpolation the output:

In [14]:
```
!picat qsort/qsort.pi
```

```
For example, the list [5,2,6,4,1,3] after sorting is [1,2,3,4,5,6].
```

Command-line arguments:

In [15]:
```
!picat qsort/qsort.pi [5,2,6,4,1,3]
```

```
[1,2,3,4,5,6]
```

Reading and writing files

In [16]:
```
!cat qsort/assorted.lists
```

```
[2, 1]
[5, 2, 6, 4, 1, 3]
[44, 11, 29, 53, 59, 70, 63, 68, 16, 30, 95, 9, 55, 71, 84, 81, 64, 46, 26,
89, 15, 40, 22, 97, 39]
```

In [17]:
```
!picat qsort/qsort.pi qsort/assorted.lists qsort/sorted.lists
!cat qsort/sorted.lists
```

```
[1,2]
[1,2,3,4,5,6]
[9,11,15,16,22,26,29,30,39,40,44,46,53,55,59,63,64,68,70,71,81,84,89,95,97]
```

The source code:

In [18]:
```
!cat qsort/qsort.pi
```

```
qsort([])    = [].
qsort([H|T]) = qsort([E : E in T, E =< H]) ++ [H] ++ qsort([E : E in T, E >
H]).

main =>
    L = [5, 2, 6, 4, 1, 3],
    printf("For example, the list %w after sorting is %w.\n", L, qsort(L)).

main([Lstring]) =>
    L = parse_term(Lstring),
    println(qsort(L)).

main([InputPath, OutputPath]) =>
    Lines = read_file_lines(InputPath),
    OutputFile = open(OutputPath, write),
    foreach(I in 1..Lines.length)
        L = parse_term(Lines[I]),
        writeln(OutputFile, qsort(L))
    end.
```

TPK algorithm

The TPK algorithm is an artificial problem designed by Trabb Pardo & Knuth to showcase the syntax of a given programming language (see Wikipedia):

```
ask for N numbers to be read into a sequence S
reverse sequence S
for each item in sequence S
    call a function to do an operation
    if result overflows
        alert user
    else
        print result
```

The following Picat implementation is from here.

In [19]:
```
!cat tpk/tpk.pi
```

```
% TPK Algorithm in Picat
% from https://www.linuxjournal.com/content/introduction-tabled-logic-progra
mming-picat

f(T) = sqrt(abs(T)) + 5 * T**3.

main =>
    N = 4,
    As = to_array([read_real() : I in 1..N]),
    foreach (I in N..-1..1)
        Y = f(As[I]),
        if Y > 400 then
            printf("%w TOO LARGE\n", I)
        else
            printf("%w %w\n", I, Y)
        end
    end.
```

In [20]:
```
!cat tpk/some_reals.txt
!printf "\n"
!picat tpk/tpk.pi < tpk/some_reals.txt
```

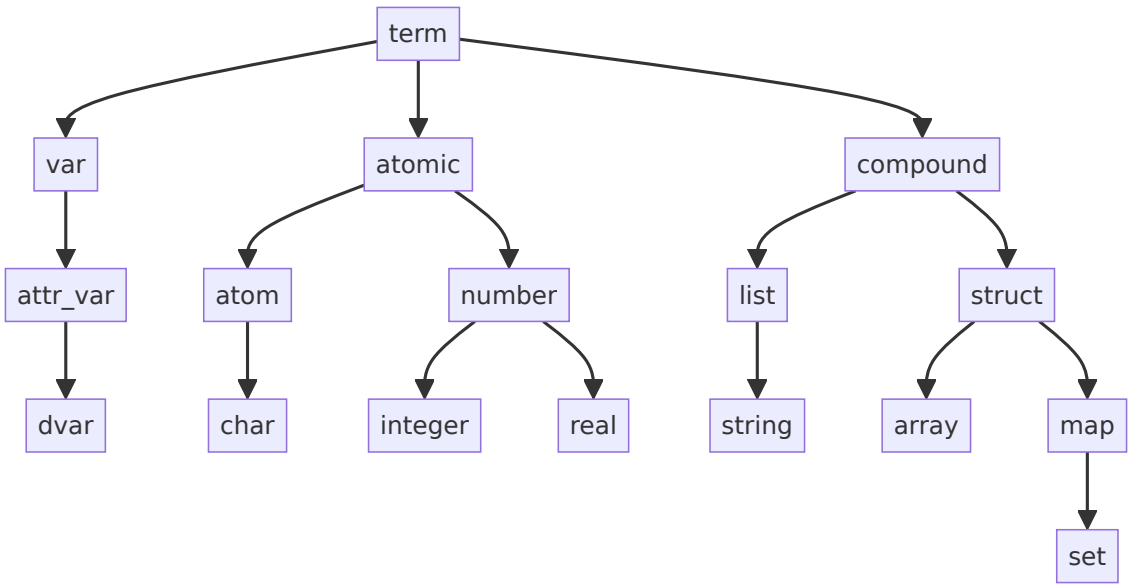
```
1.0e-2
-2.345
42.0001
-0.002
4 0.0447213
3 TOO LARGE
2 -62.9447
1 0.100005
```

An overview of Picat

Examples in this section are mostly adapted from or inspired by the Picat Book, Picat Guide, AAA2017 tutorial, and examples. For a quick overview, see pages 4–18 from Modeling and Solving AI Problems in Picat.

Data types

- Dynamic, runtime typing.
- Everything is a *term* (which can be a variable, or a value: atomic or compound)



Variables

- Start with capital letter or underscore: `var(X)` [yes], `var(_abc)` [no], `var(abc)` [no]
- A variable is *free* until *instantiated* (bound to a value).

- Single-assignment variables: after instantiation, the variable has the same identity as the value: `X=1, var(X)` [no]
- If execution backtracks over the binding point, the variable becomes free again.

Values

Primitive values:

- atom: symbolic constant, either quoted or starts with lower-case letter (incl. char)
- number: integer or real

Compound values:

- list: `L = [1, 2, 3]`, list comprehension: `L = [X : X in 1..5]`
- string: a list of chars
- struct: `X = $employee(john_doe, accounting, 62150)`, `X = new_struct(employee, 3)`
- array: `A = to_array(L)`, `A = {X*X : X in 1..10}`
- map: `Bdays = new_map(), Bdays.put(john,oct3)`
- set: a map without values, `S = new_set([1,2,3]), S.has_key(2)`

Other

- Fairly standard operators (we will introduce them as needed)
- Object-oriented notation
- `,` is conjunction, `;` is disjunction
- `foreach` and `while` loops:

```
foreach(E1 in D1, Cond1, ... , En in Dn, Condn)
    Goal
end

while (Cond)
    Goal
end
```

- assignment:
  - `Var := Exp` change to `Var'=Exp`, change occurences of `Var` to `Var'`
  - `X[I,J] := Exp` destructively updates the component (undone upon backtracking)
- Picat is not a procedural language!

Resources for learning Picat, and constraint programming in Picat:

- The Picat Book "Constraint Solving and Planning with Picat" (free PDF)
- The Picat Guide: "A User's guide to Picat" (in particular, Chapter 12: Constraints)
- There are several tutorials, e.g. this one.
- More resources are available here.
- Lots of Picat programs, as well as other resources, are on Hakan Kjellerstrand's Picat page.

A constraint programming example

For the rest of today, we will practice writing programs in "pure" Picat. We will introduce constraint modelling in Picat next tutorial. But here is one example, the N-queens problem: place N queens on an NxN chess board so that no two queens attack each other.

In [21]:
```
!picat queens/queens.pi 4
```

```
[2,4,1,3]
```

In [22]:
```
!cat queens/queens.pi
```

```
% adapted from picat-lang.org
import cp.

queens(N, Q) =>
    Q = new_list(N),
    Q :: 1..N,
    all_different(Q),
    all_different([$Q[I] - I : I in 1..N]),
    all_different([$Q[I] + I : I in 1..N]),
    solve([ff], Q).

main =>
    queens(8, Q),
    print(Q).

main([N]) =>
    queens(N.to_int, Q),
    print(Q).
```

Exercises

Exercise: count occurences

Write a program that counts the number of occurences of an integer in a list of integers, e.g.:

```
picat count-occurences.pi [1,2,4,2,3,2] 2
picat count-occurences.pi [1,2,2,1] 3
```

outputs `3` and `0`, respectively.

Exercise: transpose

Write a program that transposes a given matrix (a 2D array), e.g.:

```
picat transpose.pi "{{1,2,3},{4,5,6}}"
```

outputs `{{1,4},{2,5},{3,6}}`. (Note that we need to put the input in quotation marks.) Inside your code define a function `transpose(Matrix) = Transposed_Matrix`.

Exercise: binary trees

Write a function that receives a binary tree encoded using the structure `$node(Value,LeftChild,RightChild)` and outputs the depth of the tree. For example:

```
picat depth.pi "node(42,nil,nil)"
picat depth.pi "node(1,node(2,nil,nil),node(3,nil,nil))"
picat depth.pi
"node(1,node(2,node(3,node(4,nil,nil),node(5,nil,nil)),nil),node(6,
```

should output:

```
0
1
3
```