



PetriNet 2 Core

Anastasia Braginsky

08.07.2004

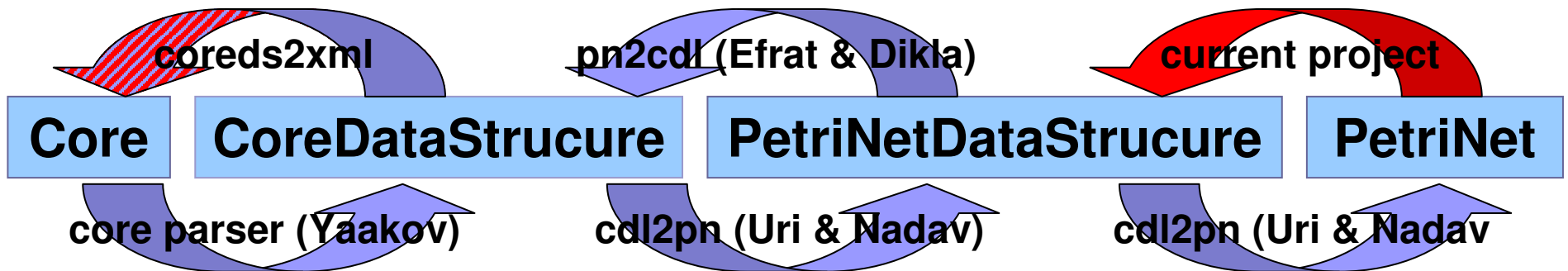


Agenda:

- Abstract
- Syntax of .net file (created by Core2PN)
- The way the parser works
- Assumptions
- An example analysis
- What need to be done

Abstract:

- The project definition to create translation from Core language to the PetriNet and back.
- What we have now:



Syntax of .net file:

```
|NET  
|CDL Module 'SYSTEM'  
|SPECIFICATION ST_Specification
```

```
|PLACES  
|x=F 100  
;  
|x=T 101  
;  
;  
;
```

```
|TRANSITIONS  
|m.0 10000  
;  
|m.1 10001  
;  
;  
;
```

```
|ARCS  
|100 --> 10000 20000  
|1  
;  
;  
|10000 --> 101 20001  
|1  
;  
;  
|101 --> 10001 20002  
|1  
;  
;  
|10001 --> 100 20003  
|1  
;  
;  
;  
;
```

```
|MARKING  
|101  
|1  
;  
;  
;
```

```
|NET_END  
EDITOR_INFOS  
PAGES  
;  
PLACE  
;  
;  
TRANSITION  
;  
ARC  
;  
;  
END_EDITOR
```



The way the parser works :

- Usage
- Used data structures
- Header
- Places and transitions part
- Arcs part
- Marking part



Usage:

- `pn2cdl filename.net`
- If program is used in other way, the program will be terminated with respective printout.



Used data structures:

- Three dictionary data structures are used in the parser:
 - The main one that will hold the pn data structure, and will be passed to the pn2cdl. This is the PetriNet data structure.
 - Places dictionary will hold places which were found in places part.
 - Transitions dictionary will hold transitions which were found in transitions part.



Used data structures (cont.):

- Places and Transitions dictionaries are only for inner use will be destroyed after using.
- We could not use the main DS in this purpose since in the PNDS the keys to the elements are their names, but we want to find them by the identical number.



Header:

- Header is used for finding PetriNet class name. Name is derived from previous CDL Module Name, if it is mentioned, else it is derived from PetriNet file name.
- The additional info comes from transition and place names. So each place/transition name is translated as it is.



Place and transition part:

- Place and transition parts are used to detect their names, and to insert places/transitions to the places/transitions dictionary. Element's identical number is used as insertion key.
- Also places and transitions are inserted as MPplaces and MPtransitions to the main dictionary (petri net data structure).



Arcs part:

- Using origin and destination identical number we would find origin place/transition and destination place/transition, using places and transitions dictionaries.
- If origin or destination is not found connector is not created (optionally the program could be terminated). Else MPconnector is created and inserted to the main data structure.



Marking part:

- Marking part: Each place is found, using id. number. Number of tokens, which follows the id. number is added to MPplace as initial tokens.



Assumptions:

- CDL Module name will be only 64 characters long. If previous CDL Module name or PetriNet file name will be longer then this, it will be cut after 64 characters.
- Connector name will be concatenation of origin and destination name
- Information and error printouts will be printed also to the stdout (screen) and to the log file. Log file name will be <PNfileName>_parser.log. This file will be created if the usage is right and if pn file exists, if not the error will be printed out only to stdout.



An example analysis:

- **cdl flip-flop program (the source):**

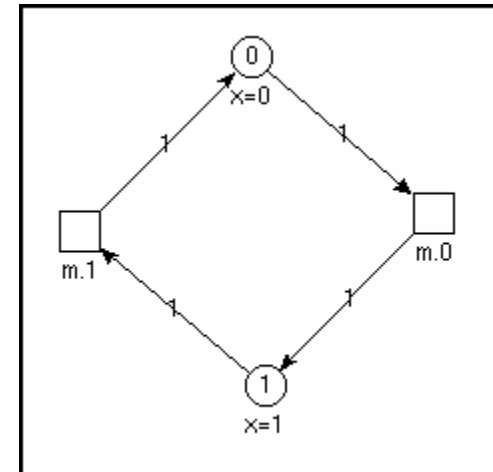
```
HOLD_PREVIOUS
MODULE SYSTEM ()
{
  VAR
  x: boolean INITVAL true;
  TRANS m:
  enable: true;
  assign: x' := !x;
}
```

An example analysis (cont.):

- pn program after translation:

```
|NET
|CDL Module 'SYSTEM'
|SPECIFICATION ST_Specification
|PLACES
|x=F 100
;
|x=T 101
;
;
|TRANSITIONS
|m.0 10000
;
|m.1 10001
;
;
|ARCS
|100 --> 10000 20000
|1
;
;
|10000 --> 101 20001
|1
;
;
|101 --> 10001 20002
|1
;
;
|10001 --> 100 20003
|1
;
;
;
```

```
|MARKING
|101
|1
;
;
|NET_END
EDITOR_INFOS
PAGES
;
PLACE
;
TRANSITION
;
ARC
;
END_EDITOR
```





An example analysis (cont.):

■ Parser run:

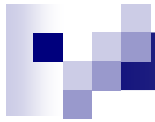
- MPclass name will be “SYSTEM”.
- Two MPplaces with the names: “x=F” and “x=T” will be created with zero initial tokens.
- Two transitions with the names “m.0” and “m.1” will be created.
- Four connectors with the names “x=Fm.0”, “m.0x=T”, “x=Tm.1” and “m.1x=F” and respective origins and destinations will be created.
- Initial token will be added to the place x=T.



An example analysis (cont.):

- cdl program (after two translations):

```
HOLD_PREVIOUS
MODULE SYSTEM ()
{
  VAR
    x: boolean INITVAL true;
  TRANS m:
    enable:  $x \vee !x$  ;
    assign:  $x' := (\text{true} \vee !x) \wedge (\text{false} \vee !x)$ ;
}
```



What need to be done:

- To complete testing on .net files
- To test the program on “real” pn files (on Linux)