

AM07: FDL and Petri Nets

Adam Megacz
megacz@cs.berkeley.edu

October 22, 2006

1 Petri Nets to FDL

This transformation is only valid for *1-safe* petri nets. We define the representation function $\lceil \cdot \rceil$ which maps elements of a petri net to FDL constructs.

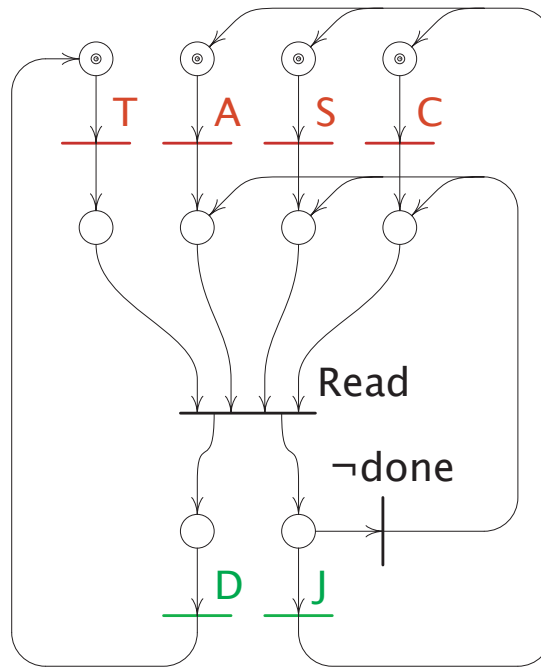
Each *place* p in a petri net is represented by an FDL *channel* $\lceil p \rceil$. Each *transition* T having inbound arcs from places $p_1 \dots p_n$ and outgoing arcs to places representing channels $q_1 \dots q_m$ becomes the FDL process shown below:

```
while true
  |  $\lceil p_1 \rceil, \lceil p_2 \rceil, \dots, \lceil p_n \rceil \rightarrow \lceil q_1 \rceil! \parallel \lceil q_2 \rceil! \parallel \dots \parallel \lceil q_m \rceil!$ 
```

The entire petri net is then represented by the parallel composition of the processes representing its transitions.

1.1 Example

An example is shown below. Note that this example is a *rote* translation from petri nets to FDL. It is not likely to be very *readable*, though it should be *correct*. A hand-written FDL program would be far more legible.



The Memory Read SHIP Petri Net

```
memRead =
|| while true | ?T -> !ReadT
|| while true | ?A -> !ReadA
|| while true | ?S -> !ReadS
|| while true | ?C -> !ReadC
|| while true | ?readT ?readA ?readS ?readC -> !Done || !J || !NotDone
|| while true | ?NotDone -> !ReadA || !ReadS || !ReadC
|| while true | ?J -> !A || !S || !C
|| while true | ?D -> !T
```

2 FDL to Petri Nets

The core FDL grammar is given by:

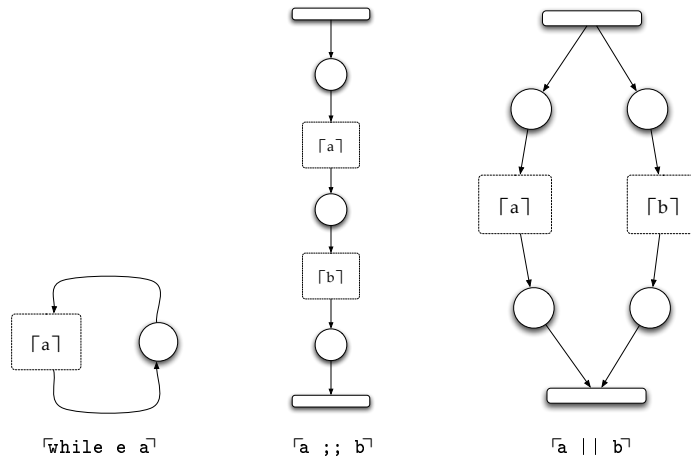
```

STMT ::=  STMT ;; STMT
        |  STMT || STMT
        |  EXPR !-> CHAN
        |  (CHAN+ -> STMT)+
        |  while EXPR STMT
    
```

Channels are statically identified; the `CHAN` production is simply a set of identifiers. Since Petri Nets cannot model computation, the `EXPR` production is irrelevant and the `while` loop is assumed to decide nondeterministically whether or not to continue.

Each occurrence `S` of `STMT` will be represented by a (non-closed) petri net fragment $\lceil S \rceil$. Each fragment will have a designated *start transition* and *end transition*.

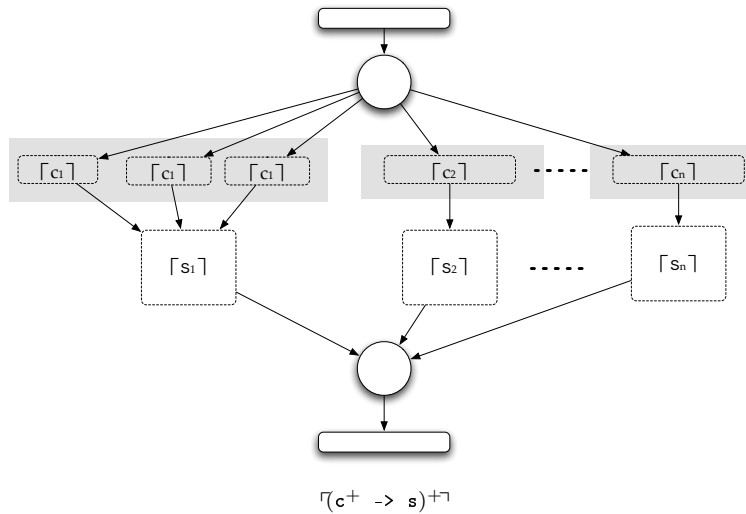
The representation fragments for looping, sequencing, and parallel composition are straightforward:



2.1 Channels

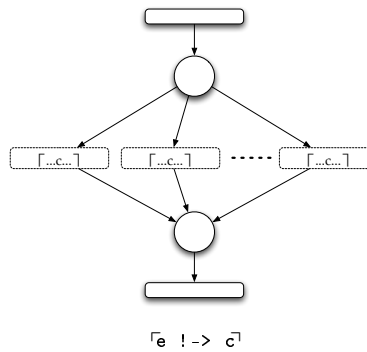
Sending and receiving along channels is a bit more difficult.

For each guard in each receive statement, create a *guard transition set*. In the following diagram, each gray box represents a single guard transition set.



Each guard transition set contains one transition for each possible combination of send statements which could have satisfied the guard. For example, if a guard is triggered by channels a , b , and c , and the program has N_a , N_b , and N_c statements which send to these channels respectively, the guard transition set will contain $N_a \times N_b \times N_c$ transitions in total. Each transition will have an incoming arc from the initial place of each of the SEND statements which it represents.

The translation of a send statement is shown below:



2.2 Example

Below is a hand-written version of the memory write ship. Notice how it is vastly more legible than the mechanically-translated version.

```
memWrite =  
  || while true  
  | ?Token ?Addr ?Stride ?Count -> !DoRead  
  | ?Token ?Repeat               -> !DoRead  
  || while true  
  | ?DoRead ?Done -> if Done then !Repeat
```

The mechanical translation of this FDL program into a petri net appears below:

