Daemon Game (Turner 1993) Problem Description

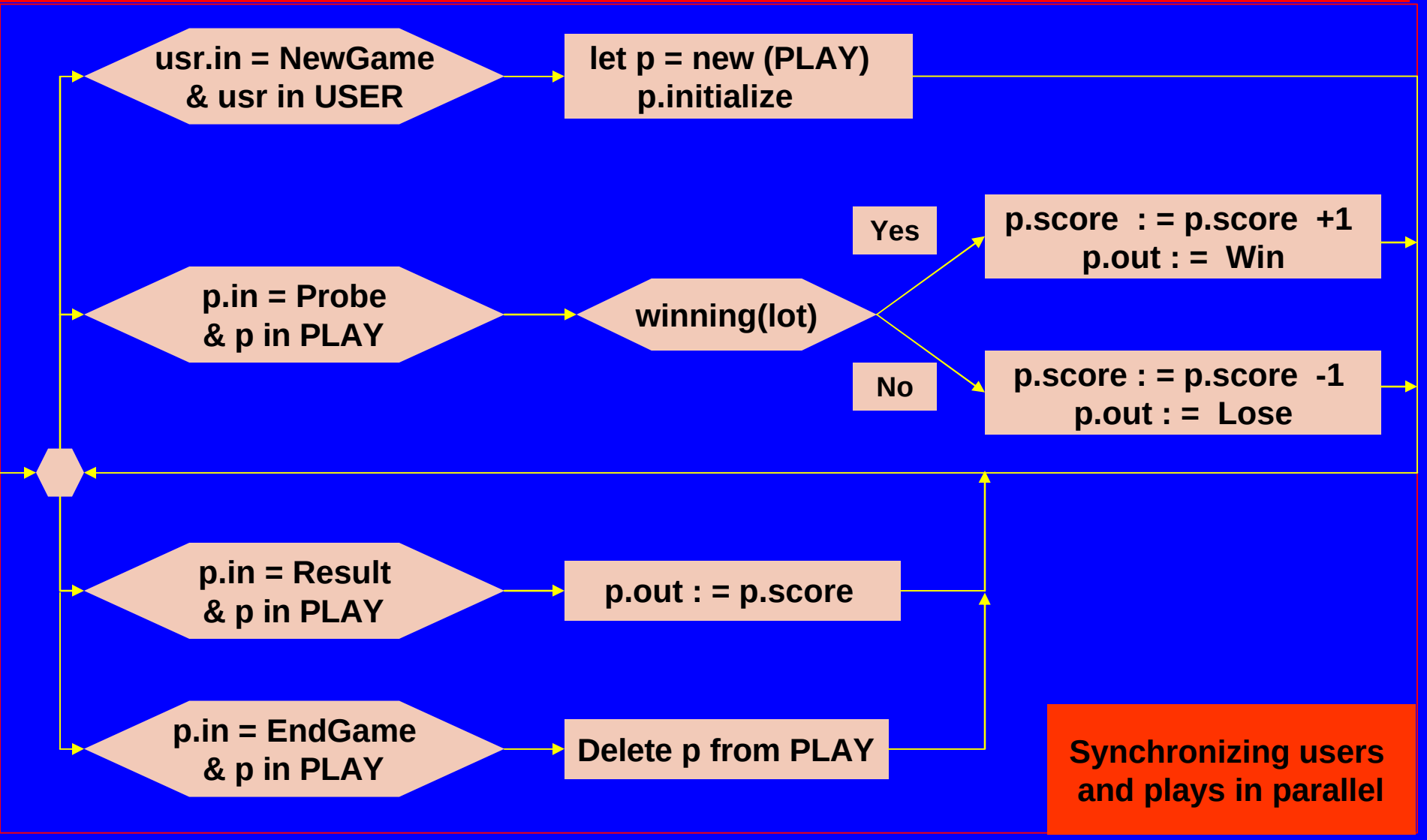
- Design a system for the following multi-player game:
  - A Daemon generates Bump signals at random. Players in system env have to guess whether the number of generated Bump signals is odd or even, by sending a Probe signal. The system replies by sending the signal Win resp. Lose if the number of the generated Bump signals is odd resp. even.
  - The system keeps track of the score of each player. The score is initially 0. It is increased (decreased) by 1 for each (un) successful guess. A player can ask for the current value of the score by the signal Result , which is answered by the system with the signal Score.
  - Before a player can start playing , the player must log in via signal NewGame. A player logs out via signal EndGame. The system allocates a player a unique ID on logging in, and de-allocates it on logging out. The system cannot tell whether different IDs are being used by the same player.

Multi-agent ASM for Daemon Game

with global dynamic external lot issuing daemon

user & play rules

forall usr in USER , forall p in PLAY



Daemon Game : Further Explanations

- p.initialize = p.score:=0
- Is the (lot issuing) Daemon
  - 'integral part of the description' or 'artefact of the informal explanation'?
  - bump count global ('since the system started') or per game?

Both questions are about the nature of a lot (global or local):

  - Monitored fct (s)
    - winning/losing lot, Bump counter increasing by 1, alternating 0-1-fct
  - Agent with rule forall p in PLAY p.bump := 1 – p.bump
  - Derived fct p.bump = bump–$p.bump_{init}$, with global bump, adding to p.initialize the update $p.bump_{init}$:= bump

- Identification of players and games: 'Presumably some identifiers are needed, but how should they be allocated and what should they distinguish?' 'The players should be an anonymous part of the env of the system.'
  - new(PLAY) adds fresh elements p to PLAY, identified as instance parameters
  - possible to link user and play adding p.user:= sender to p.initialize

Daemon Game : Further Explanations (Cont'd)

- Interruption of Probe or Result: 'Should it be allowable for another signal to be processed by the system bw Probe and Win/Lose, or bw Result and Score?' 'Probe and Result should be followed by their respective responses before any other signal is processed.' Guaranteed by atomicity of ASM rule exec
- Robustness Conditions:
  - 'What should happen if a player who is already logged in tries to issue NewGame again?' 'NewGame should be allowed to happen in a current game, but should be ignored.'
  - 'What should happen if a player issues any signal other than NewGame before logging into a game?' 'The intention was to allow Probe, Result, or EndGame when a game is not current, but to ignore these signals.'

  Guaranteed by no ASM rule being applicable (equiv to SKIP)
- Excluding behaviour: 'What should happen if the player issues Win, Lose, or Score signals?' 'The intention was to disallow such behaviour:it simply must not happen, as opposed to happening but be ignored.'

  Excluded by sign cond x.in = NewGame, Probe, Result, EndGame

Daemon Game : References

- K. J. Turner (Ed.): Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL.
  - John Wiley, New York, 1993.
- For a Petri net solution see section 11 (in particular Fig. 7, pg.314) of
  - J. Billington: Protocol Specification Using P-Graphs, a Technique Based on Coloured Petri Nets.
    - In: W. Reisig, G. Rozenberg (Eds.): Lectures on Petri Nets II: Applications. Springer LNCS 1492 (1998) 293-330.
- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook
  - See Chapter 2.2