A Formal Framework for the Verification, Validation, and Simulation of Distributed Systems through Petri Nets and TLA+

Universidad Zaragoza
1474

José Ángel Bañares

Instituto Universitario de Investigación de Ingeniería de Aragón (I3A),
Universidad de Zaragoza,
Spain
banares@unizar.es

Universidad Zaragoza

GECON

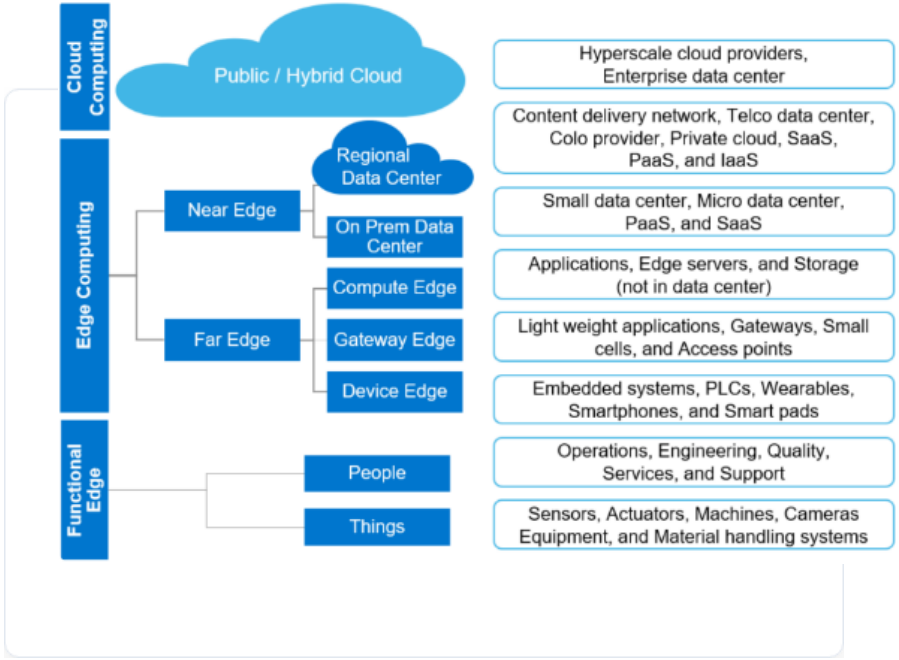
Context and Motivation

Cloud-Edge Infrastructures: Highly concurrent, dynamic, and asynchronous with elastic resource management.

Concurrency Risks: Deadlocks and race conditions severely affect reliability and operating cost.

Insufficient Traditional Testing: Conventional simulation does not explore all explicit execution paths.

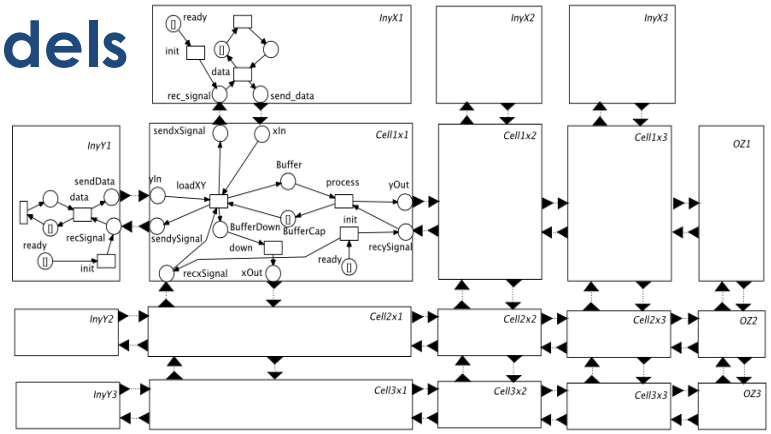
Formalism Gap: A common disjunction between abstract high-level specifications and actual executable code.



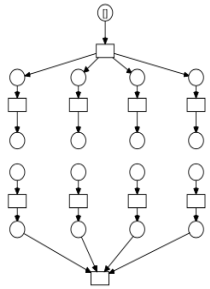
Universidad Zaragoza
1542

Bringing the gap

Models

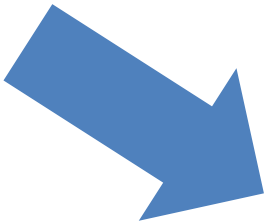


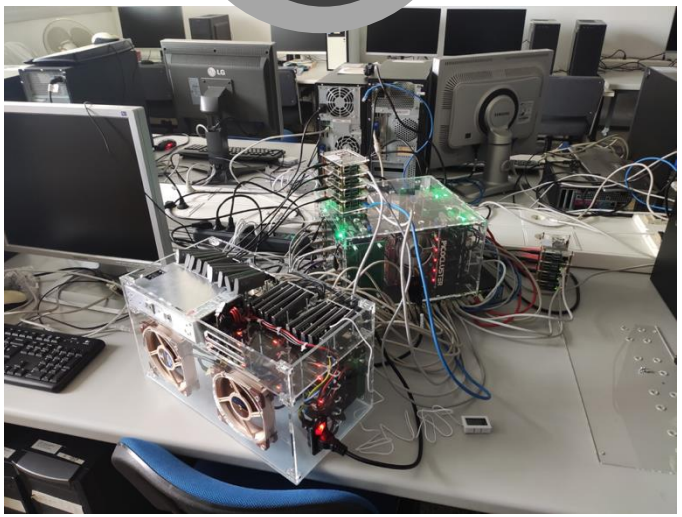
Wafefront model for centralized simulations



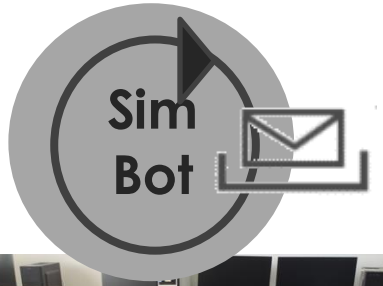
PN model for distributed simulation

Models for simulation experimentation.



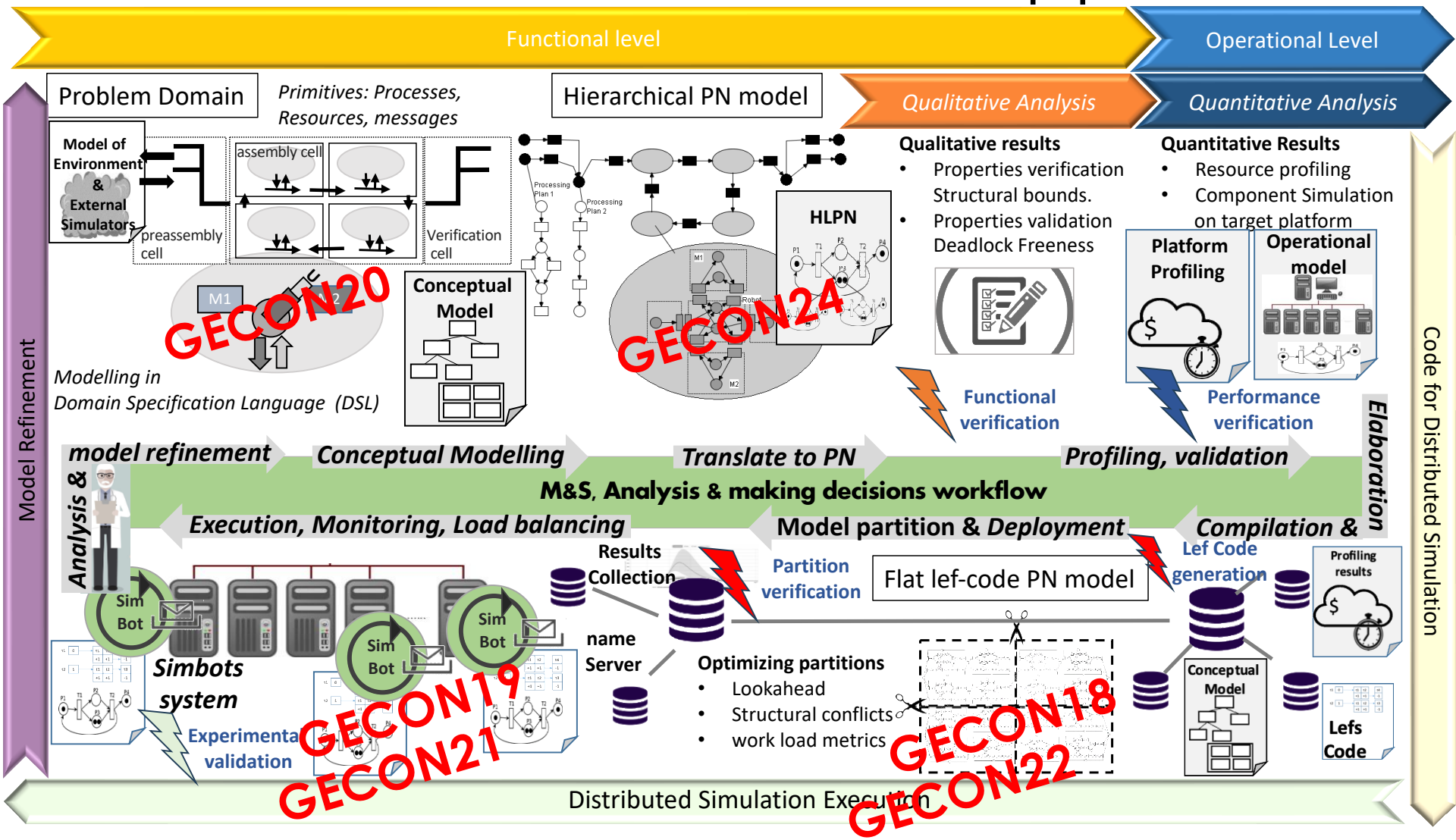


Execution Infrastructure



Simbot

Our MDE PN based approach



- 2025 **Large scale system design aided by modelling and DES simulation: A Petri net approach.** Softw. Pract. Exp. 55(2): 243-271 (2025) **https://doi.org/10.1002/spe.3374**

PN versus TLA+

Petri Nets(PN)

Strengths: Graphical and intuitive model of concurrency, explicit causal structure and flow of resources through token game dynamics.

- Structural and achitectural aspects are naturally captured by PN.

Limitations: Less expressive for verifying complex temporal logical invariants at the level of the global system.

TLA+ (Temporal Logic)

Strengths: Rigorous formal state-oriented logic to verify safety and liveness properties.

- Behavioral correctness and temporal properties.

Limitations: Lacks a native graphical representation of workflow or structured resource topology.

PN versus TLA+ (model analysis)

Petri Nets(PN)

Analysis:

Structural analysis:

- **Incidence matrix C & State equation**

$$M = M_0 + C \cdot \sigma$$

- **Structural** Invariants via **Algebraic Operations** (P and T invariants)
- **Structural Verification** using **Integer Linear Programming**. (siphons, traps, boundness, and LEF)

Qualitative analysis:

- Reachability graph of TPN.

TLA+ (Temporal Logic)

Analysis:

Model checking

- **Safety invariant Checking**
- **Liveness Property Verification**

Proof Verification

- **Deductive Theorem Proving**: Mechanically checks mathematical proofs for systems with infinite state spaces where state-based model checking is computationally intractable.

Simulation: Executable specifications

PN versus TLA+ & model complexity ⚖️

Petri Nets(PN)

Model complexity:

- Colored Petri nets
- Hierarchy
- Nets-within-nets

</> TLA+ (Temporal Logic)

Model Complexity:

- Modular logic separation
- Formal refinement
- PlusCal: pseudocode & structures

PN versus TLA+

Petri Nets(PN)

Model complexity:

- Colored Petri nets
- Hierarchy
- Nets-within-nets

TLA+ (Temporal Logic)

Model Complexity:

- Modular logic separation
- Formal refinement
- PlusCal: pseudocode & structures

The fragmentation trap: Extending PN (high-level, timed, continuous) creates syntactic islands. Every new aspect requires a new language. Tools become incompatible

Formalisms are easy to invent. However, practical methods must have a precise language and robust tools. … **Simple mathematics** (Set theory, logic) **is the ultimate specification language** because it is universal, precise, and timeless. *Leslie Lamport*

Modular Framework Architecture

PetriNet.tla

Central mathematical library. It defines the semantics of bags, input/output relationships and enabling conditions without an explicit concept of time.

PNModelChecker.tla

Orchestrator for thorough verification. Instantiate specific models and execute non-deterministic trigger rules with the TLC model checker.

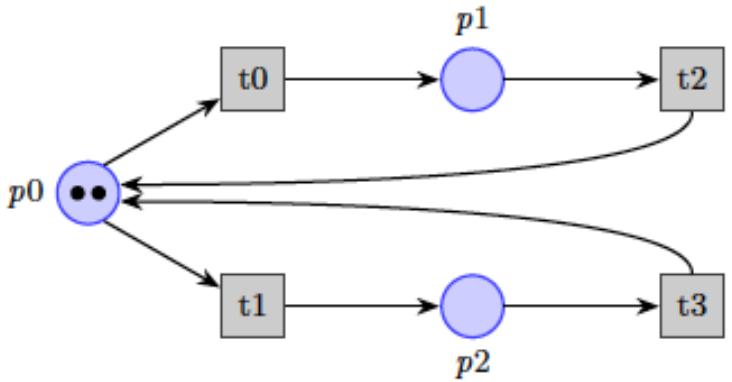
TPNSemantics.tla

Semantic extension with explicit time constraints (global 'now’ clock, 'EnabledAt' transition delays, and event-driven discrete advance).

Specifications

```
---------------------------- MODULE PetriNet ----------------------------
\*       Role: Refinement Mapping and Compliance.
EXTENDS Naturals, Reals, Sequences, FiniteSets, Bags

CONSTANTS
    Places,
    Transitions,
    InputArcs,
    OutputArcs,
    InitialMarking, \*Initial marking
    FiringSemantic,  \* To complete in the model
    TransitionDelays \* Firing delays (for TPN semantics)

\*****************************************************************************
\* Invariants (predicates over the constants)
\*****************************************************************************

ModelInvariant ==
    /\ Places \in SUBSET STRING
    /\ Transitions \in SUBSET STRING
    /\ Places \intersect Transitions = {}
    /\ InputArcs \subseteq Transitions \X Places \X  Nat   \* Input and Output arcs
    /\ OutputArcs \subseteq Transitions \X Places \X Nat   \* are triples (t, place, weight)
    /\ \A arc \in InputArcs  : arc[3] \in Nat \ {0}
    /\ \A arc \in OutputArcs : arc[3] \in Nat \ {0}
    /\ InitialMarking \in [Places -> Nat]   \* Function from places to natural numbers
     /\ FiringSemantic \in {"Interleaving", "SimpleSteps", "MultiSteps", "MaxMultiSteps"}
     /\ TransitionDelays \in [Transitions -> Nat]
    /\ FiringSemantic \in {"Interleaving",
                           "SimpleSteps", \* Simple Step with Simple Use of Place
                           "MultiSteps",
                           "MaxMultiSteps"}
    /\ TransitionDelays \in [Transitions -> Nat] \*  Type checking for delays

\*****************************************************************************
\* Operators:  (all of them operate on the marking as an explicit parameter)
\*****************************************************************************

\*addutuib if tokens per place
SumOfTokens(current_marking) ==
PT!ReduceSet(LAMBDA p, acc: current_marking[p] + acc, Places, 0)

Max(x, y) == IF x > y THEN x ELSE y
```

```
---------------------------- MODULE PetriNetConstants----------------------------
\*****************************************************************************
\* --- Petri Net Constants ---
\*****************************************************************************

Places ==  {"P0", "P1", "P2"}
Transitions == {"T0","T1","T2", "T3"}
InputArcs == {<<"T1", "P0", 1>>, <<"T0", "P0", 1>>,
              <<"T2", "P1", 1>>, <<"T3", "P2", 1>>}
OutputArcs == {<<"T0", "P1", 1>>, <<"T1", "P2", 1>>,
               <<"T2", "P0", 1>>, <<"T3", "P0", 1>>}
InitialMarking == [P0 |-> 2, P1 |-> 0, P2 |-> 0]


FiringSemantic == "Interleaving" \* We can change this to explore different semantics
```



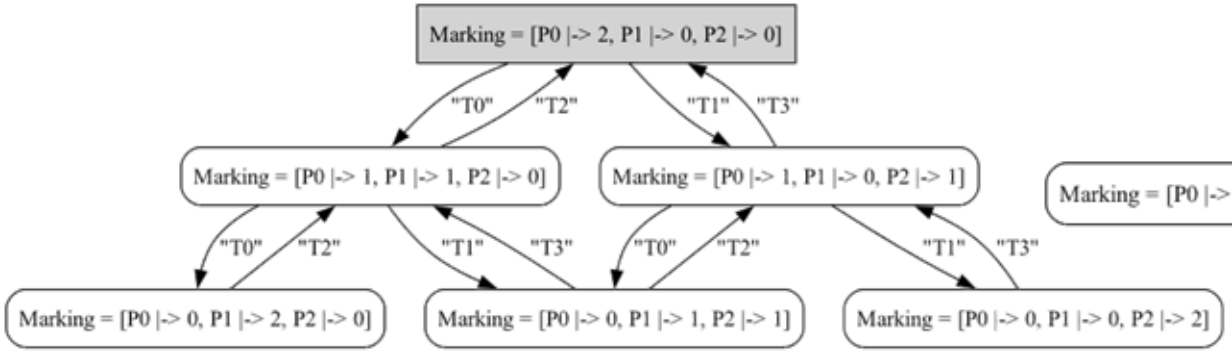
Supported fire semantics

Interleaving: A single transition-enabled fires at each non-deterministic step. Ideal for finding subtle deadlocks in state scans.

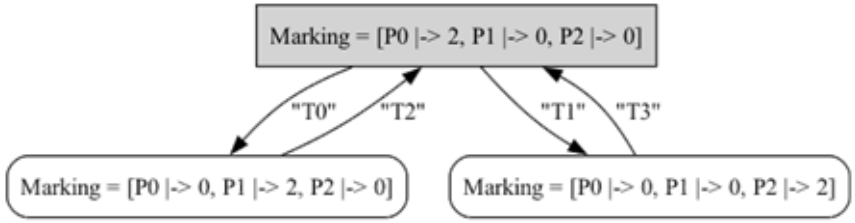
Simple Steps: Sets of independent transitions fire simultaneously without shared resource conflict.

Max MultiSteps: Maximizes the number of transitions executed per step, simulating the maximum possible concurrency in the system.

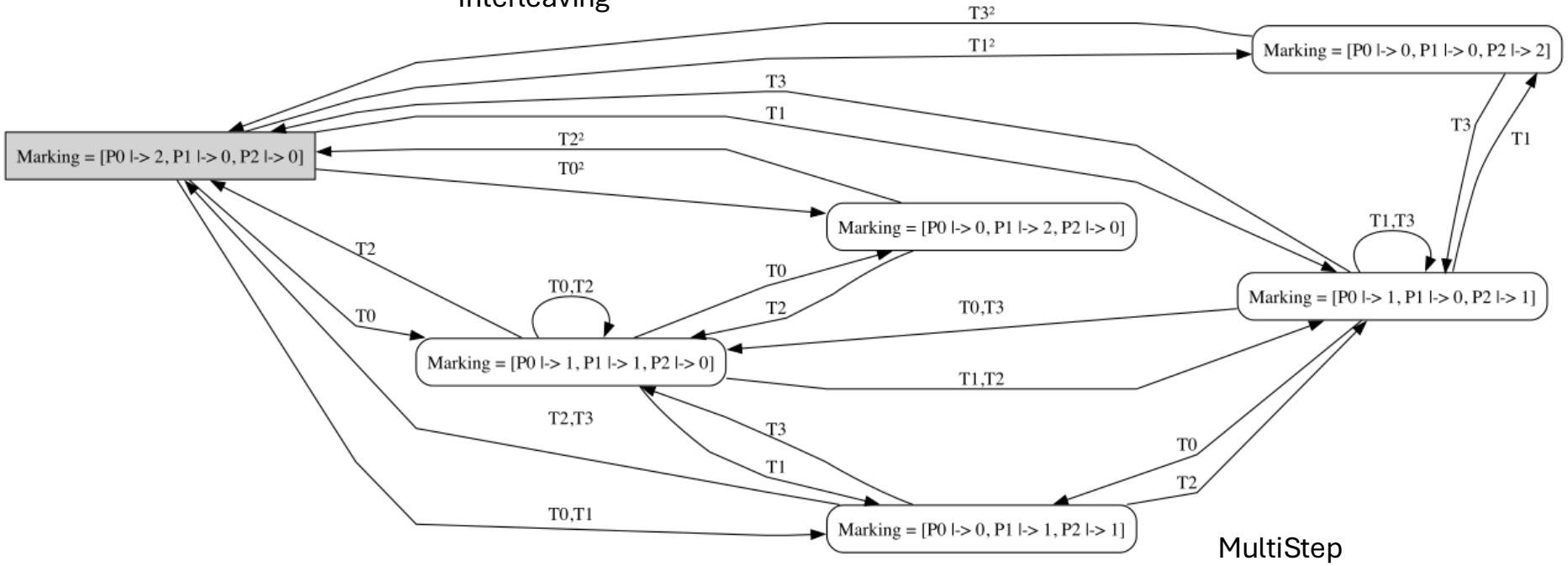
```
InterleavingStep(t) ==
    /\ FiringSemantic = "Interleaving"
    /\ t \in Transitions
    /\ PN_Instance!Enabled(t, Marking)
    /\ Marking' = PN_Instance!FireInterleavingOperator(t, Marking)

MaxMultiStep(t) ==
    /\ FiringSemantic = "MaxMultiSteps"
    /\ t \in Transitions
    /\ PN_Instance!Enabled(t, Marking)
    /\ Marking' = PN_Instance!FireMaxMultiStepOperator(t, Marking)

SimpleSteps(s) ==
    /\ FiringSemantic = "SimpleSteps"
    /\ s \in (SUBSET Transitions \ {{}})
    /\ PN_Instance!EnabledSet(s, Marking)
    /\ Marking' = PN_Instance!FireSimpleStepsOperator(s, Marking)

MultiSteps(mset) ==
    /\ FiringSemantic = "MultiSteps"
    /\ mset \in PN_Instance!Multiset(2)
```



Interleaving



Max.MulltiStep



MultiStep

Timed Petri Nets

Petri net interpretation with time

Transitions are neither instantaneous nor purely logical. Each transition t has a preset delay δ(t).

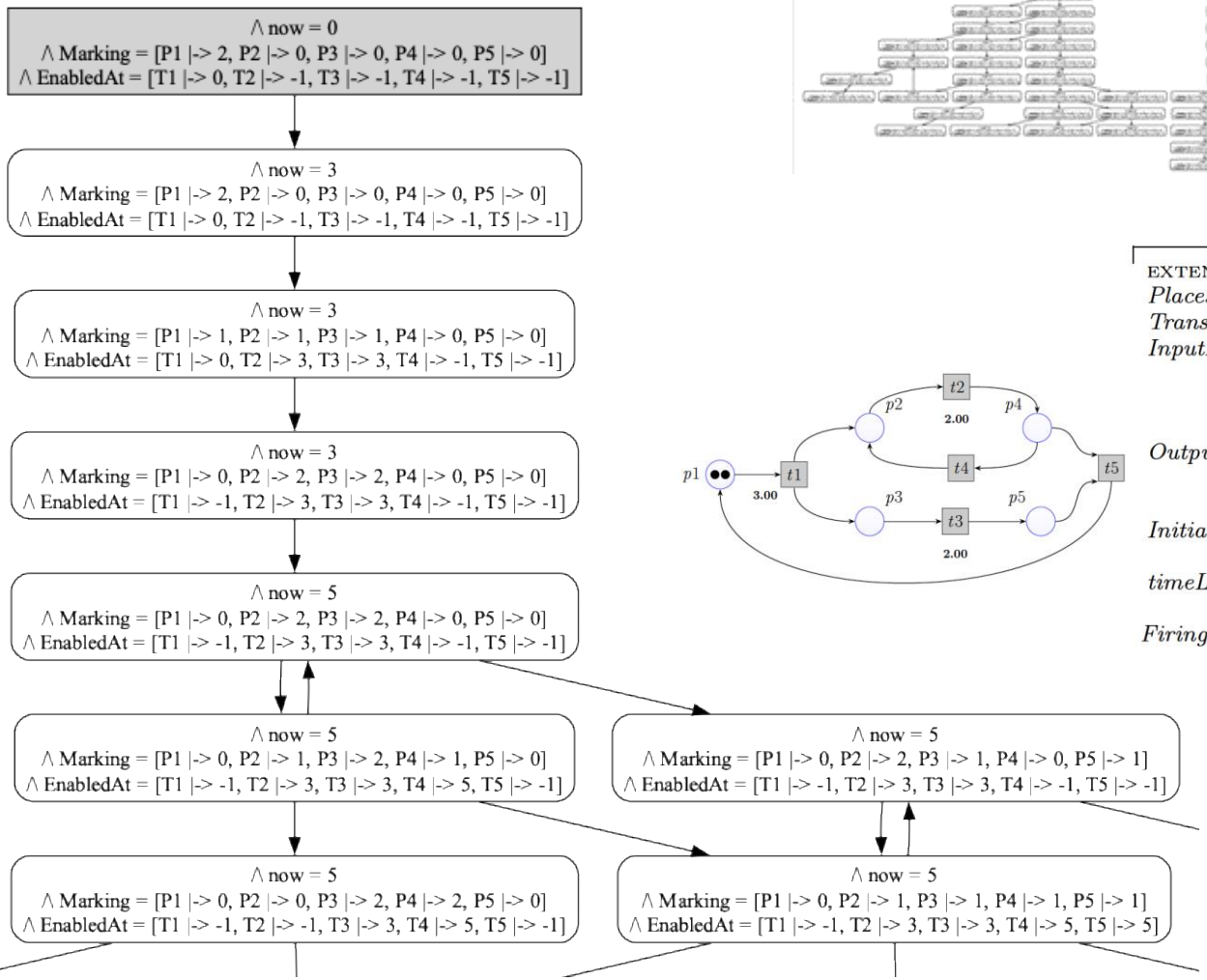
The state of the system is formalized by the triplet S = ⟨M, τ, E⟩ where M is the Marking, τ the Global Clock, and E the Enabling Timer.

```
---------------------- MODULE TPNSemantics ----------------------
EXTENDS Naturals, PetriNet
VARIABLES Marking, now, EnabledAt

\* Versión robusta de NextEventTime
NextEventTime ==
    LET
        \* 1. First, filter the transitions that actually have an active clock.
        Habilitadas == { tr \in Transitions : EnabledAt[tr] /= -1 }

        \* 2. Create the set of firing times (deadlines)
        deadlines == { EnabledAt[tr] + TransitionDelays[tr] : tr \in Habilitadas }
    IN
    \* 3. If nothing is enabled, time can advance freely (+1)
    \* If any are enabled, we pick the minimum (the closest event)
    IF deadlines = {}
    THEN now + 1
    ELSE CHOOSE x \in deadlines : \A y \in deadlines : x <= y

\* Time jump
Tick ==
    /\ now < NextEventTime
    /\ now' = NextEventTime
    /\ UNCHANGED <<Marking, EnabledAt>>

\* Atomic transition firing
InterleavingStep(t) ==
    /\ t \in Transitions
    /\ EnabledAt[t] /= -1
    /\ now = EnabledAt[t] + TransitionDelays[t]
    /\ Marking' = FireInterleavingOperator(t, Marking)
    /\ now' = now
    /\ EnabledAt' = [tr \in Transitions |->
        IF Enabled(tr, Marking')
        THEN IF Enabled(tr, Marking) /\ EnabledAt[tr] /= -1
             THEN EnabledAt[tr]
             ELSE now
        ELSE -1]
=============================================================
```

NextEventTime operator: Determines the exact jump of the clock to the nearest scheduled event.

$$\text{NextEventTime} \triangleq \begin{cases} \min \{ E(t) + \delta(t) : t \in T, E(t) \neq -1 \} \\ \tau + 1 \end{cases}$$

Recheability graph



MODULE *TPN1_Constants*

EXTENDS *Naturals*

Places $\triangleq$ {"P1", "P2", "P3", "P4", "P5"}

Transitions $\triangleq$ {"T1", "T2", "T3", "T4", "T5"}

InputArcs $\triangleq$ {⟨"T1", "P1", 1⟩, ⟨"T3", "P3", 1⟩, ⟨"T5", "P5", 1⟩, ⟨"T5", "P4", 1⟩, ⟨"T4", "P4", 1⟩, ⟨"T2", "P2", 1⟩}

OutputArcs $\triangleq$ {⟨"T1", "P3", 1⟩, ⟨"T1", "P2", 1⟩, ⟨"T3", "P5", 1⟩, ⟨"T4", "P2", 1⟩, ⟨"T2", "P4", 1⟩, ⟨"T5", "P1", 1⟩}

InitialMarking $\triangleq$ $[P1 \mapsto 2,\ P2 \mapsto 0,\ P3 \mapsto 0,\ P4 \mapsto 0,\ P5 \mapsto 0]$

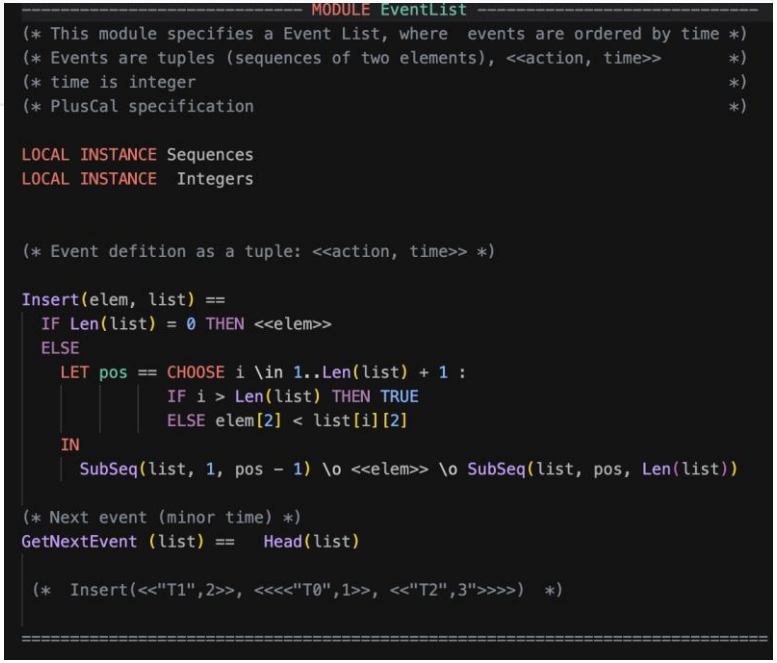
timeLabels $\triangleq$ $[T1 \mapsto 3,\ T2 \mapsto 2,\ T3 \mapsto 2,\ T4 \mapsto 0,\ T5 \mapsto 0]$

FiringSemantic $\triangleq$ "Interleaving"

PlusCal-based Simulation

FUL and LVT algorithms

Use of a Future Update List (FUL) and structures and algorithmic compositions



```
---------------------------- MODULE EventList ----------------------------
(* This module specifies a Event List, where  events are ordered by time *)
(* Events are tuples (sequences of two elements), <<action, time>>       *)
(* time is integer                                                       *)
(* PlusCal specification                                                 *)

LOCAL INSTANCE Sequences
LOCAL INSTANCE  Integers


(* Event defition as a tuple: <<action, time>> *)

Insert(elem, list) ==
  IF Len(list) = 0 THEN <<elem>>
  ELSE
    LET pos == CHOOSE i \in 1..Len(list) + 1 :
                 IF i > Len(list) THEN TRUE
                 ELSE elem[2] < list[i][2]
    IN
      SubSeq(list, 1, pos - 1) \o <<elem>> \o SubSeq(list, pos, Len(list))

(* Next event (minor time) *)
GetNextEvent (list) ==   Head(list)

 (*  Insert(<<"T1",2>>, <<<<"T0",1>>, <<"T2",3">>>>)  *)

=============================================================================
```

Simulation-Verification Duality

It allows transitioning between Model Checking and Random Walks sampling for large-scale systems.

Plus Cal (Centralized) Simulator

- **Bridging Specification to Code:** PlusCal provides an algorithmic representation that translates into TLA+ for model checking.

- **Centralized Event-Driven Simulator:**
  - Uses a Future Update List (FUL) and Local Virtual Time (LVT).
  - **Two-Step Firing:** Separates token consumption (Immediate Update List) from token production upon time completion (Posterior Update List).
  - **Shuffle Operation:** Explores non-deterministic execution orderings for simultaneous events.

```
---- MODULE TPN_CentralizedSimulatorNonDeterminism----
    FUL = << >>,
    t_aux = "";

procedure ScheduleEvent(type, t_id, scheduled_time) begin
    Sched:
        \* Ahora el elemento es << [type, trans], tiempo >>
        FUL := EL!Insert(<< [type |-> type, trans |-> t_id], scheduled_time >>, FUL);
        return;
end procedure;

begin
    InitializeAlg:
        t_set := {tr \in Transitions : PN!Enabled(tr, Marking)};
    InitLoop:
        while t_set /= {} do
            t_aux := CHOOSE x \in t_set : TRUE;
            t_set := t_set \ {t_aux};
            \* El primer evento es IUL: intentar disparar en tiempo LVT
            call ScheduleEvent("IUL", t_aux, LVT);
        end while;

    MainLoop:
        while FUL /= << >> /\ LVT <= MaxTime do
            NextTime:
                LVT := EL!GetNextEvent(FUL)[2];
                (* PRINT 1: Estado al avanzar el reloj *)
                (*print <<"--- TIME:", LVT, "Marking:", Marking, "Events in FUL:", Len(FUL)>>;*)
                print "--- TIME: " \o ToString(LVT);(* \o "| Marking: " \o ToString(Marking) \o
                print " << Al principio del ciclo : " \o " FUL: " \o ToString(FUL) \o " Time: "
                if LVT > MaxTime then goto Finish; end if;


            Collect:
                eventsToProcess := << >>;
            CollectLoop:
                while FUL /= << >> /\ EL!GetNextEvent(FUL)[2] = LVT do
                    eventsToProcess := Append(eventsToProcess, EL!GetNextEvent(FUL));
                    FUL := Tail(FUL);
                end while;
```

```
Shuffle:
   if Len(eventsToProcess) > 1 then
       with p \in { s \in [1..Len(eventsToProcess) -> 1..Len(eventsToProcess)] :
                        \forall i, j \in 1..Len(eventsToProcess) : i /= j => s[i] /= s[j] } do
           eventsToProcess := [ i \in 1..Len(eventsToProcess) |-> eventsToProcess[p[i]] ];
       end with;
    end if;
```

Advance Time and Invariants / **Verification**

Formal Refinement Proof:

- Extraction operator FindTimeInFUL(t) maps the concrete FUL sequence back to theoretical EnabledAt states.
- **Event Awareness Property:** Proves the simulator never misses an enabled transition:

$$\forall t \in \text{Transitions}: (\text{Enabled}(t, M) \Leftrightarrow \text{FindTimeInFUL}(t) \neq -1)$$

LEF Optimization

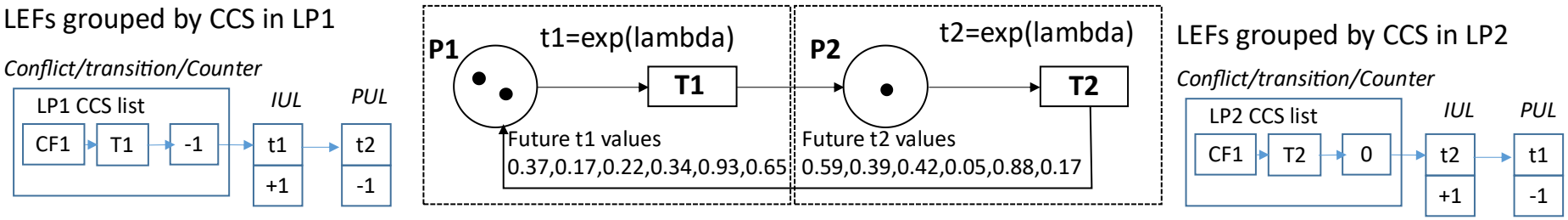
LEF

Linear Enabling Function

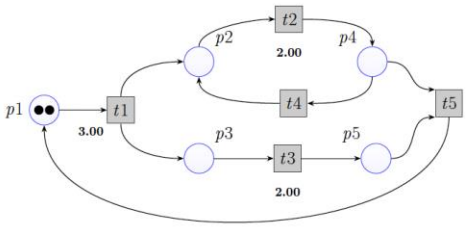
Efficient compute of enabled transitions

Standard implementations evaluate all transitions in each cycle

Through Linear Enabling Function value (LEFs), the computational cost is drastically reduced. Only afected transitions are updated the value of LEF



PlusCal LEF based Simulation



MODULE *TPN1_Constants*

EXTENDS *Naturals*

Places $\triangleq$ {"P1", "P2", "P3", "P4", "P5"}

Transitions $\triangleq$ {"T1", "T2", "T3", "T4", "T5"}

InputArcs $\triangleq$ {⟨"T1", "P1", 1⟩, ⟨"T3", "P3", 1⟩,
⟨"T5", "P5", 1⟩, ⟨"T5", "P4", 1⟩,
⟨"T4", "P4", 1⟩,
⟨"T2", "P2", 1⟩}

OutputArcs $\triangleq$ {⟨"T1", "P3", 1⟩, ⟨"T1", "P2", 1⟩,
⟨"T3", "P5", 1⟩, ⟨"T4", "P2", 1⟩,
⟨"T2", "P4", 1⟩, ⟨"T5", "P1", 1⟩}

InitialMarking $\triangleq$ $[P1 \mapsto 2,\ P2 \mapsto 0,$
$P3 \mapsto 0,\ P4 \mapsto 0,\ P5 \mapsto 0]$

timeLabels $\triangleq$ $[T1 \mapsto 3,\ T2 \mapsto 2,\ T3 \mapsto 2,$
$T4 \mapsto 0,\ T5 \mapsto 0]$

FiringSemantic $\triangleq$ "Interleaving"

MODULE *C1_Constants*

EXTENDS *Integers*, *Sequences*, *TLC*

CONSTANTS *LPs*, *MaxTime*

TransitionNames $\triangleq$ ("C1" :> ⟨"T1", . . . , "T5"⟩)

InitialLef $\triangleq$ ("C1" :> ⟨ − 1, 1, 1, 1, 2⟩)

Delays $\triangleq$ ("C1" :> ⟨3, 2, 2, 0, 0⟩)

IUL_Updates $\triangleq$ ("C1" :> ⟨

$\langle[idx \mapsto 1,\ val \mapsto 1]\rangle,$ $-T1$

$\langle[idx \mapsto 2,\ val \mapsto 1]\rangle,$ $-T2$

$\langle[idx \mapsto 3,\ val \mapsto 1]\rangle,$ $-T3$

$\langle[idx \mapsto 5,\ val \mapsto 1], \ldots\rangle,$ $-T4$

$\langle[idx \mapsto 4,\ val \mapsto 1], \ldots\rangle$ $-T5$

⟩)

PUL_Updates $\triangleq$ ("C1" :> ⟨

$\langle[idx \mapsto 2,\ val \mapsto -1], \ldots\rangle,$ $-T1$

$\langle[idx \mapsto 4,\ val \mapsto -1], \ldots\rangle,$ $-T2$

$\langle[idx \mapsto 5,\ val \mapsto -1]\rangle,$ $-T3$

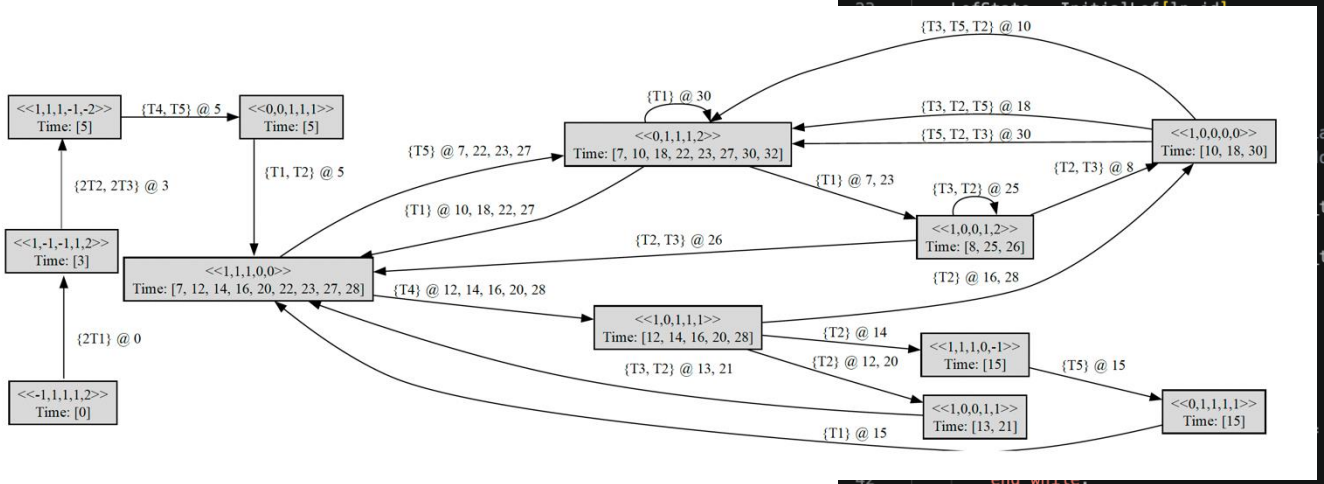
$\langle[idx \mapsto 2,\ val \mapsto -1]\rangle,$ $-T4$

$\langle[idx \mapsto 1,\ val \mapsto -1]\rangle$ $-T5$

⟩)

PlusCal LEF based Simulation

```
TPN_Lef_CentralizedSimulatorNonDeterminism copy.tla > { } TPN_Lef_CentralizedSimulatorNonDeterminism
1   ---- MODULE TPN_Lef_CentralizedSimulatorNonDeterminism ----
2   EXTENDS Integers, Naturals, Sequences, TLC, C1_Constants
3
4   (* Event manager instance *)
5   EL == INSTANCE EventList
6   (***************************************************************************)
7   (* Sematinc operators           (Replace a PetriNe.tla)                    *)
8   (***************************************************************************)
9
10
11  (* LEF threshold-based firing condition *)
12  CanFire(val) == val <= 0
13
14
15  Range(s) == {s[i] : i \in 1..Len(s)}
16
17  (*--algorithm TPN_Lef_CentralizedSimulator
18  variables
19      lp_id \in LPs,
20      LVT = 0,
21      FUL = << >>,
22      pu_list = << >>,
```

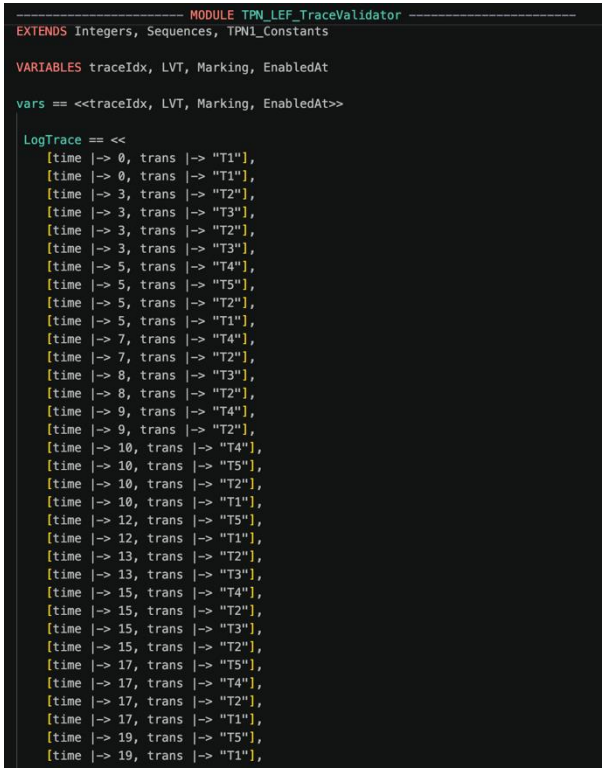



```
43          return;
```

Validation

- To formally validate the LEF implementation, we follow a **trace-driven model checking**.
  - it is validate wheter the specific sequendce of firing events reported in the simlator's execution log constitues a legal trajectory withn the formal semantixcs defined by TPNsemantic.tla

```
--- Step 1: Running TLC Simulator ---
Executing: java -cp /path/tla2tools.jar tlc2.TLC -simulate num=1
          -depth 10000 TPN_Lef_CentralizedSimulatorNonDeterminism.tla
"--- TIME: 0"
" >> LefSate: <<-1, 1, 1, 1, 2>> after collecting events for time 0"
" >> FIRING: ID-394.C1.T1 at time 0"
" >> FIRING: ID-394.C1.T1 at time 0"
"--- TIME: 3"
" >> LefSate: <<1, -1, -1, 1, 2>> after collecting events for time 3"
" >> FIRING: ID-394.C1.T2 at time 3"
" >> FIRING: ID-394.C1.T3 at time 3"
" >> FIRING: ID-394.C1.T2 at time 3"
" >> FIRING: ID-394.C1.T3 at time 3" ...
```



```
---------------------- MODULE TPN_LEF_TraceValidator ----------------------
EXTENDS Integers, Sequences, TPN1_Constants

VARIABLES traceIdx, LVT, Marking, EnabledAt

vars == <<traceIdx, LVT, Marking, EnabledAt>>

 LogTrace == <<
    [time |-> 0, trans |-> "T1"],
    [time |-> 0, trans |-> "T1"],
    [time |-> 3, trans |-> "T2"],
    [time |-> 3, trans |-> "T3"],
    [time |-> 3, trans |-> "T2"],
    [time |-> 3, trans |-> "T3"],
    [time |-> 5, trans |-> "T4"],
    [time |-> 5, trans |-> "T5"],
    [time |-> 5, trans |-> "T2"],
    [time |-> 5, trans |-> "T1"],
    [time |-> 7, trans |-> "T4"],
    [time |-> 7, trans |-> "T2"],
    [time |-> 8, trans |-> "T3"],
    [time |-> 8, trans |-> "T2"],
    [time |-> 9, trans |-> "T4"],
    [time |-> 9, trans |-> "T2"],
    [time |-> 10, trans |-> "T4"],
    [time |-> 10, trans |-> "T5"],
    [time |-> 10, trans |-> "T2"],
    [time |-> 10, trans |-> "T1"],
    [time |-> 12, trans |-> "T5"],
    [time |-> 12, trans |-> "T1"],
    [time |-> 13, trans |-> "T2"],
    [time |-> 13, trans |-> "T3"],
    [time |-> 15, trans |-> "T4"],
    [time |-> 15, trans |-> "T2"],
    [time |-> 15, trans |-> "T3"],
    [time |-> 15, trans |-> "T2"],
    [time |-> 17, trans |-> "T5"],
    [time |-> 17, trans |-> "T4"],
    [time |-> 17, trans |-> "T2"],
    [time |-> 17, trans |-> "T1"],
    [time |-> 19, trans |-> "T5"],
    [time |-> 19, trans |-> "T1"],
```

TPN_LEF_TraceValidator.tla

```
LogTrace == <<[time|->0, trans|->"T1"], [time|-> 0,trans |->"T1"],
[time|->3, trans|->"T2"],  ...>>
TPN == INSTANCE TPNSemantics WITH
now <- LVT, Marking <- Marking, EnabledAt <- EnabledAt ...
TraceIsConsistent == (traceIdx <= Len(LogTrace) =>
TPN!Enabled(LogTrace[traceIdx].trans,Marking)/LogTrace[traceIdx].time>LVT)
```

- **Antecedent (traceIdx <= Len(LogTrace)):** Ensures that trace evaluation remains active as long as there are unprocessed events remaining in the log.
- **Consequent (TPN!Enabled(...) /\ LogTrace[...].time > LVT):** Demands two simultaneous conditions to validate the current logged event:
  - **Structural Enabling:** The transition LogTrace[traceIdx].trans must be genuinely enabled within the Petri Net under the current marking (Marking).
  - **Time Monotonicity:** The timestamp of the recorded event must be strictly greater than (or monotonic with respect to) the accumulated local virtual time (LVT).

```
\* We instantiate the Timed Petri Net semantics to serve as a formal oracle
TPN == INSTANCE TPNSemantics WITH
    now <- LVT,
    Marking <- Marking,
    EnabledAt <- EnabledAt,
    Places          <- Places,
    InputArcs       <- InputArcs,
    OutputArcs      <- OutputArcs,
    TransitionDelays <- timeLabels

Init ==
    /\ traceIdx = 1
    /\ LVT = 0
    /\ Marking = InitialMarking
    /\ EnabledAt = [t \in Transitions |-> IF TPN!Enabled(t, InitialMarking) THEN 0 ELSE -1]

Next ==
    /\ traceIdx <= Len(LogTrace)
    /\ LET currentEvent == LogTrace[traceIdx]
       IN
        IF currentEvent.time > LVT
        THEN
           \* If the log event occurs in the future, advance the simulation time
           /\ LVT' = currentEvent.time
           /\ EnabledAt' = [t \in Transitions |->
                                IF EnabledAt[t] /= -1 THEN EnabledAt[t] ELSE -1]
           /\ UNCHANGED <<traceIdx, Marking>>
        ELSE
           \* If the log event occurs at the current time, validate and fire the transition
           /\ currentEvent.time = LVT
```

Conclusions

- **Unified Formal Framework:** Deep embedding of Petri Net structural models inside TLA+, combining structural intuition with declarative temporal logic verification.
- **Formal Refinement:** Model-Driven Engineering bridge connecting abstract Timed Petri Nets to executable PlusCal simulation algorithms.
- **Dual Verification-Simulation Capability:** Exhaustive model checking for safety/liveness invariants. Random-walk state-space sampling for deep trace exploration. Trace conformance checking against real execution logs.
- **Economic Relevance:** Provides verifiable guarantees for distributed system implementations.

Model Driven Engineering based on PN &TLA+

Model-Driven Engineering

It bridges the gap between abstract specifications and the executable code of complex distributed systems.

AI Development Support

It provides a rigorous "Source of Truth" that guides and validates AI-assisted code.

Qualitative and Quantitative Analysis

It combines strict logical verification with timed simulation-based performance metrics.

Future work

- **Pluscal specification of distributed algorithms**
- **Composition versus refinement:**
  Composition of PN specifications (LEF coded)
- **PN extensions:** High level Petri nets, Continuous PN
- **Conformance of specifications**
- **Deductive Theorem Proving Use**

Universidad Zaragoza

A Formal Framework for the Verification, Validation, and Simulation of Distributed Systems through Petri Nets and TLA+

José Ángel Bañares

Instituto Universitario de Investigación de Ingeniería de Aragón (I3A),
Universidad de Zaragoza,
Spain
banares@unizar.es

Universidad Zaragoza

GECON