Tag-based modelling of the AAA/SynDEx methodology

Work in progress

Dumitru Potop
Yves Sorel

INRIA Rocquencourt
France

Motivation

- Changes (recent and upcoming) in the AAA/SynDEx methodology
- Need for a better formal framework
  - Temporal aspects
    - Need a finer, more expressive, and dedicated formal framework
    - Benveniste’s **tagged systems** look good
  - Components and events
    - Implementation includes structural refinement, distribution, complex synchronization mechanisms, etc.
- Currently: Take the current SynDEx version and model it

Outline

- SynDEx presentation
- Theoretical bases
  - The tagged systems of Benveniste et al.
  - Tagged system transformations
- Modelling of SynDEx
- Conclusion

AAA/SynDEx methodology (today!)

- Dataflow
  - Dataflow specification (synchronous)
  - Implementations of basic blocks/functions
- Architecture
  - Architecture specification
  - HW, drivers and libraries
- Timing information
  - WCETs of basic block implementations

R-T Implementation

- Kahn-like implementation
  - Untimed
  - Distributed
  - I/O deterministic
  - Sequential processes
- WCET of a reaction

Current online version:
Static periodic offline scheduling

SynDEx flow

Phase 1: Real-Time Scheduling

- Dataflow specification
- Architecture spec
- Timing information

⇒

- Scheduled dataflow (timed)
- WCET for one sync reaction

Phase 2: Synchronization Synthesis

- Scheduled dataflow
- Architecture spec

⇒ Implementation model (untimed)

Phase 3: Code generation

- Implementation model
- Architecture (HW and libraries)
- User-supplied (dataflow) routines

⇒ Implementation (untimed)

Dataflow specification

- Graphical/textual synchronous formalism
- Similar to Lustre
  - Hierarchic dataflow. Each operator reads/writes all inputs/outputs at each reaction
  - Differences:
    - No « when », nor « current », but « conditioning »
      - Multiway tests on integers. No bounds check, no « default » branch
    - Only one branch is evaluated in a test/conditioning (primitive hierarchical control flow)
      - Tests create activation conditions (simple sub-clocks)

Example

Architecture specification

- High-level description
  - Operators = Computing units (processors, ASICs, etc.)
  - Communicators (3 types)
  - Arcs between compatible operators and communicators
- Determinism through sequentiality
  - Operators are sequential
  - Operations on communicators are fully ordered using provided deterministic token passing mechanisms
    - Complex when combined with conditional execution
- Untimed

Communicators

- Two main types (quite different)
  - RAM
    - Data passing primitives: Write, Read
  - SAM: memory-less message passing (rendez-vous)
    - Data passing primitives: Send, Receive
    - Two sub-types: Unicast and Broadcast
      - Unicast makes one Send per destination
      - For Broadcast, all operators must read each data
- Communicators are not directed
  - Everybody can send or receive (difficult under conditional exec)
- Deterministic control (token) passing primitives:
  - In SAMs, control goes to the next sender
  - In RAMs, control goes to the operator that reads or writes next

Operators

- Programmable HW units
- Complex structure
  - Sequential, fully programmable functional units
    - 1 computing unit. Executes user-defined (dataflow) operations.
    - 1 communication unit per interfaced communicator. Executes data sending and receiving primitives.
    - All functional units support conditional execution
  - Implicit internal RAM memory
    - Variable-addressed
    - Each variable behaves like a RAM communicator (operations fully ordered using deterministic token passing)
- No time-triggered behavior!

Timing information

- Computations
  - The WCET of each basic dataflow block on each compatible operator.
- Communications
  - The WCET of each communication operation (send/receive, read, write) for each data type supported by each communicator
  - The access to internal operator memory takes 0 time
- Synchronization
  - Token passing delays are abstracted as 0

Other inputs, for actual implementation

- Implementation of basic dataflow functions
  - User-provided
  - Time-independent
- HW, drivers and libraries
  - Implementing the abstract architecture

The implementation

- Must ensure that « send/receive » and « write/read » protocols match
- Solution:
  - Schedule everything globally, then project it on components
  - Global order must be sequential on components
- Untimed:
  - Functionally deterministic, regardless of input arrival dates (delay-insensitive on the interface)

Semantics and correctness

- Graphs (for causality/order), tags for R-T dates, distribution
  - Typical representation used for R-T scheduling
- Graph transformations
  - Per-transformation correctness criteria (no global one)

The future

- Dataflow specification extensions
  - aperiodic events
  - multi-periodic specifications
- Architecture and implementation
  - Better definition of current assumptions
  - Timed architectures and implementations, multi-periods
  - Less constrained implementations (complete ordering of communications not always necessary, synchronization optimization)
  - Less complex operators
  - Pre-computed preemption
- Maybe a better formal model, allowing global correctness reasoning

Tagged systems

- Basic Benveniste et al. theory
- Behavior transformations

Some notations

- N = set of non-negative integers
- $\overline{N}$ = N∪{∞}
- Words(D)=D^*∪D^ω
  - Len(w):Words(D)→ $\overline{N}$
  - Indexing for w□Words(D)
    - w[n]□D for all 0≤n<Length(w)
  - Projection operators on Words(AxB):
    - Π_A: Words(AxB)→Words(A), Π_B:Words(AxB)→Words(B)

Behaviors and events

- Behavior/trace over variables V and data domain D

  $\sigma: V \rightarrow Words(D)$
- Beh(V,D) = set of behaviors over V and D
- System: $\Sigma \subseteq Beh(V,D)$
- $Events(\sigma)=\{(x,n) \mid 0 \leq n \leq Length(\sigma(x))\}$

Tagged behavior

- Tag structure
  - Non-void poset $(T,\leq)$
  - Tag structure morphism $\rho : (T_1,\leq_1) \rightarrow (T_2,\leq_2)$
    - Increasing function: $\tau_1 \leq_1 \tau_2 \Rightarrow \rho(\tau_1) \leq_2 \rho(\tau_2)$
    - Composing morphisms gives a morphism
- Tagged behavior over V and $(T,\leq)$
  - σ□Beh(V,DxT) with σ(x) increasing for all x
  - $Beh^T(V,D)$ = set of all such behaviors
- Tagged system: $\Sigma \subseteq Beh^T(V,D)$

Tagged behavior (2)

- « Pure » tagged behaviors
  - Data values are not meaningful
  - Notations: Beh(V) $Beh^T(V)$
- Notations: Let σ□Beh^T(V,D) and e□Events(σ)
  - Tag(e) = the tag associated with event e
  - Value(e) = the data value associated with e, if any

Sample tag structures

- Product tag structure
  - $(T,\leq_T)x(U,\leq_U)=(TxU,\leq)$, where
    $(t_1,u_1) \leq (t_2,u_2)$ iff $t_1 \leq u_1$ and $t_2 \leq u_2$
- Vector tags T(R), where R is a set of resources (e.g. variables)
  - Particular case of product tag systems
  - $T(R)=\mathbb{N}^R$ with the product order
- Lexicographic product of tags
  - $(T,\leq_T)\square(U,\leq_U)=(TxU,\leq_l)$, where
    $(t_1,u_1) \leq_l (t_2,u_2)$ iff $t_1<u_1$ or ($t_1=u_1$ and $t_2\leq u_2$)

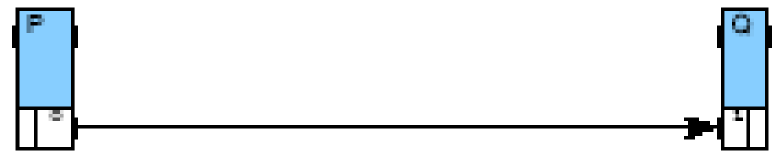
Vector tags

- Allows the representation of any causality pattern between the events of σ□Beh(R,D)
  - Natural choice for causality study: $Beh^{T(R)}(R)$
- Special cases (notations):
  - T({CLK}) = synchronous « clock »
  - T({RT}) = discretized RT clock

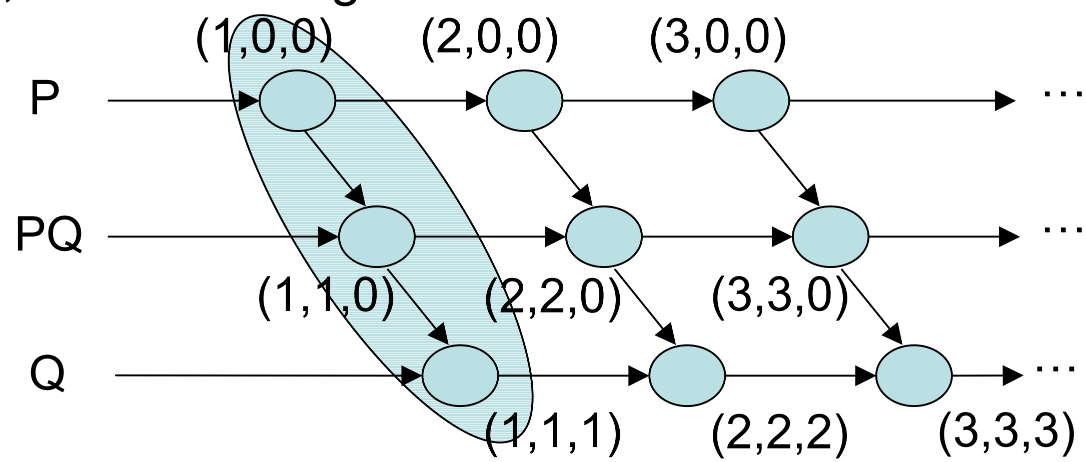
Sample tag structure morphisms

- Projections, in product tag systems
  - Π_T: (TxU, $\leq$)$\longrightarrow$(T,$\leq_T$), Π_U: (TxU, $\leq$)$\longrightarrow$(U,$\leq_U$),
- Projections in vector tags
  - Π_S: T(R)$\longrightarrow$ T(S), for some $S \subseteq R$

A simple SynDEx dataflow spec



- Two operations
- One data comm
- Tag structure: T({P,PQ,Q})
- Tagged behavior (unique) σ_1□$Beh^{T(\{P,PQ,Q\})}(\{P,PQ,Q\})$
  - nodes = events; arcs = order generators



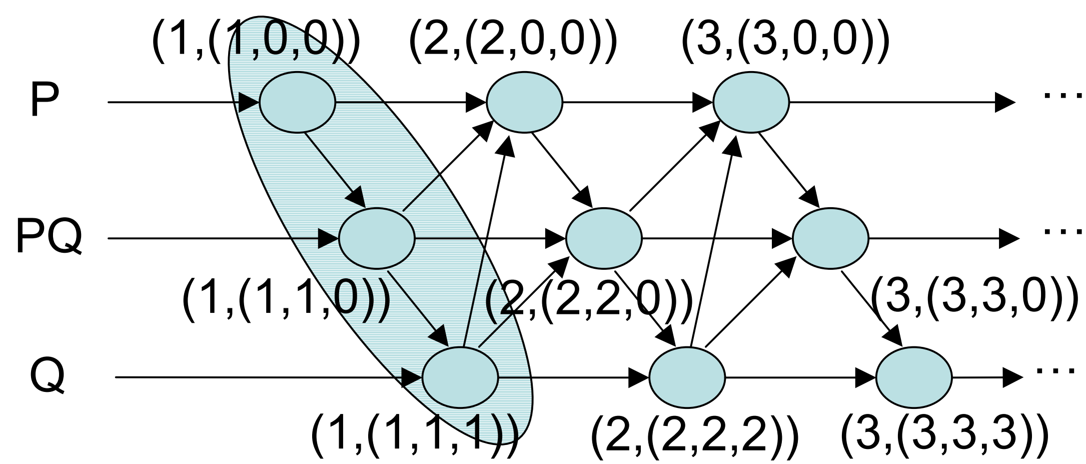
- Same as the Cucu/Sorel model: infinite pattern repetition

Synchronous dataflow semantics

- Tag structure = lexicographic product:

  T_{Sync}=T({CLK})□T({P,PQ,Q})

- σ_2□$Beh^{T_{Sync}}$({P,PQ,Q})



- Same as the Cucu/Sorel model: infinite pattern repetition

Behavior transformations

- Transformation of σ_1 in σ_2
  - Binary relation $\alpha \square$ Events(σ_1)xEvents(σ_2)
  - Notation: $\sigma_1 \square^{\alpha} \sigma_2 \quad e_1 \square^{\alpha} e_2$
  - Transformation composition: $\sigma_1 \square^{\alpha} \sigma_2 \square^{\beta} \sigma_3 \Rightarrow \sigma_1 \square^{\beta \circ \alpha} \sigma_3$
- Correct transformation = order-preserving

$$\left.\begin{array}{l} e_1 \square^{\alpha} e'_1,\ e_2 \square^{\alpha} e'_2 \\ Tag(e_1) \leq_1 Tag(e_2) \end{array}\right\} \Rightarrow \ Tag(e'_1) \leq_2 Tag(e'_2)$$

  - Composition preserves correctness

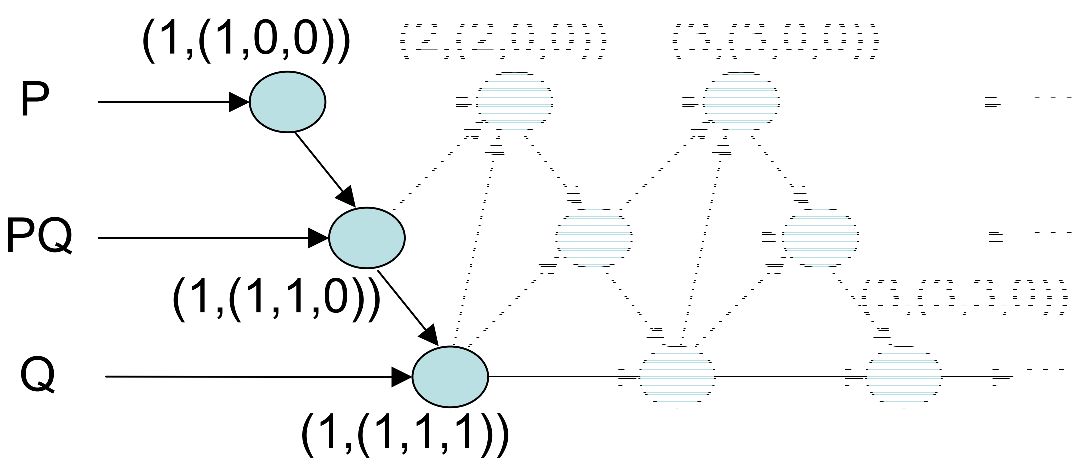
Behavior transformations

- Time transformations
  - Time refinement – reverse tag morphism
  - Time abstraction – direct tag morphism

- The identity relation Events(σ_1)□Events(σ_2) is correct
  - No tag morphism, though ☹

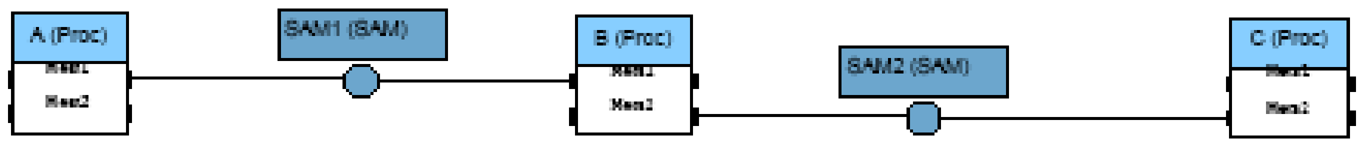
Reduction to a single repetition

- Timing analysis done on one instant only. Then repeat the pattern on the synchronous clock
  - Tag machines could represent this repetition, but they are currently not well-defined



Architecture representation

- Example:



- Tagged system representation:
  - Variable set: {A,B,C,SAM1,SAM2}
  - Causal tagged system
  - $\Sigma_{Archi} \subseteq Beh^{T(\{A,B,C,SAM1,SAM2\})}(\{A,B,C,SAM1,SAM2\})$
    - All behaviors where direct causalities exist only between adjacent architecture elements
    - We assume all these behaviors are implementable in practice
- Find a correct transformation σ_2 into σ □ Σ

Distribution and Scheduling

- Allocation
  - Assign to each operation variable a processor variable
  - Assign to each communication variable a route (sequence of architecture variables)
    - Refine the behavior (the comm events) to take this into account
- RT scheduling
  - Assign dates to all events in the refined behavior
  - The event ordering by RT tags must:
    - Be total among the events assigned to each architecture variable (moreover, it must respect given WCETs)
    - Be stronger than the synchronous causal order
- Change the variable set to the implementation one

Allocation

- P □ A, Q □ C, PQ □ (SAM1,B,SAM2)

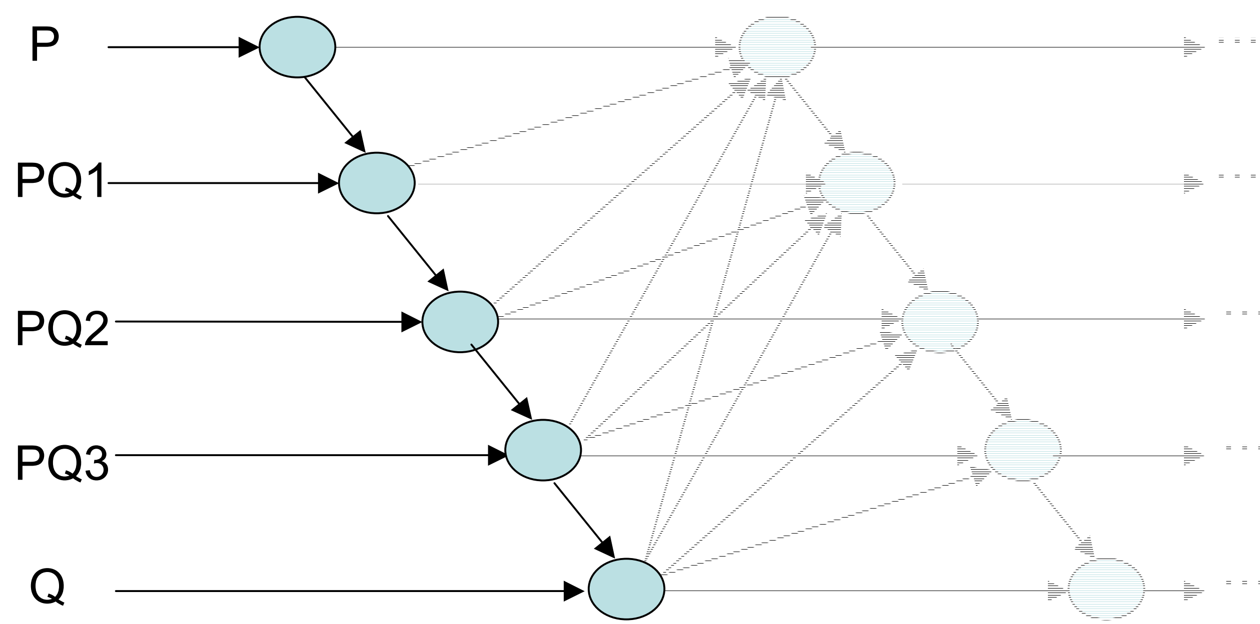
  σ_3 □ $\mathrm{Beh}^{T(\{CLK\}) \square T(\{P,PQ1,PQ2,PQ3,Q\})}(\{P,PQ1,PQ2,PQ3,Q\})$

Refinement of PQ:

PQ1 □SAM1

PQ2 □B

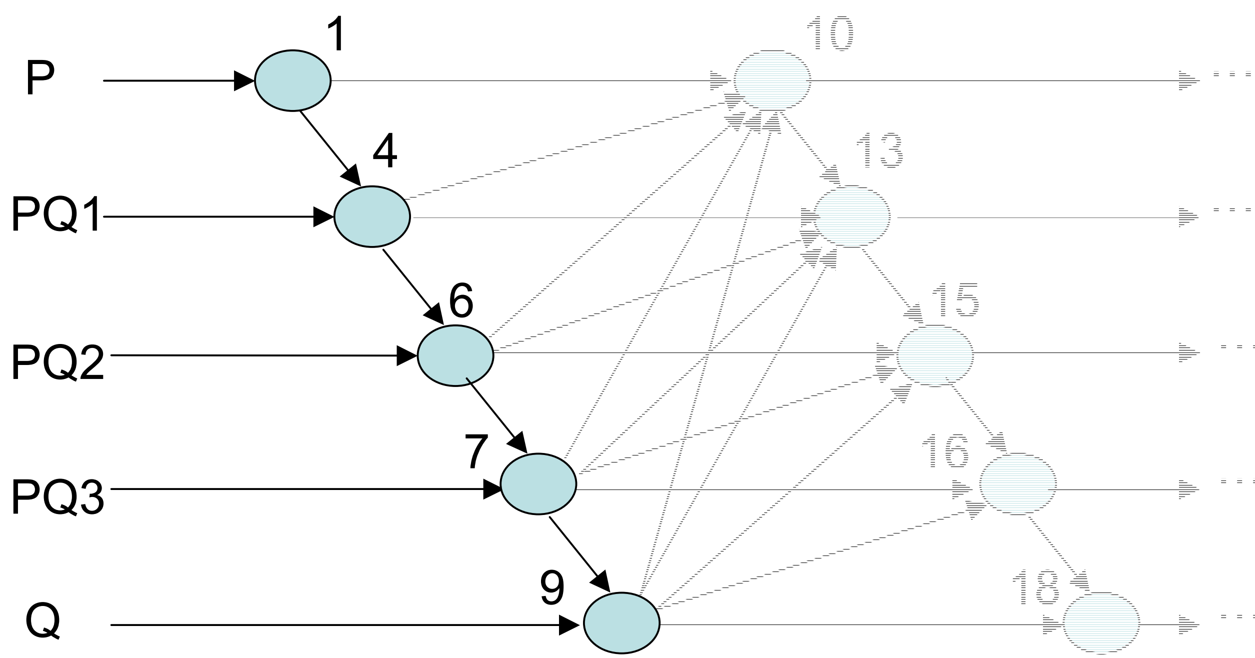
PQ3 □SAM2



RT Scheduling

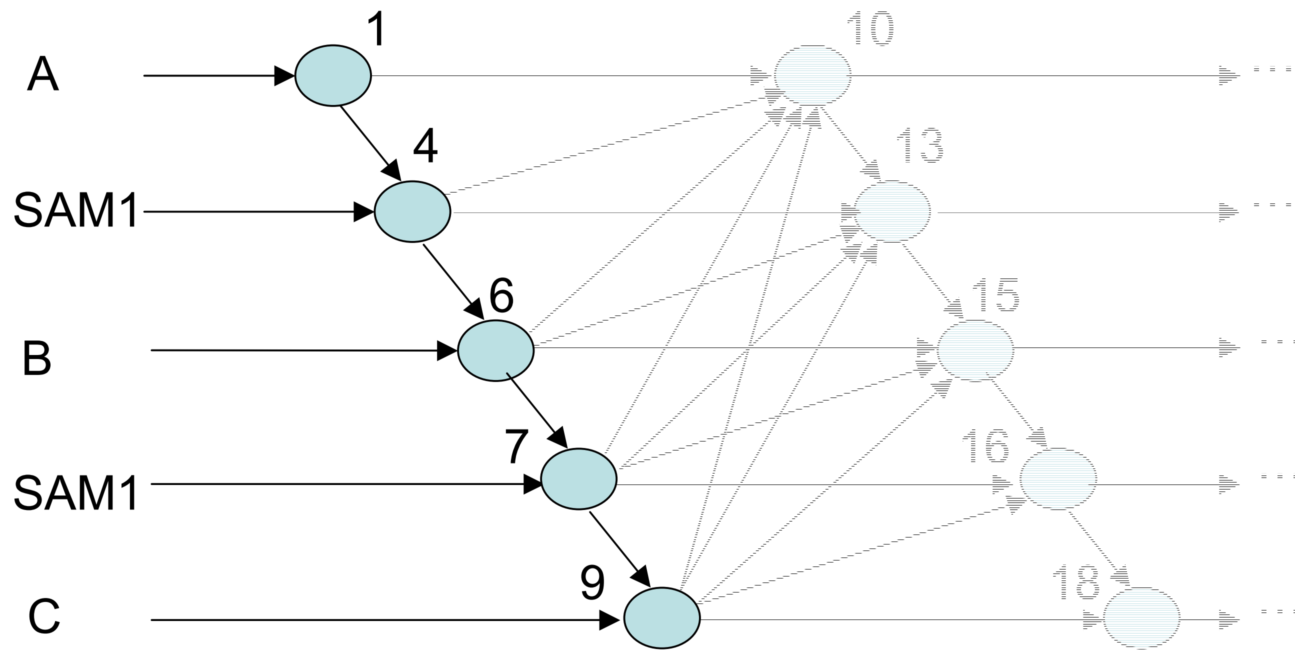
- σ_4 □Beh$^{T(\{RT\})}$({P,PQ1,PQ2,PQ3,Q})
  - RT = real-time clock (integer dates)

Refinement of PQ:
PQ1 □SAM1
PQ2 □B
PQ3 □SAM2



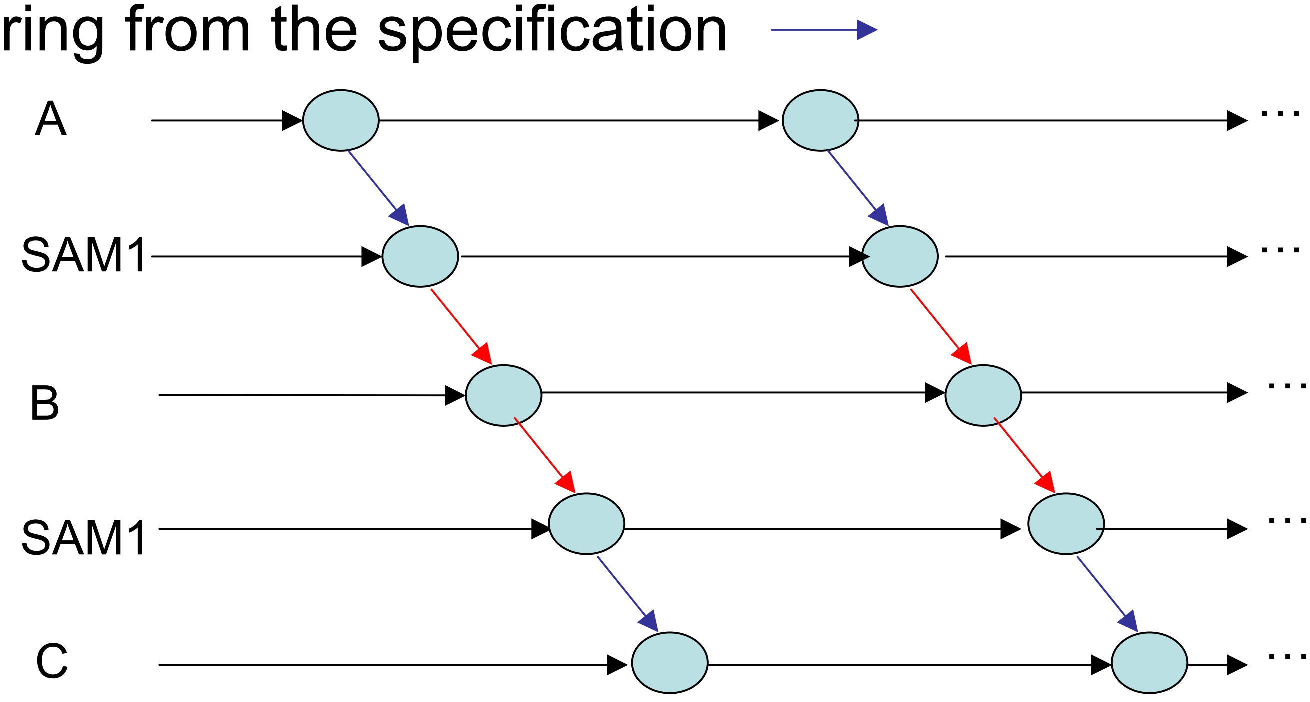
Move to implementation variables

- Place events on implem variables
  $\sigma_3 \in Beh^{RT}$ ({A,B,C,SAM1,SAM2})



Untimed implementation

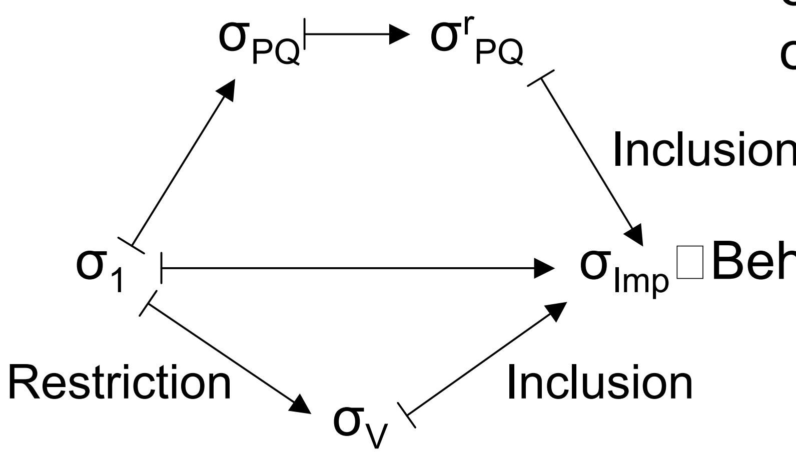
- Preserve only:
  - Order among events of one variable
  - Event ordering from the specification



  - Order between communication events along a path

Untimed implementation

- Formally:



σ_{PQ}□$Beh^{T(\{PQ\})}$ ({PQ})

σ^{r}_{PQ}□$Beh^{T(\{PQ1,PQ2,PQ3\})}$ ({PQ1,PQ2,PQ3})

σ_{Imp}□$Beh^{T(\{A,B,C,SAM1,SAM2\})}$({A,B,C,SAM1,SAM2})

σ_{V}□Beh^{RT} ({V}), V□{A,B,C,SAM1,SAM2}

- The least partial event ordering of $Beh^{T(\{A,B,C,SAM1,SAM2\})}$ ({A,B,C,SAM1,SAM2}) that makes all transformations correct

Conclusion

- Ongoing modelling work
- Some things are missing: tag machines
- Few time transformations
- Future:
  – Use the approach to model architecture model refinement
    - Have the events correspond to actuall calls of implementation functions (send/receive, token passing, dataflow block computations)