Lossy channels in a dataflow model of computation

Pascal Fradet, Alain Girault, Leila Jamshidian, Xavier Nicollin, Arash Shafiei

Spades, Inria Grenoble Rhone-Alpes

November 28, 2017

Outline

Introduction

Context

- We target Internet of Things applications (Iot).
- IoT applications form a dataflow communication between nodes.
- It seems a good idea to use a dataflow Model of Computation (MoC) to program such applications.

Advantages

- Formal reasoning
- Ensuring bounded memory
- Ensuring absence of deadlock
- Computing a static or quasi-static schedule
- Performance evaluation: max buffer sizes, latency, period, ...

Problem

- IoT applications are subject to communication losses that can arise from interferences, low bandwidth, and power shortage
- Many communication protocols to deal with lossy channels are based on retransmission
- Traditional dataflow MoCs cannot cope with lossy channels
- Either they are too static and do not allow the retransmissions to be controlled dynamically
- Or they are too dynamic and do not allow formal reasonning

Example with a lossy channel $\rightsquigarrow$

- Consider the graph $X \rightarrow Y \rightsquigarrow Z \rightarrow T$
- If the data received by Z is corrupted, it requests a retransmission from Y
- The dataflow semantics requires Y to read a new data from X for the retransmission of its output to Z
- So we do not want to re-execute Y, we would like some partial re-execution
- But partial re-executions do not fit in standard dataflow MoCs

Proposed solution

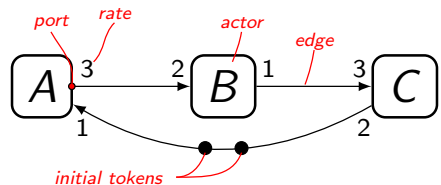
- We propose to use the Boolean Parametric Dataflow MoC (BPDF)
- The Boolean parameters will allow us to dynamically enable and disable edges and to trigger the requested retransmissions
- This would be impossible to achieve in SDF

[BPDF: Bebelis et al., EMSOFT 2013]

Background

SDF (1)

- An SDF graph consists of actors and edges (dataflow channels)
- Each actor firing consumes a fixed number of data on its input edges, computes, and produces a fixed number of data on its output edges



[[SDF: Lee and Messerschmitt, Proc. IEEE 1987]]

SDF (2)

- A static schedule can be produced and analyses can be performed at compile time (e.g., boundedness, liveness, throughput, latency, and buffer analysis)
- A non-empty sequence that returns the graph to its initial state is called an iteration.
- The basic repetition vector indicates the number of firings of each actor in one iteration.

SDF (3)

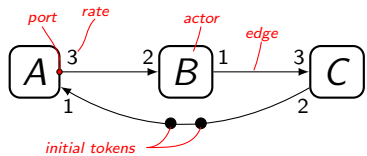
- The basic repetition vector is obtained by solving the system of balance equations
- For the edge $X \xrightarrow{p \quad q} Y$ the equation is $\#X.p = \#Y.q$: all tokens produced must be consumed within the same iteration
- If a non-null solution exists, then the graph is called consistent
- Example: $[\#A = 2; \#B = 3; \#C = 1]$
- Consistency ensures that the graph can be executed in bounded memory

SDF (4)

- Deadlock/liveness analysis checks that the graph admits a live schedule
- Any SDF graph with no directed cycle is always live.
- Otherwise, in each directed cycle, there must be enough initial tokens

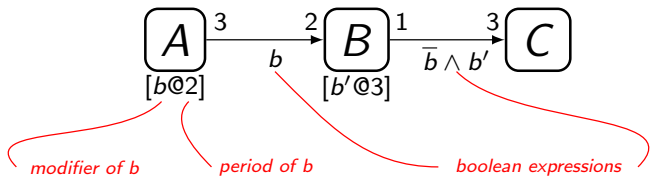
SDF (5)

- Among the admissible schedules, we distinguish Flat Single Appearance Schedules (SAS), where once factorized each actor appears exactly once.
- Example: $A^2; B^3; C^1$
- A cyclic graph admits an SAS if and only if each cycle includes at least one saturated edge



BPDF (1) – MoC definition

- Each edge can have an enabling boolean expression built with boolean parameters
- Each parameter is modified by its modifier with some period
- For boolean parameters, values change dynamically



BPDF (2) – Actor semantics

1. Each actor first reads the value of each boolean parameter that it needs
2. Then it consumes the required tokens on its enabled incoming edges
3. Then it computes its internal function (and state)
4. Then it produces the tokens on its enabled output edges
5. Finally, if it is the modifier of a parameter, it changes its value according to its period

SDF with lossy channels

Lossy SDF (1)

- Formally, an SDF graph cannot contain lossy channels
- Here we assume that an SDF graph with lossy channels behaves exactly like SDF (its semantic is given by SDF)
- On each lossy channel, tokens are eventually delivered correctly after a finite number of retransmissions

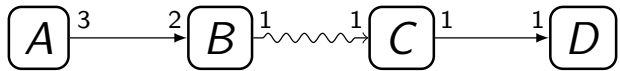
Lossy SDF (2)

- Example: $X \to Y \rightsquigarrow Z \to T$
- We divide the execution of this graph into three phases:

1. The upstream phase: X–Y part
2. The lossy phase: Y–Z part
   repeated until the necessary tokens are received correctly
3. And the downstream phase: Z–T part

From SDF with lossy channels to BPDF

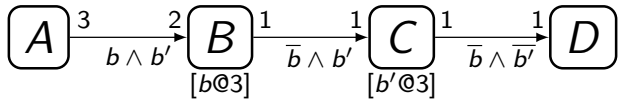
From Lossy SDF to BPDF (1)



The three phases are:

- Upstream phase : $A^2; B^3$
- Lossy phase : $B^3; C^3$
- Downstream phase : $C^3; D^3$

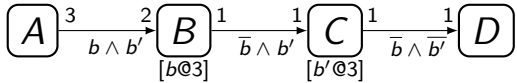
From Lossy SDF to BPDF (2)



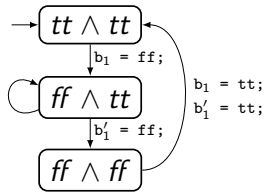
The three phases are implemented with two boolean parameters:

- Upstream phase : $b = tt, b' = tt$
- Lossy phase : $b = ff, b' = tt$
- Downstream phase : $b = ff, b' = ff$

From Lossy SDF to BPDF (3)



Phase	Partial schedule	b_1	b'_1
1 (upstream)	$\{A^2; B^3\}$	tt	tt
2 (lossy)	$\{B^3; C^3\}^*$	ff	tt
3 (downstream)	$\{C^3; D^3\}$	ff	ff



Actor B	Actor C
`if (phase==1) then b1=ff;`	`if (phase==2) then b'1=ff;`
`if (phase==3) then b1=tt;`	`if (phase==3) then b'1=tt;`

General translation algorithm (1)

1. Compute a flat SAS S_G for G
2. S_G induces a total order on the actors of G so we number them $V_1, \ldots, V_n$
3. S_G induces a total order on the edges of G so we number them $E_1, \ldots, E_p$
4. The total order on the edges is projected on the set of lossy channels so we number them $L_1, \ldots, L_q$

General translation algorithm (2)

5. For each lossy channel L_i we add two boolean parameters b_i, b'_i
6. Each edge E_i gets the enabling boolean expr. $bc_1 \wedge bc_2 \wedge bc_3$
7. bc_1 accounts for all lossy channels that are after E_i in S_G:
   $bc_1 = \bigwedge_{j=u}^{q}(b_j \wedge b'_j)$ *(E_i is in the upstream phase of L_j)*
8. bc_2 accounts for the fact that E_i may be itself a lossy channel:
   if $\exists j$ such that $E_i = L_j$, then $bc_2 = \overline{b_j} \wedge b'_j$ else $bc_2 = tt$
9. bc_3 accounts for all lossy channels that are before E_i in S_G:
   $bc_3 = \bigwedge_{j=1}^{\ell}(\overline{b_j} \wedge \overline{b'_j})$ *(E_i is in the downstream phase of L_j)*

Sequencing the phases in the general case

For each lossy channel L_i:

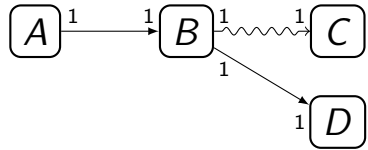
General case ($1 \leq i \leq q$)	
$src(L_i)$	$dst(L_i)$
`if (phase==2*i-1) then` b_i`=ff;`	`if (phase==2*i) then` b'_i`=ff;`
`if (phase==last) then` b_i`=tt;`	`if (phase==last) then` b'_i`=tt;`

Underlying mechanisms

- The communication system layer provides information about token corruption and/or loss
- Token corruption: error-detecting codes
- Token loss: time out mechanisms such as Automatic Repeat-Request (ARQ) protocols, e.g., Stop-and-Wait ARQ, Go-Back-N ARQ, or Selective Repeat ARQ

Future work and conclusion

Choice of the flat SAS



- This SDF graph admits two flat SAS
- $A; B; C; D$: Edge BD belongs to the downstream phase
  $\Longrightarrow$ three phases!
- $A; B; D; C$: Edge BD belongs to the upstream phase
  $\Longrightarrow$ two phases (because the downstream phase is empty)!
- We could optimize the number of phases

Parallel schedules

- Our translatetion algorithm relies on a flat SAS, which is sequential
- With a parallel schedule, we generalize the phases to cones
- Downstream cone: the set of all predecessor edges
- Upstream cone: the set of all successor edges

Conclusion

- Scenario-Aware DataFlow: PFSM-SADF [Skelin et al., EMSOFT 2015]
- But the phases must be coded explicitely versus implicitely with the boolean parameters
- Plus BPDF also offers integer parameters for variable input and output rates
- Many open questions: implementation, performance evaluation, ...