Modular Analysis of Petri Nets

Laure Petrucci

LIPN, CNRS UMR 7030
Université Paris 13
Villetaneuse
FRANCE

Motivation

Systems nowadays very large $\Rightarrow$ high-level Petri nets

- more compact representation
- data manipulation

Motivation

Systems nowadays very large $\Rightarrow$ high-level Petri nets

- more compact representation
- data manipulation

Main analysis technique: state spaces $\Rightarrow$ state space explosion problem

To cope with that, different state space reduction methods have been introduced.

Motivation

- modular design of systems
- hierarchical CP nets [Jensen 1992], modular Petri nets [Christensen Petrucci 2000]

Motivation

- modular design of systems
- hierarchical CP nets [Jensen 1992], modular Petri nets [Christensen Petrucci 2000]
- advantages:
  - model structuring
  - functional decomposition of the system modelled
  - reusability of system parts

Motivation

- modular design of systems
- hierarchical CP nets [Jensen 1992], modular Petri nets [Christensen Petrucci 2000]
- advantages:
  - model structuring
  - functional decomposition of the system modelled
  - reusability of system parts
- modular verification
  - take advantage of system architecture
  - provide efficient and flexible analysis techniques
  - composition of subsystems invariants
  - modular state spaces
  - modular/compositional/incremental verification

Outline

- Modular state spaces
  - Construction algorithm
  - Experimental results
  - Modular verification of properties
  - Timed extensions
- Compositional verification
  - Observation graph
- Incremental approach
  - Model refinement
  - State space construction
- Conclusion
- Ongoing and future work

Modular State Spaces

Introduced in [Christensen Petrucci 1995, 2000], further refined in [Lakos Petrucci 2004] to:

- take advantage of modular design
- avoid interleaving
- reduce memory consumption

Modular State Spaces

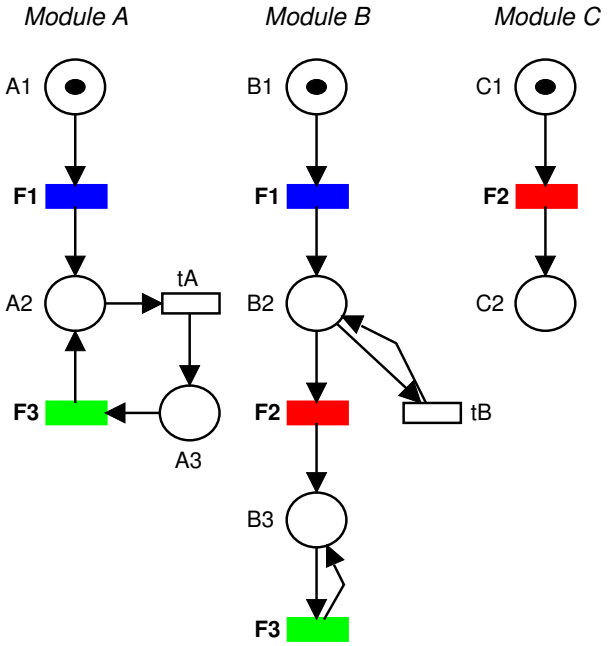
Introduced in [Christensen Petrucci 1995, 2000], further refined in [Lakos Petrucci 2004] to:

- take advantage of modular design
- avoid interleaving
- reduce memory consumption

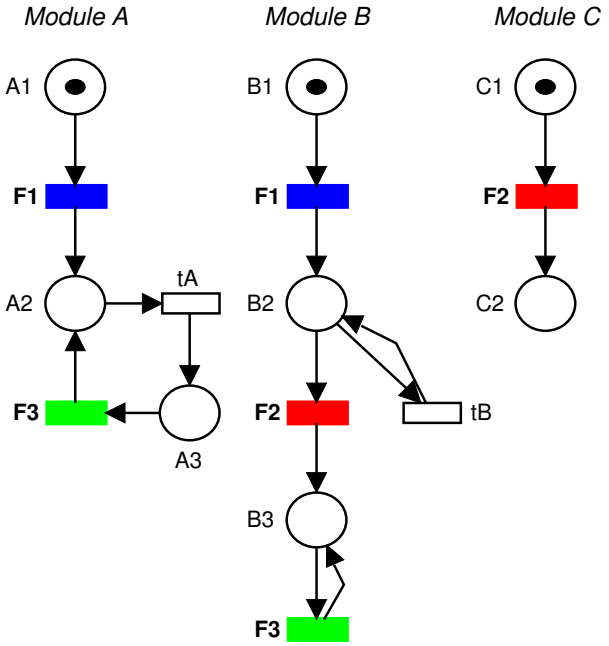
Achieved by:

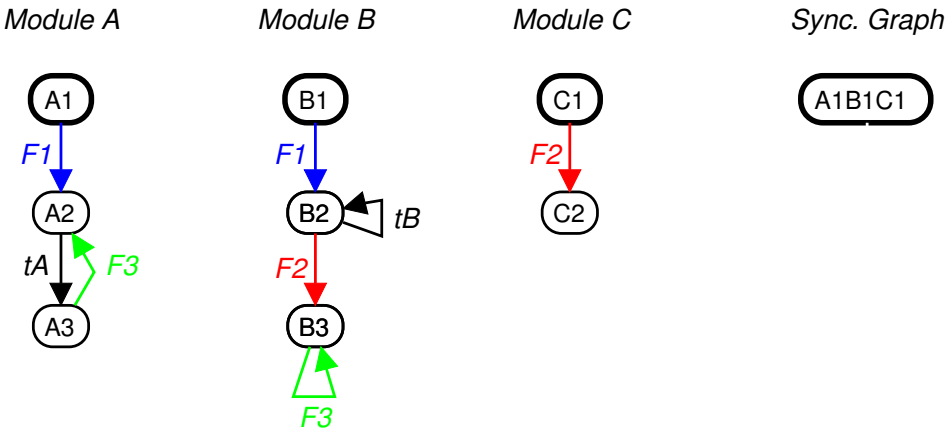
- keeping a local state space per module (local states and transitions)
- constructing a synchronisation graph to capture synchronisation between modules.

Example

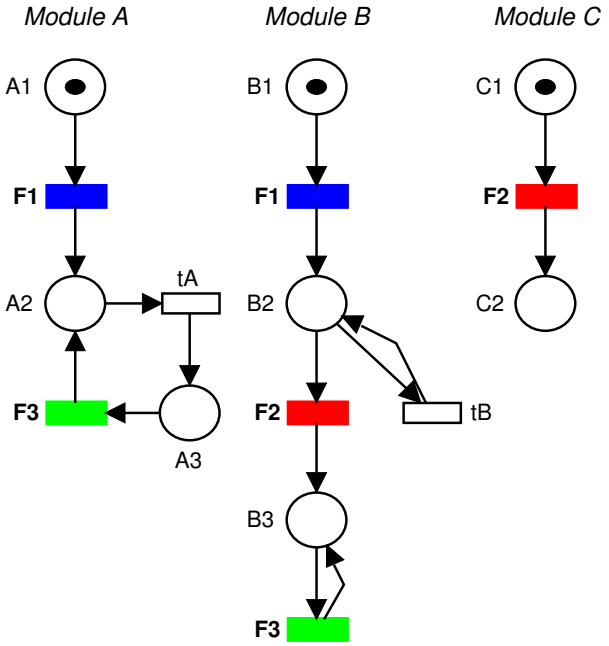

Module A
Module B
Module C
A1
B1
C1
F1
F1
F2
tA
A2
B2
C2
F3
F2
tB
A3
B3
F3

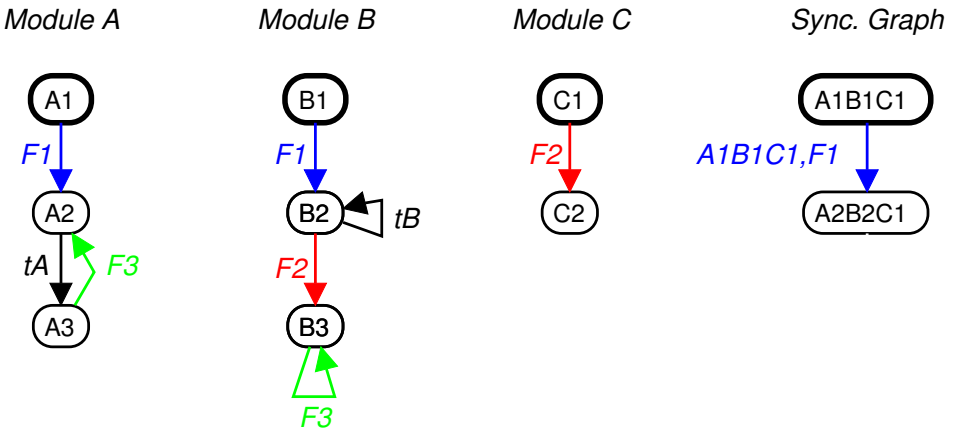
Example


Module A
Module B
Module C
A1
F1
A2
tA
F3
A3
B1
F1
B2
F2
tB
B3
F3
C1
F2
C2

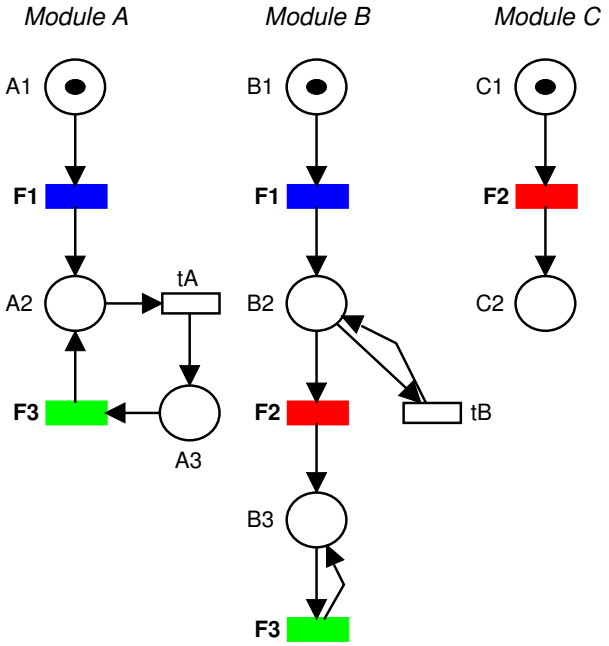

Module A
Module B
Module C
Sync. Graph
A1
F1
A2
tA
F3
A3
B1
F1
B2
tB
F2
B3
F3
C1
F2
C2
A1B1C1

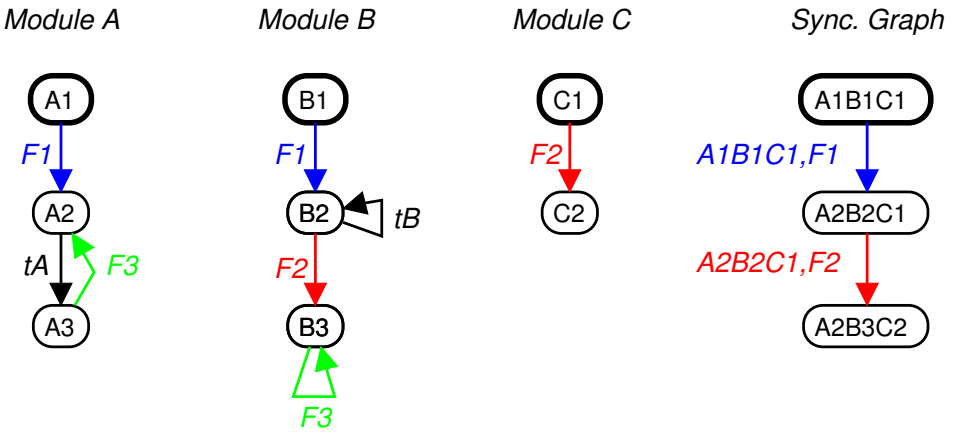
Example


Module A
Module B
Module C
A1
F1
A2
tA
F3
A3
B1
F1
B2
F2
tB
B3
F3
C1
F2
C2

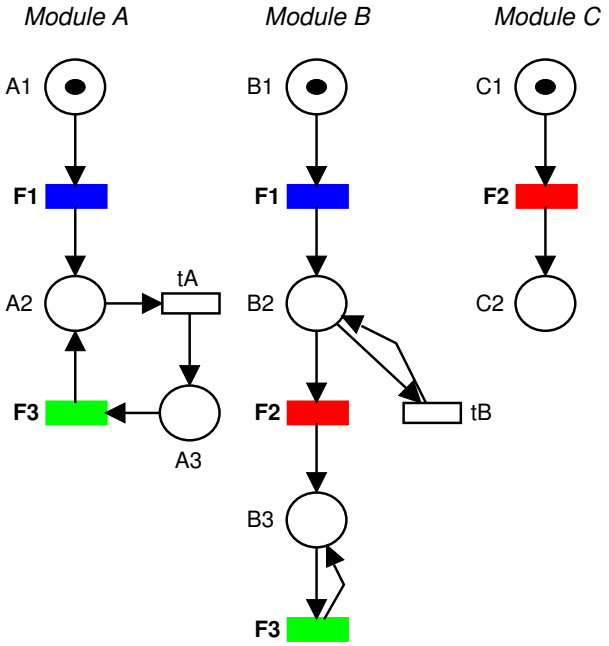

Module A
Module B
Module C
Sync. Graph
A1
F1
A2
tA
F3
A3
B1
F1
B2
tB
F2
B3
F3
C1
F2
C2
A1B1C1
A1B1C1,F1
A2B2C1

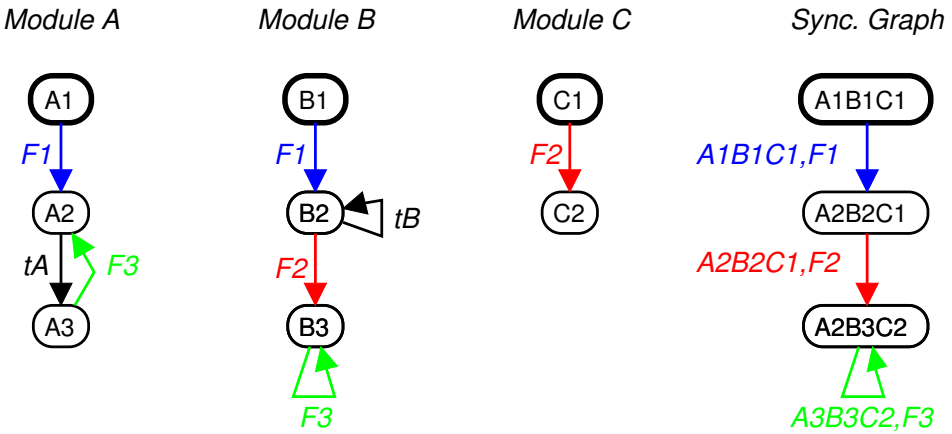
Example


Module A
Module B
Module C
A1
F1
A2
tA
F3
A3
B1
F1
B2
F2
tB
B3
F3
C1
F2
C2

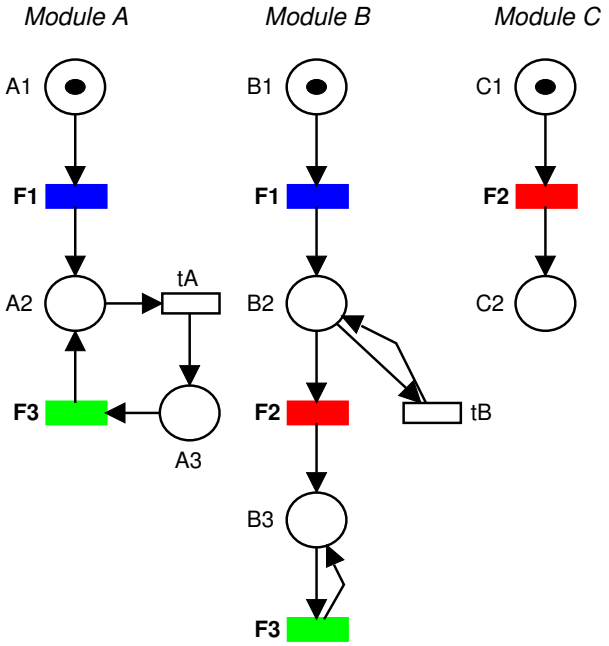

Module A
Module B
Module C
Sync. Graph
A1
F1
A2
tA
F3
A3
B1
F1
B2
tB
F2
B3
F3
C1
F2
C2
A1B1C1
A1B1C1,F1
A2B2C1
A2B2C1,F2
A2B3C2

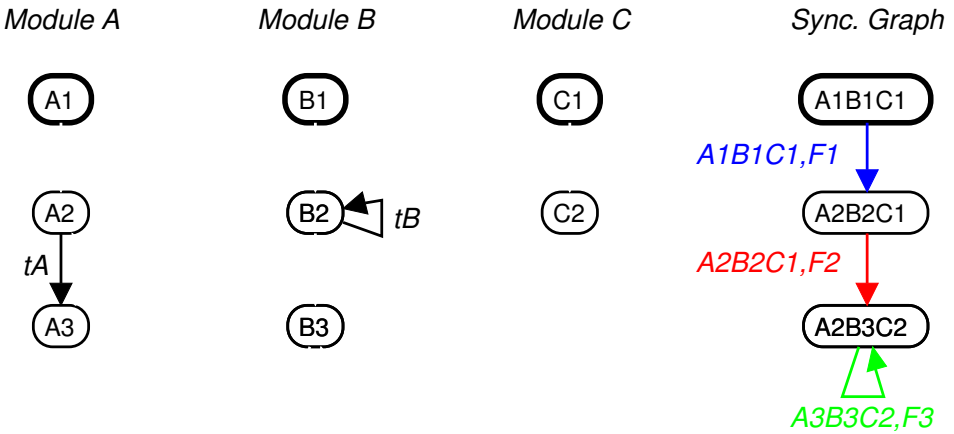
Example


Module A
Module B
Module C
A1
F1
A2
tA
F3
A3
B1
F1
B2
F2
tB
B3
F3
C1
F2
C2


Module A
A1
F1
A2
tA
F3
A3
Module B
B1
F1
B2
tB
F2
B3
F3
Module C
C1
F2
C2
Sync. Graph
A1B1C1
A1B1C1,F1
A2B2C1
A2B2C1,F2
A2B3C2
A3B3C2,F3

Example


Module A
Module B
Module C
A1
F1
A2
tA
F3
A3
B1
F1
B2
F2
tB
B3
F3
C1
F2
C2


Module A
Module B
Module C
Sync. Graph
A1
A2
tA
A3
B1
B2
tB
B3
C1
C2
A1B1C1
A1B1C1,F1
A2B2C1
A2B2C1,F2
A2B3C2
A3B3C2,F3

Construction algorithm

1. construct possible local state space
   - could be done on-the-fly, but framework easy to adapt to models other than Petri nets
   - construct the state spaces of modules
   - construct SCCs of state spaces without synchronisation transitions
   - unreachable states might be present that will be deleted later
2. compute synchronisation graph SG
3. delete local unreachable parts
   - delete synchronised transitions in local state spaces
   - mark states locally reachable from nodes in SG
   - delete unmarked nodes and connected arcs

Computing SG

- construct the initial node
- for each unprocessed node
  - mark local successors
  - for each synchronised transition tf
    - consider only modules participating in the synchronisation
    - identify marked nodes enabling tf
    - add the successor nodes and corresponding arcs

Experiments [Petrucci 2005]

Model	param	Occurrence Graph	Modular State Space
5 AGVs		$30,965,760$	900
Database	n	$n \times 3^{n-1}$	$6n+3$
Philosophers	2	3	11
	3	4	16
	n	$N_{OG}(n-1)+N_{OG}(n-2)$	$N_{OG}(n)+4n$
Poisoned philosophers	2	21	30
	3	99	99
	n	$4N_{OG}(n-1)+$ $3N_{OG}(n-2)+6$	$4N_{MSS}(n-1)+$ $N_{MSS}(n-2)+$ $8n+4$
Railway	n	$4(n^2+n+1)$	$\frac{n(n+1)}{2}+5n+10$

Properties

- analyse the system without unfolding the modular state space
- algorithms for standard properties [Lakos Petrucci 2004]:
  - reachability
  - deadlocks
  - liveness
- generalised to cater for subsets of modules or transitions

Properties

- analyse the system without unfolding the modular state space
- algorithms for standard properties [Lakos Petrucci 2004]:
  - reachability
  - deadlocks
  - liveness
- generalised to cater for subsets of modules or transitions

Achieved by:

- decomposition of properties in local/global parts
- marking of nodes
- state spaces traversal

Timed Extensions

Extensions to :

- Timed nets [Lakos Petrucci 2005, 2006]: state spaces, global time
- Time nets [Mazouz Petrucci 2006]: T-states graphs, relative time

Timed Extensions

Extensions to :

- Timed nets [Lakos Petrucci 2005, 2006]: state spaces, global time
- Time nets [Mazouz Petrucci 2006]: T-states graphs, relative time

Difficulty: preemption of global vs. local transitions unpredictable

$\Rightarrow$ the modular state space does contain unreachable states

Timed Extensions - Experiments

Implementations:

- timed nets: prototype implementation in MARIA
- time nets: standalone implementation

Timed Extensions - Experiments

Implementations:

- timed nets: prototype implementation in MARIA
- time nets: standalone implementation

Satisfactory results that depend on:

- structuring into modules
- coupling between modules
- timing constraints
- scheduling of actions

Compositional verification

Verification of a LTL\X formula

- identify visible transitions required by the formula
- construct an observation graph
- verify the property

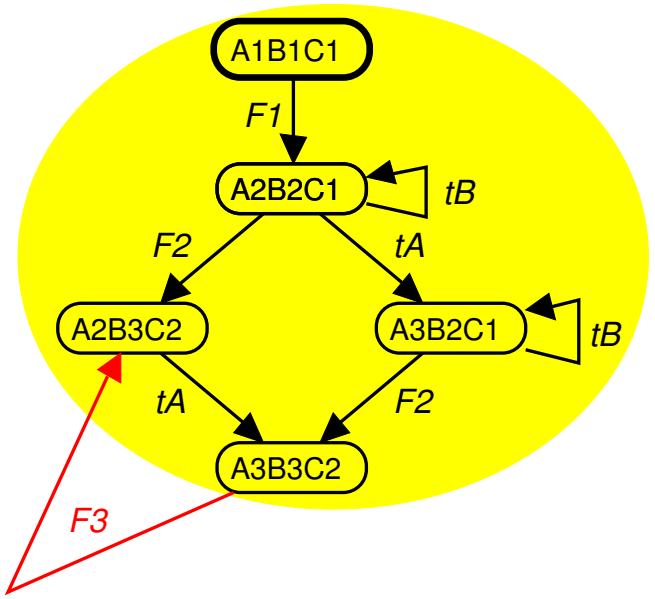
Pros: traversal of a single state space

Cons: the state space depends on the property to check

Proposals in [Klai 2003] and [Latvala Mäkelä 2004], very similar in essence

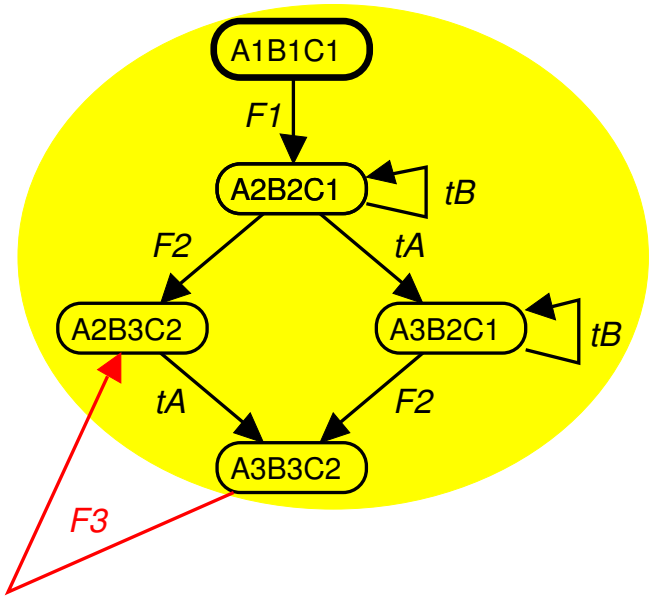
Example

F3 live

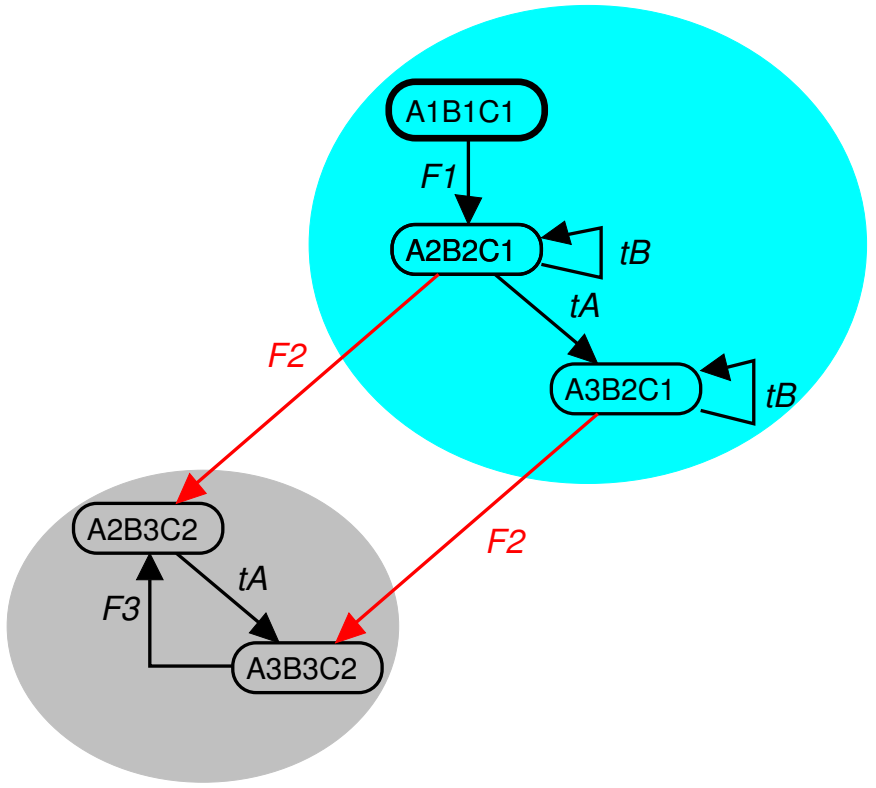


Example

F3 live

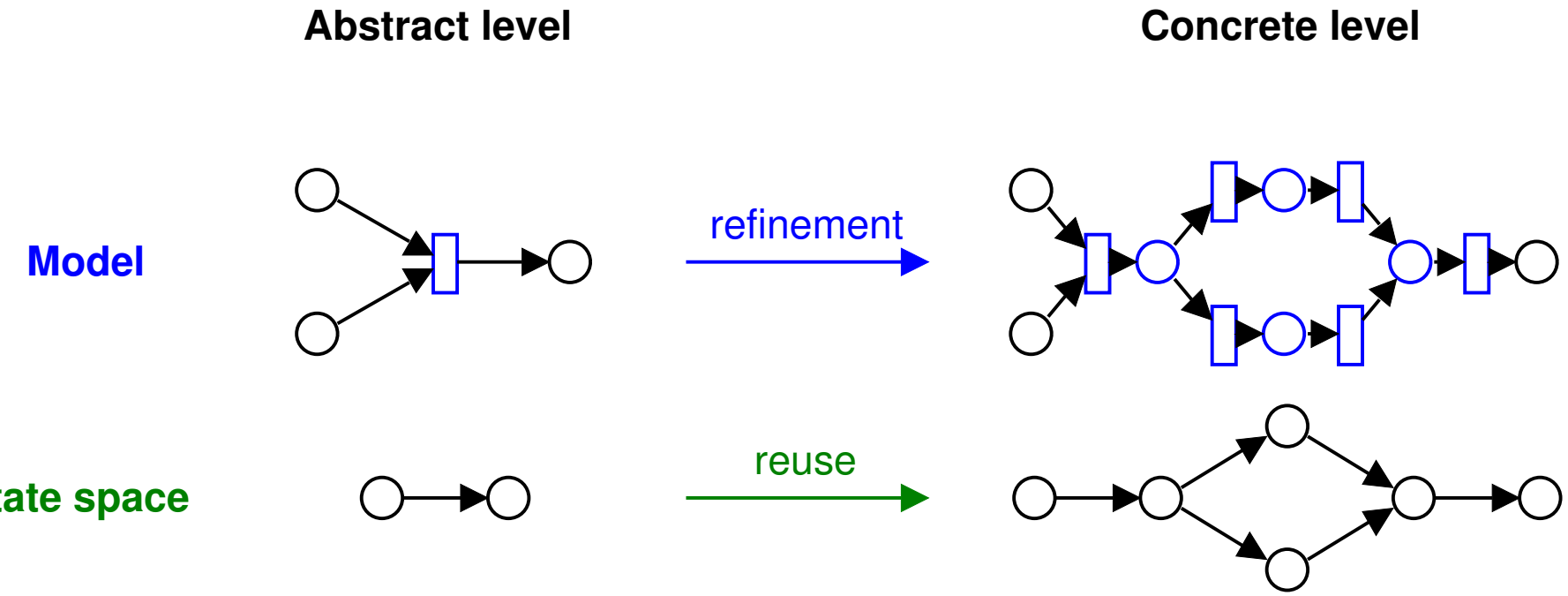


F2 not live



Incremental approach

[Lakos Lewis 2001]



Model refinement

Different types of refinement:

- Type refinement: simple refinements where each value of the refined type can be projected onto a value of the abstract type.
- Subnet refinement: augmenting a subnet with additional places, transitions and arcs. Each behaviour of the refined system must have a corresponding behaviour in the abstract system.
- Node refinement: replace a place or a transition by a subnet, such that each behaviour of the refined system has a corresponding behaviour in the abstract system.

Refined model state space

Use the abstract model state space to guide the refined model state space computation:

- some markings already computed
- some markings have to be extended
- less transitions enablings to check

Conclusion

- Several trends to take advantage of modularity
- Efficiency depends on the coupling of modules
- Techniques suitable for high-level and time(d) nets
- Refinement fits a model engineering design approach
- Other approaches (e.g. abstraction places, invariants)

Ongoing and future work

- Check properties using Modular state spaces with time (with C. Lakos and S. Mazouz)
- Data abstraction, refinement, parametric analysis (with J. Billington, C. Choppy, C. Lakos and M. Mayero)
- Distributed state space analysis (with C. Boukala and L. Kristensen)
- Case studies (Fieldbus protocol, with C. Lakos)