Concurrent Program Synthesis Based on Supervisory Control

Marian V. Iordache
School of Engineering and Eng. Tech.
LeTourneau University
Longview, TX 75607-7001

Panos J. Antsaklis
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

July 1, 2010

Motivation

- Tools for converting sequential programs to parallel programs are available.
- Often programs are not nearly as efficient as when written from the beginning as parallel programs.
- However, concurrent programming is known to be difficult.

Motivation

- Concurrent programming is difficult because of shared resources.
- These can cause *race conditions*.
- Race conditions denote the situations in which the result depends on the order of execution of concurrent operations.

Race Conditions

PROCESS #1

$$\begin{array}{llll} & & \vdots & \\ (1) & u & = & x \\ (2) & u & = & u+10 \\ (3) & x & = & u \\ & & \vdots & \end{array}$$

PROCESS #2

$$\begin{array}{llll} & & \vdots & \\ (a) & v & = & x \\ (b) & v & = & v+20 \\ (c) & x & = & v \\ & & \vdots & \end{array}$$

- Initial value of shared variable x is $x = 30$.
- Correct result: $x = 60$.
- Possible results:
  - $x = 40$ for operation sequence (1), (a), (2), (b), (c), (3).
  - $x = 50$ for operation sequence (1), (a), (2), (3), (b), (c).
  - $x = 60$ for operation sequence (1), (2), (3), (a), (b), (c).

Motivation

- *Locks* can be used to ensure that critical operation sequences are not interrupted.
  - Generally, it is difficult to find the best way to use locks in a program.
  - Improper use of locks may result in
    - deadlocks;
    - sequential execution of parallel code.
- Concurrent programs are hard to debug.

Proposed Solution

- Use a high level description of the problem (to avoid the complexity of low level details).

  The programmer describes what he would like to do.

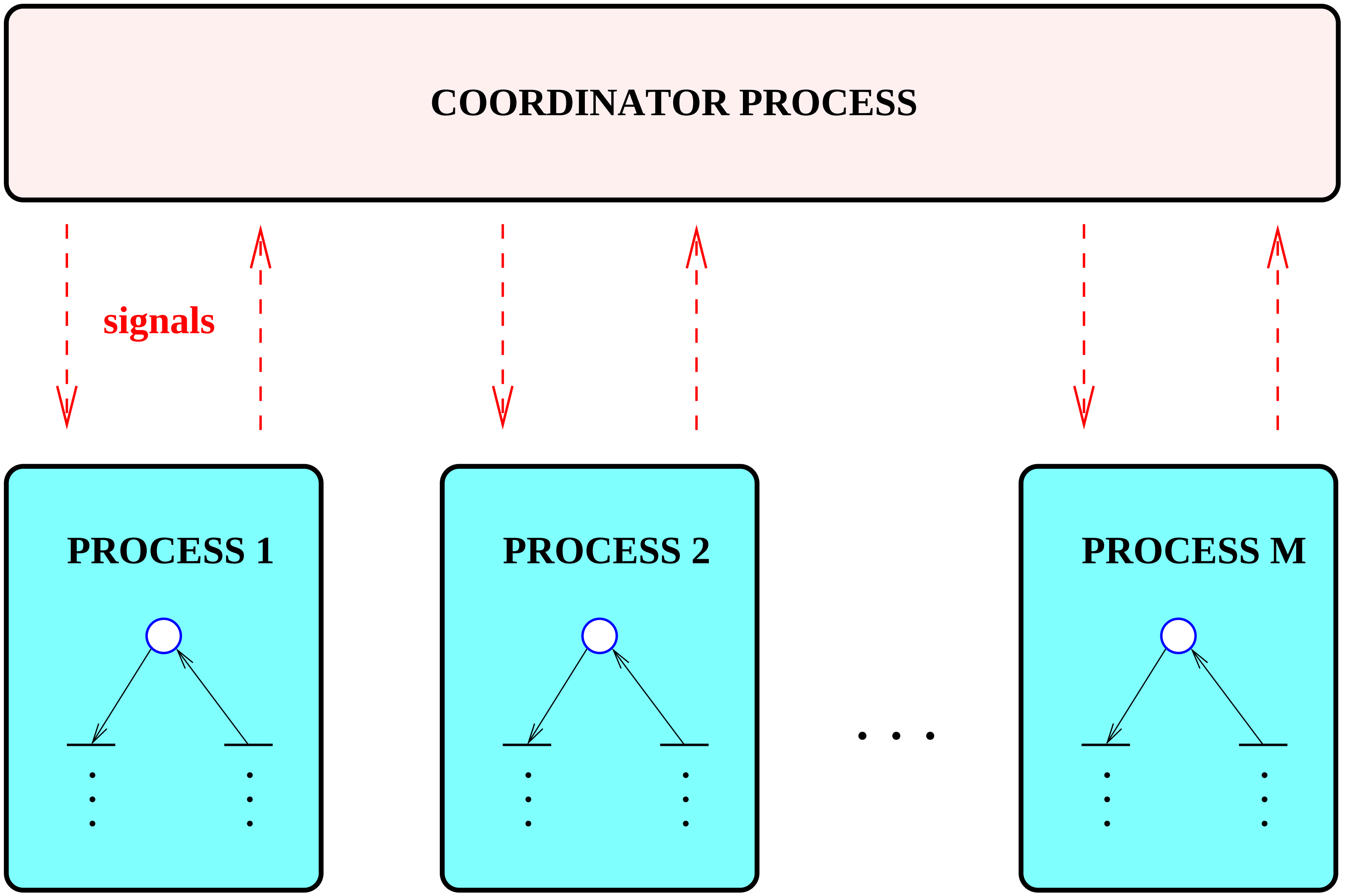
- Obtain a formal model from specs (Petri nets).

  This step is performed automatically.

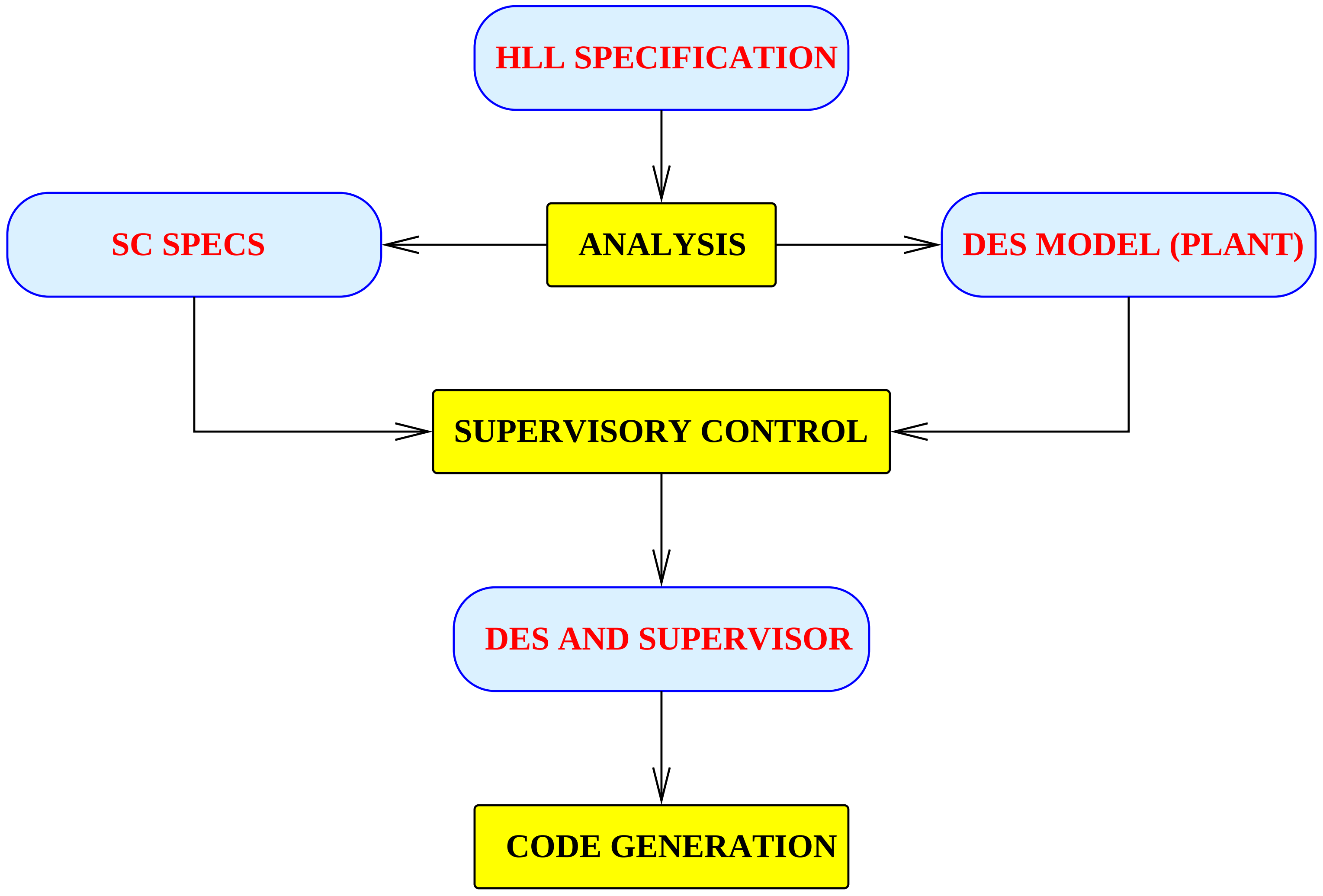
- Generate automatically the low level code that implements the specs.

  Formal approach: Supervisory Control.

Approach



Approach


HLL SPECIFICATION
SC SPECS
ANALYSIS
DES MODEL (PLANT)
SUPERVISORY CONTROL
DES AND SUPERVISOR
CODE GENERATION

Specification

The specification describes how the concurrent processes should be coordinated.

The specification describes the problem (what to do) not the solution (how to do).

Each process consists of blocks of low level code (functions).

The specification gives:

1. the blocks of low level code of each process;
2. constraints on the order of execution of these blocks.

A custom high level language (HLL) is used for the specification.

Note that high level descriptions do not define explicitly PNs.

Rather, PNs are automatically extracted from the specification.

Specification

Synchronization problem (adapted from [Downey, 2008]):

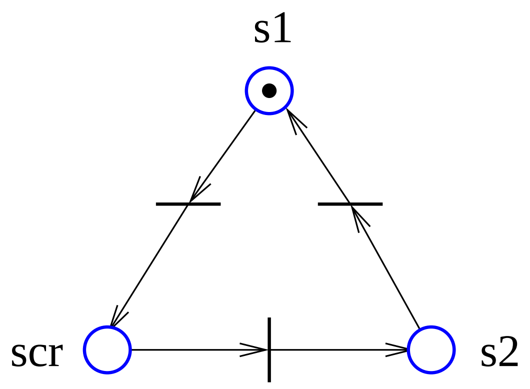
- Shared data accessed by reader, inserter, and deleter processes.
- At any time, only one inserter may modify the data.
- At any time, only one deleter may modify the data.
- Readers and inserters may not access the shared data at the same time as a deleter.

Specification

```
process p_reader {
  s1 to scr;
  scr to s2;
  s2 to s1;
}

process p_inserter = p_reader;  /* Initialize p_inserter */
p_inserter {  /* Enhance the DES of p_inserter */
  scr <= 1;  /* this restricts both p_inserter and its copies */
}

process p_deleter = p_inserter;
p_deleter.scr EXCLUDES p_inserter.scr p_reader.scr;

COPIES OF p_inserter:  pi1, pi2;
COPIES OF p_deleter:  pd1, pd2;
COPIES OF p_reader:  pr1, pr2, pr3, pr4;
```

Specification

```
process p_reader {
   s1 to scr;
   scr to s2;
   s2 to s1;
}
```



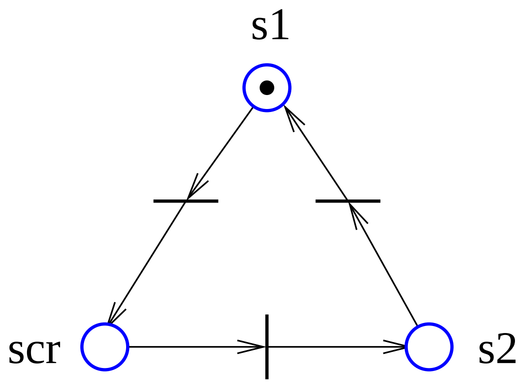
READER PROCESS

- This segment of code defines a process and a process type.
- *Process*: a sequential program.
- Processes implemented as *executable programs* or *threads*.
- Processes having the same executable code are said to have the same *process type*.
- Process types: modeled by PN structures.
- Processes: modeled by tokens.
- PN places model stages of execution of a process.

Specification Example

- The inserter process has the same structure as the reader process.
- Thus, it is initialzed to equal the reader process.

```
process p_inserter = p_reader;
```

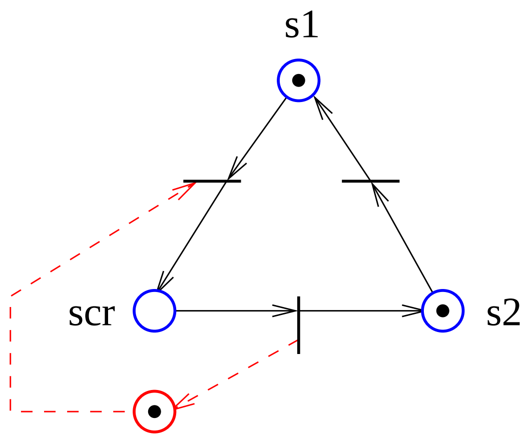


INSERTER PROCESS

- Only one inserter may be in the critical section.
- The following restricts *all* processes having the type of p_inserter.

```
p_inserter {
  scr <= 1;
}
```

- The constraint is implemented by the supervisor process.



TWO INSERTERS WITH SUPERVISOR

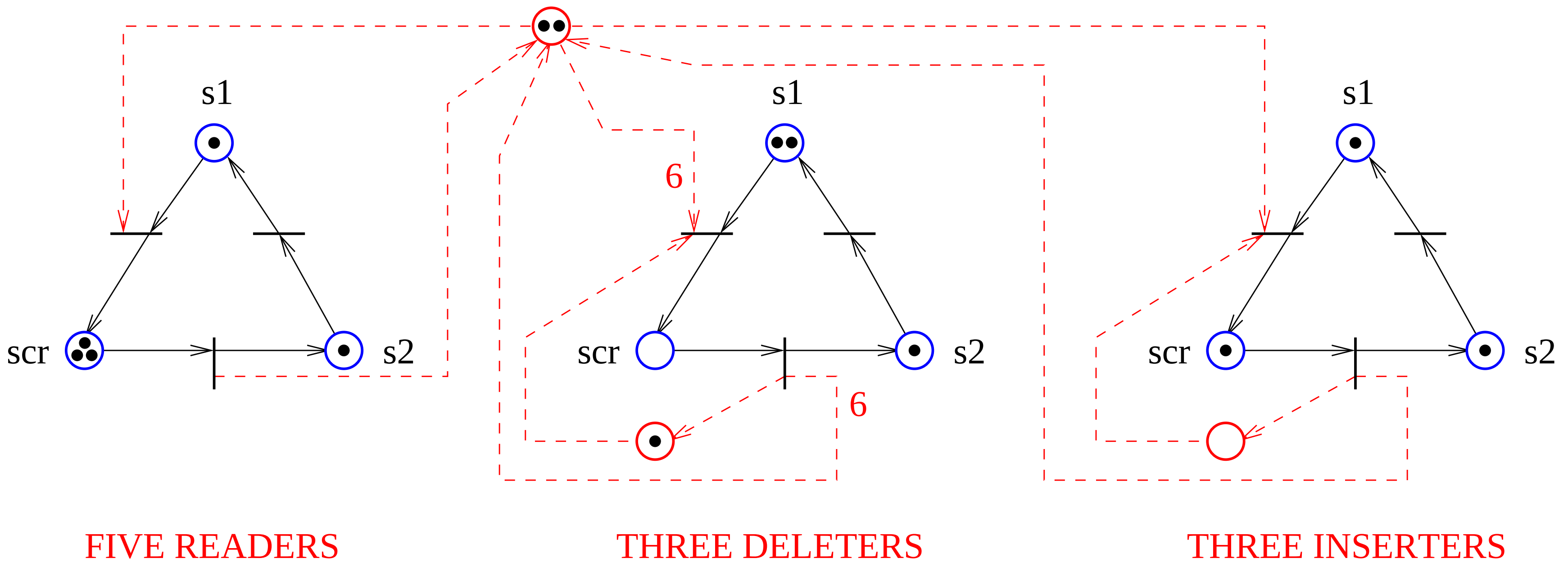
Specification

- The deleter process has the same structure as the inserter process.
- Thus, it is initialzed to equal the reader process.

```
process p_deleter = p_inserter;
```

- A deleter may not be in the critical region together with a reader or an inserter.

```
p_deleter.scr EXCLUDES p_inserter.scr p_reader.scr;
```



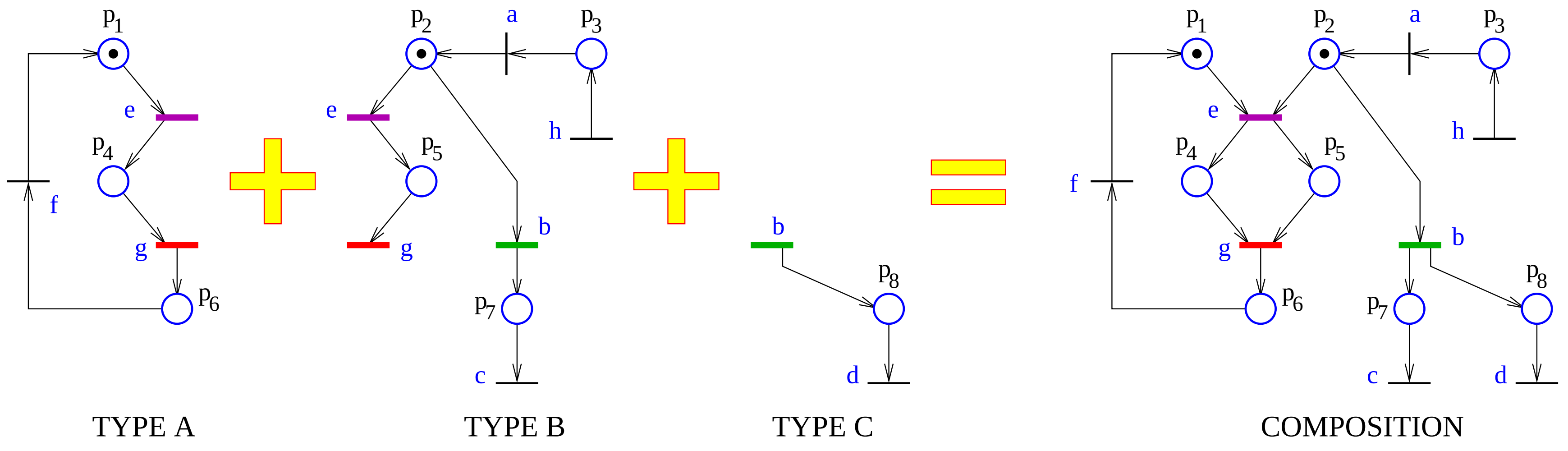
Specification — Example

- Each process copy counts as an additional token in the PN of the process type.

```
COPIES OF p_inserter:  pi1, pi2;
COPIES OF p_deleter:  pd1, pd2;
COPIES OF p_reader:  pr1, pr2, pr3, pr4;
```

- In this example, the plant consists of state machines.
- In general, this may not be the case.

Petri Nets — Synchronization

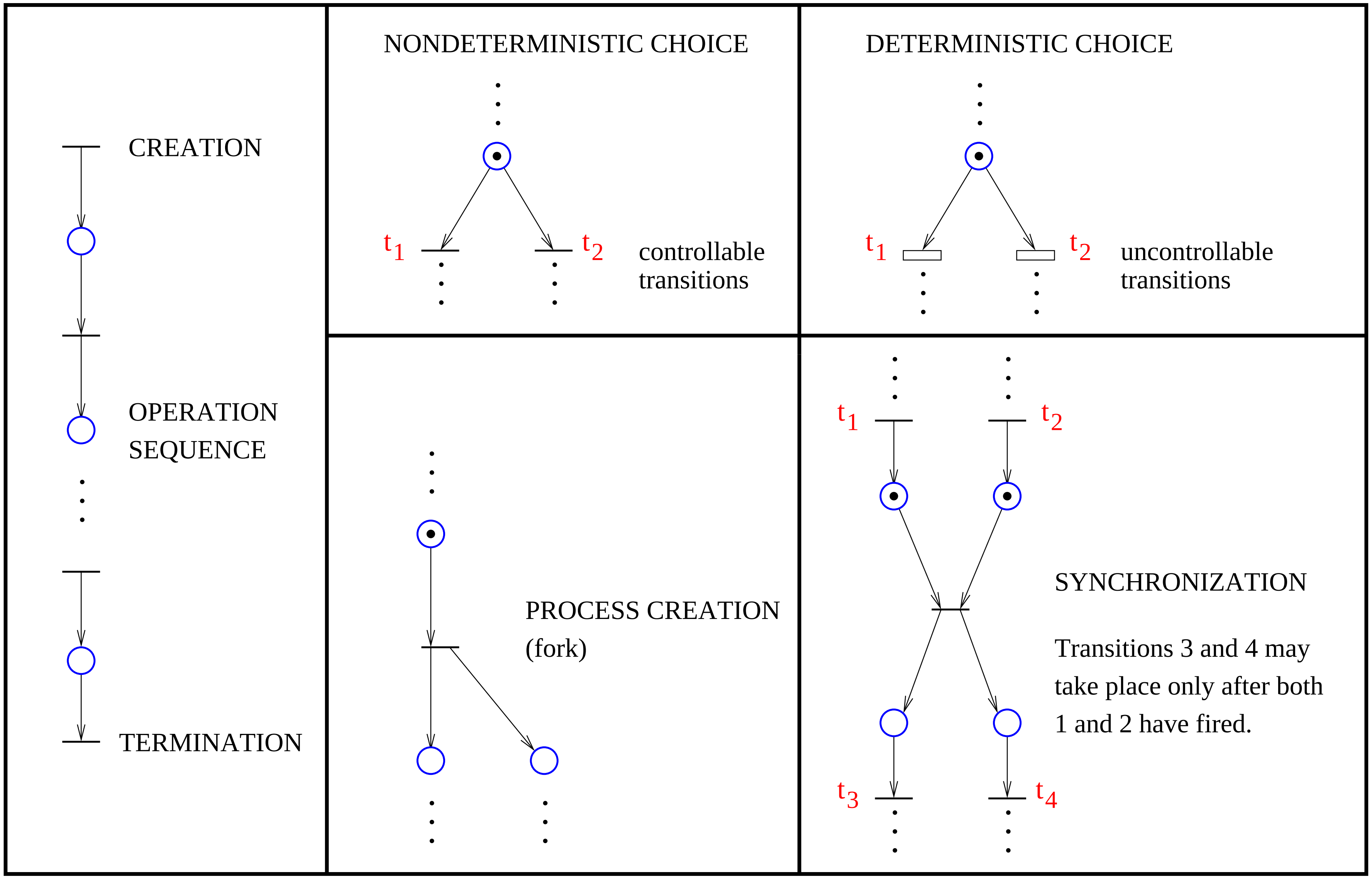


- Interprocess synchronization is possible.

  Synchronizations $\longrightarrow$ PNs that are not state machines.

- Sink transitions: model process termination.

- Source transitions: model events that start processes.


CREATION
OPERATION SEQUENCE
TERMINATION
NONDETERMINISTIC CHOICE
t_1
t_2
controllable transitions
DETERMINISTIC CHOICE
t_1
t_2
uncontrollable transitions
PROCESS CREATION (fork)
t_1
t_2
SYNCHRONIZATION
Transitions 3 and 4 may take place only after both 1 and 2 have fired.
t_3
t_4

Application to Software

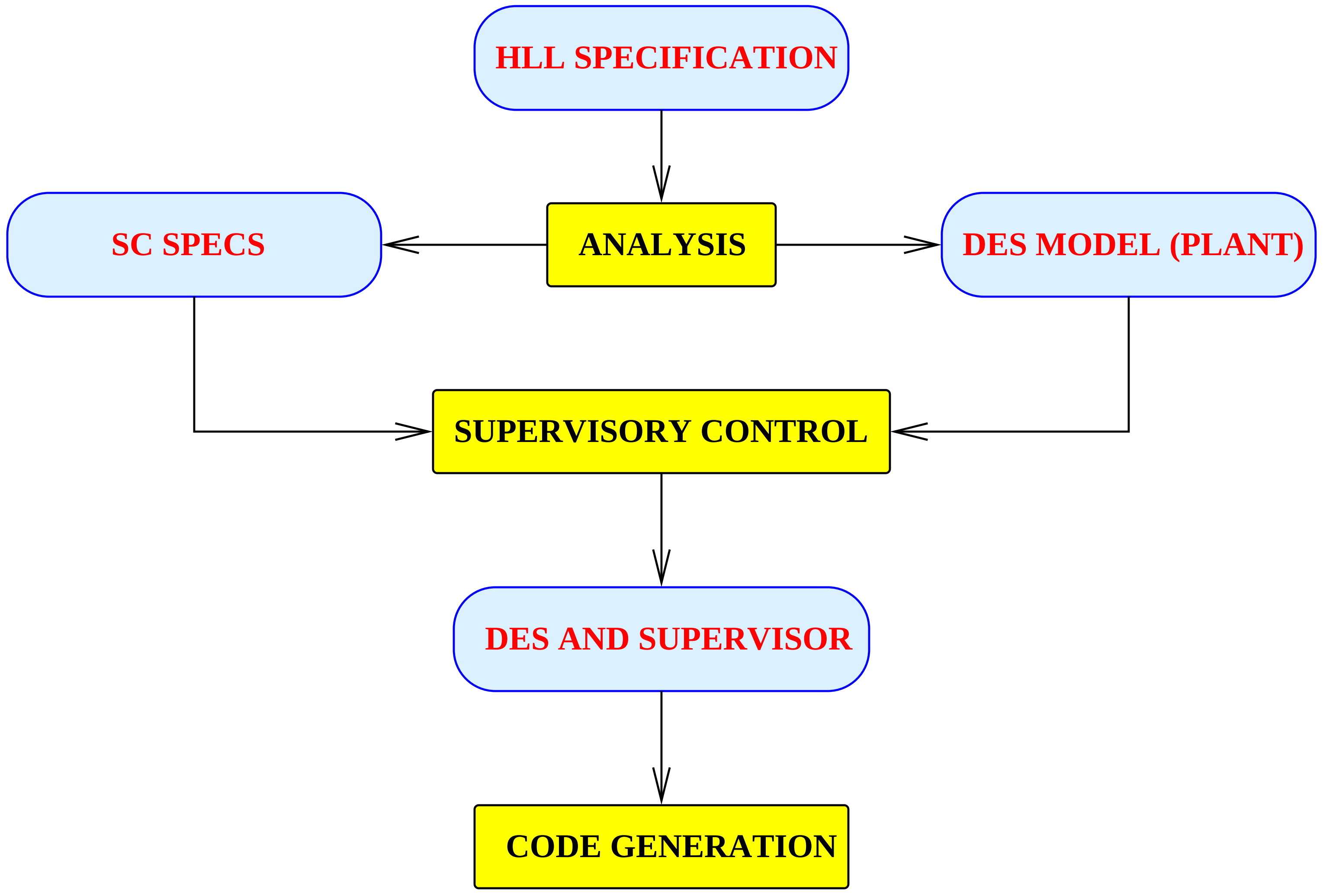
Specific software requirements:

- Execution with finite memory (SC should guarantee boundedness).
- Parallel code is not serialized (permissive supervisor).
- Software should be responsive (liveness + fairness).

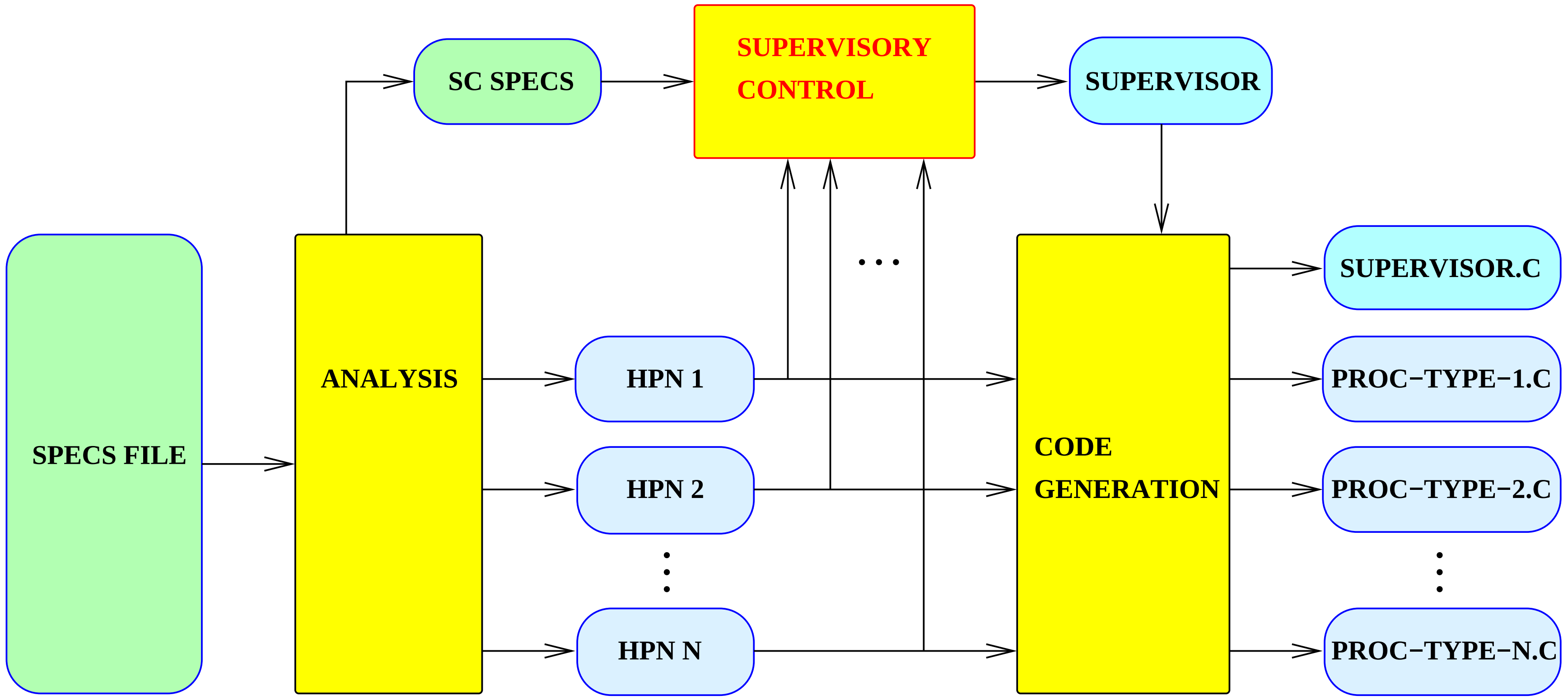
Disjunctive constraints important (so that general specs can be represented).

Methods for partial controllability important, esp. for liveness enforcement.

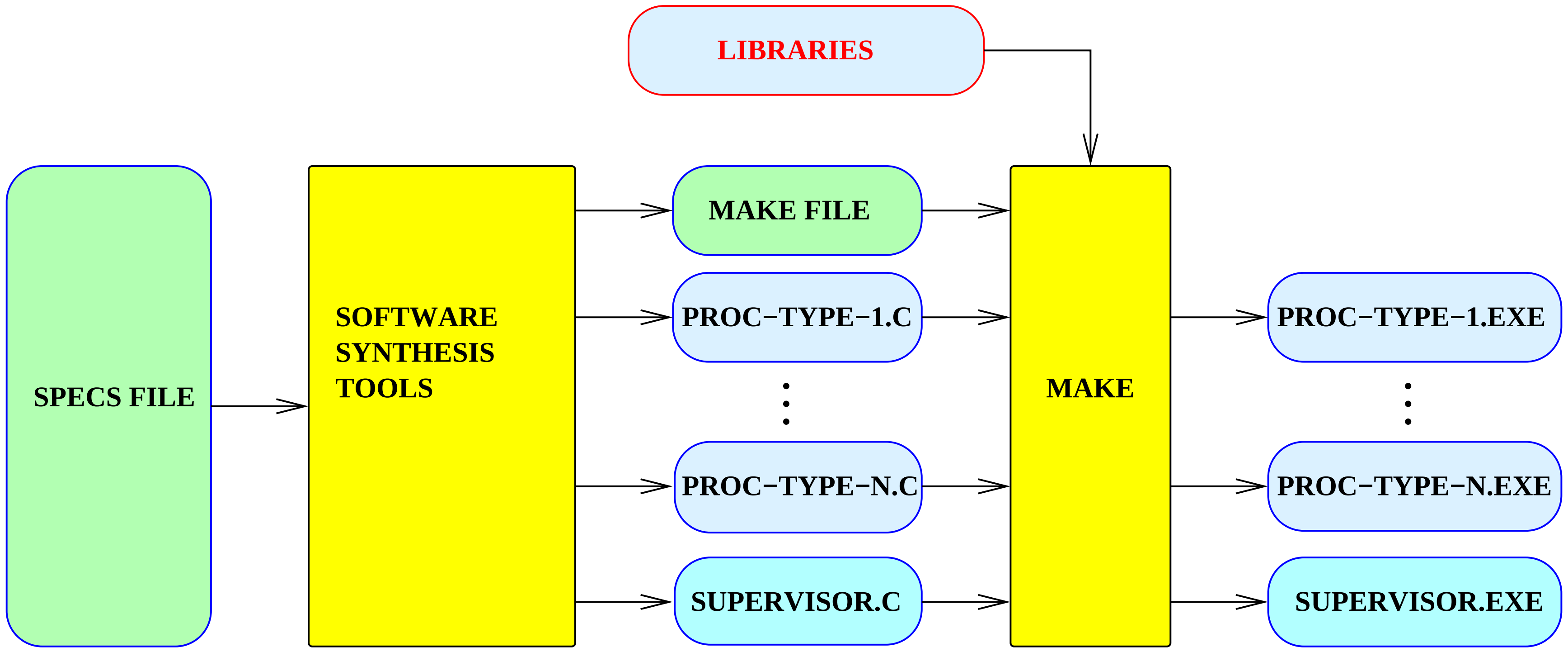
Approach


HLL SPECIFICATION
ANALYSIS
SC SPECS
DES MODEL (PLANT)
SUPERVISORY CONTROL
DES AND SUPERVISOR
CODE GENERATION

Approach


SC SPECS
SUPERVISORY CONTROL
SUPERVISOR
SPECS FILE
ANALYSIS
HPN 1
HPN 2
HPN N
CODE GENERATION
SUPERVISOR.C
PROC−TYPE−1.C
PROC−TYPE−2.C
PROC−TYPE−N.C

How It Is Used



Code Generation

- The implementation of the supervisor process is more involved.
- The supervisor process
  - performs synchronization of user processes;
  - implements the SC policies.
- Computational overhead should be small.
  - Limitation on the complexity of supervisor operations performed online.
  - Decentralized supervision can be used to distribute operations between several supervisors.
- Fairness: partially ensured by considering requests in the order in which they come.

Final Remarks

- Constraints on the operation of concurrent processes can be expressed as SC specifications.
- SC can help automate concurrent program synthesis.
- Software tools for concurrent program synthesis are under development.

  `http://www.letu.edu/people/marianiordache/acts`

- Of special interest are SC methods for
  - liveness and fairness constraints
  - disjunctive constraints
  - decentralized supervision.