CIF:modelling and analysis of heterogeneous models

Part II

Bert (D.A.) van Beek
Control Systems Technology Group – Systems Engineering
Eindhoven University of Technology
Contents

Part I

• Heterogeneous models in supervisory control system design
• CIF language and tools for
  • simulation, visualization
  • synthesis, verification
  • code generation for co-simulation and real-time control

Part II

• Case: baggage handling control system design
• Demo: CIF tooling for supervisory control synthesis
Airport baggage handling systems
Airport baggage handling systems
Baggage handling control system design
Conveyor physical model and controller visualization

Virtual positions in controller

Switch on/off for interactive simulation
Add/Remove products for interactive simulation
Simulation: adding products
Simulation: one bag with control head and tail
Simulation: second product gets head
Simulation: generate missing error
• System ABC represents wafer handling problem

• System consists of:
  • Three wafer handling stations C, B, A
  • Generator G
  • Exit E

• Two recipes:
  • G A B A E
  • G C B A E
CIF model ABC

plant automaton G
marked;

plant automaton A
Empty
marked;
ToB
ToE
Empty
marked;

plant automaton B
Empty
marked;
Full

plant automaton C
Empty
marked;
Full

Declarations
controllable void c_G_A;
controllable void c_G_C;
controllable void c_A_B;
controllable void c_B_A;
controllable void c_C_B;
controllable void c_A_E;
c_G_A!, c_G_C!
c_G_A?
c_B_A?
c_G_A,

A

E

G

C

B

A

E

G

C

B
CIF model demo and evaluation

- DEMO 01_ABC.sim.tooldef2
- Model ABC contains deadlock: when A and B are both occupied, and A still needs to go to B
CIF model ABC synthesis

- Synthesis ensures that from all reachable states a marker state can be reached.
- Mark the state where the system is empty => this empty state will be reachable after synthesis.
- In this way, synthesis removes the deadlock state.
- DEMO 02_ABC.sup.sim.tooldef2
supervisor automaton sup:
  alphabet c_G_A, c_G_C, c_A_B, c_A_E, c_B_A, c_C_B;
  location:
    initial;
    marked;
    edge c_A_B when true;
    edge c_A_E when true;
    edge c_B_A when true;
    edge c_C_B when not A.ToB;
    edge c_G_A when B.Empty;
    edge c_G_C when true;
end
CIF model ABC: from C to B uncontrollable

- **plant automaton A**
  - Empty
  - marked;
  - \( c_{A,B}! \)
  - \( c_{A,E}! \)
  - ToExit
  - \( c_{B,A}? \)
  - \( c_{G,A}? \)

- **plant automaton C**
  - Empty
  - marked;
  - \( c_{G,A}? \)
  - \( c_{A,B}? \)
  - ToExit
  - \( c_{G,C}? \)

- **plant automaton B**
  - Full
  - \( c_{B,A}! \)
  - \( c_{A,B}? \)
  - \( u_{C,B}? \)
  - ToExit
  - \( c_{G,C}? \)

- **plant automaton G**
  - marked;
  - \( c_{G,A}, c_{G,C}! \)

- **plant automaton Exit**
  - marked;
  - \( c_{A,E}? \)

- **Declarations**
  - controllable void \( c_{G,A}, c_{G,C} \);
  - controllable void \( c_{A,B} \);
  - controllable void \( c_{B,A} \);
  - controllable void \( c_{A,E} \);
  - uncontrollable void \( u_{C,B} \);
CIF model ABC: from C to B uncontrollable

- Controllable events can be disabled by the supervisor
- Uncontrollable events cannot be disabled by the supervisor
- When transition from C to B is uncontrollable, the supervisor cannot disable it
- To prevent *uncontrollable* event $u_{C\_B}$ from occurring, the supervisor must disable the *controllable* event that directly precedes $u_{C\_B}$
- DEMO 11_ABC_uncontrollable.sup.sim.tooldef2
supervisor automaton sup:

alphabet c_G_A, c_G_C, c_A_B, c_A_E, c_B_A; // u_C_B!!

location:
  initial;
  marked;
  edge c_A_B when true;
  edge c_A_E when true;
  edge c_B_A when true;
  edge c_G_A when B.Empty and C.Empty;
  edge c_G_C when not A.ToB; // see c_C_B below

end

// supervisor with c_C_B
  edge c_C_B when not A.ToB;
  edge c_G_A when B.Empty;
  edge c_G_C when true;
Zero wait for B to C

How to model the following requirement:

• product that is done processing in C and ready to go to B, *must never wait*?

Solution, add single requirement statement to the model:

• `requirement C.Full => B.Empty;`

DEMO 21_ABC_zero_wait.tooldef2
Add product numbers to visualize overtaking

DEMO 31_ABC_overtaking.sim.tooldef2
How to use synthesis to prevent overtaking

• Increasing product numbers can lead to an infinite state-space

Synthesis:

• Decrease product numbers by one for every leaving product. This ensures that the first product to leave always has number 1

• Add single requirement statement to the model:

```text
requirement A.ToE => A.i = 1;
```

DEMO 51_ABC_no_overtaking.sup.sim.tooldef2 (max prod_nr = 4)
CIF model ABC: No overtaking

- Declarations

- plant automaton G
  Declarations
disc prod_nr i = 1;
marked ;

- plant automaton A
  Declarations
disc prod_nr i = 0;
Empty marked ;

- plant automaton B
  Declarations
disc prod_nr i = 0;
Full

- plant automaton Exit
  Declarations
disc prod_nr i = 0;

- plant automaton C0
  Declarations
disc prod_nr i = 1;
Empty marked ;

- plant automaton C1
  Declarations
disc prod_nr i = 1;
Empty marked ;

Declarations

- type prod_nr = int[0..4];
- controllable prod_nr c_G.A;
- controllable prod_nr c_G.C0;
- controllable prod_nr c_G.C1;
- controllable prod_nr c_A.B;
- controllable prod_nr c_B.A;
- controllable prod_nr c_C0.B;
- controllable prod_nr c_C1.B;
- controllable prod_nr c_A.E;

requirement invariant A.ToE => A.i = 1;

Concluding remarks

- CIF well suited to analysis and design of complex heterogeneous systems:
  - Very expressive hybrid automata based formalism.
  - Rich toolset.

- Applied on a wide range of high tech industrial systems.

- CIF uniquely integrates supervisory control system *synthesis* in a complete model based supervisory control system design toolchain.

- Data-based synthesis algorithm now based on BDDs.

- Models that previously could not be synthesized are now synthesized in (milli-)seconds.