GRL: a Formal Language for the Specification of GALS Systems

Fatma Jebali, Frédéric Lang, Radu Mateescu
Inria – LIG – Grenoble, France

ICFEM 2014
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously

Synchronous systems
- Several components, one common clock
- Instantaneous computations and communications
- Deterministic behaviour
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously

Synchronous systems
– Several components, one common clock
– Instantaneous computations and communications
– Deterministic behaviour

Asynchronous composition
– Several synchronous systems, different speeds
– Arbitrary delays in communications
– Nondeterministic behaviour
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously

Synchronous systems
– Several components, one common clock
– Instantaneous computations and communications
– Deterministic behaviour

Asynchronous composition
– Several synchronous systems, different speeds
– Arbitrary delays in communications
– Nondeterministic behaviour
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously

Synchronous systems
- Several components, one common clock
- Instantaneous computations and communications
- Deterministic behaviour

Asynchronous composition
- Several synchronous systems, different speeds
- Arbitrary delays in communications
- Nondeterministic behaviour
GALS: Globally Asynchronous, Locally Synchronous

A set of synchronous systems composed asynchronously

Synchronous systems
- Several components, one common clock
- Instantaneous computations and communications
- Deterministic behaviour

Asynchronous composition
- Several synchronous systems, different speeds
- Arbitrary delays in communications
- Nondeterministic behaviour
Formal Verification of GALS Systems

Problem:
- Hard to design and debug
- Safety-critical applications

Formal modeling and verification:
- Powerful automatic tools
- Correctness of the design process

However:
- Expertise in formal methods required
- Scalability to industrial-size applications

Solution:
GRL (GALS Representation Language)
Rationale for GRL

(GALS Representation Language)

User convenience
- Unified language (synchronous and asynchronous)
- Modular modeling
- Abstraction
- Easy-to-use

Efficient formal verification
- Formal semantics
- Pivot language (industrial tools, CADP [1] toolbox)

[1] Construction and Analysis of Distributed Processes
http://cadp.inria.fr/
GRL in a nutshell

**Synchronous systems**
- **Blocks**: synchronous behaviour
- Based on the dataflow model

**Asynchronous composition**
- **Media**: communication between blocks
- **Environments**: external constraints
- Inspired by process algebraic languages

**Imperative flavour**
Running Example

Flight Control System (FCS)
Airbus
Blocks

Cyclic behaviour (active):

- Discrete deterministic steps
  1. Consume inputs
  2. Compute a reaction
  3. Produce outputs

- Memory maintained: **permanent** variables

- Atomic

Composition of subblocks
Blocks

Cyclic behaviour (active):

- Discrete deterministic steps
  1. Consume inputs
  2. Compute a reaction
  3. Produce outputs

- Memory maintained: permanent variables
- Atomic

Composition of subblocks

Receive, Send: asynchronous communication
Blocks: Simple Example

block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
      NUM [3] as B03,
      AND as B04,

temp c1 : bool, c2 : nat

  B03 (?c2);
  B02 (_, Sensor, c2, ?c1);
  B04 (Switch; c1; ?Is_On)

end block
**Blocks: Simple Example**

```plaintext
block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
    NUM [3] as B03,
    AND as B04,

temp c1 : bool, c2 : nat

B03 (?c2);
B02 (_, Sensor, c2; ?c1);
B04 (Switch; c1; ?Is_On)

end block
```

Physical interactions
**Blocks: Simple Example**

```plaintext
block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
    NUM [3] as B03,
    AND as B04,

temp c1 : bool, c2 : nat

B03 (?c2);
B02 (_, Sensor, c2; ?c1);
B04 (Switch; c1; ?Is_On)

end block
```

Creation of instances
Separate memories
block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
    NUM [3] as B03,
    AND as B04,

    temp c1 : bool, c2 : nat

B03 (?c2);
B02 (_, Sensor, c2; ?c1);
B04 (Switch; c1; ?Is_On)

end block
block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
    NUM [3] as B03,
    AND as B04,

temp c1 : bool, c2 : nat

B03 (?c2);
B02 (_; Sensor, c2; ?c1);
B04 (Switch; c1; ?Is_On)

end block
block Heater (in Switch : bool; in Sensor : nat; out Is_On : bool) is

allocate Comparator [Strictly_Inferior] as B02,
    NUM [3] as B03,
    AND as B04,

temp c1 : bool, c2 : nat

B03 (?c2);
B02 (_; Sensor; c2; ?c1);
B04 (Switch; c1; ?Is_On)

end block
Blocks: FCS Example

```haskell
block Ail (in spo : bool; out cpo : nat)
{receive lock, up, down :bool; send apo : nat} is
perm pos : nat := 0
if (not (lock) and spo) then
    if up then pos := pos + 1
    elsif down then pos := pos + 1
    end if
    end if;
cpo := pos;
apo := pos
end block
```

Physical interactions
Blocks: FCS Example

block Ail (in spo : bool; out cpo : nat)
{receive lock, up, down : bool; send apo : nat} is

perm pos : nat := 0
if (not (lock) and spo) then
  if up then pos := pos + 1
  elsif down then pos := pos + 1
  end if
end if;
cpo := pos;
apo := pos
end block
Blocks: FCS Example

Memory initialization

```
block Ail (in spo : bool; out cpo : nat)
  {receive lock, up, down :bool; send apo : nat} is
  perm pos : nat := 0
  if (not (lock) and spo) then
    if up then pos := pos + 1
    elsif down then pos := pos + 1
    end if
  end if;
  cpo := pos;
  apo := pos
end block
```
block Ail (in spo : bool; out cpo : nat)

{receive lock, up, down :bool; send apo : nat} is

perm pos : nat := 0

if (not (lock) and spo) then
    if up then pos := pos + 1
    elsif down then pos := pos + 1
    end if
end if;

cpo := pos;
apo := pos

end block
Modeling of asynchronous communication

Activated on demand (passive)
  - Several connected blocks, different instants
  - Nondeterminism

*Signal* statements to control activation
Mediums

- Modeling of asynchronous communication
- Activated on demand (passive)
  - Several connected blocks, different instants
  - Nondeterminism
- *Signal* statements to control activation
Mediums

- Modeling of asynchronous communication
- Activated on demand (passive)
  - Several connected blocks, different instants
  - Nondeterminism
- Signal statements to control activation
Mediums

- Modeling of asynchronous communication
  - Activated on demand (passive)
    - Several connected blocks, different instants
    - Nondeterminism
  - *Signal* statements to control activation
Mediums

- Modeling of asynchronous communication
- Activated on demand (passive)
  - Several connected blocks, different instants
  - Nondeterminism
- *Signal* statements to control activation
Mediums: FCS Example

```plaintext
medium Coord {receive apo : nat | send lock; up; down : bool |
    receive lp, up, dp : bool | send app : nat |
    receive ls, us, ds : bool | send aps : nat} is

perm lock_bu : bool := true, up_bu, down_bu : bool := false, apo_bu : nat := 0
select
  on lp, up, dp -> lock_bu := lp; up_bu := up; down_bu := dp
  on ls, us, ds -> lock_bu := ls; up_bu := us; down_bu := ds
  on apo -> apo_bu := apo
  on ?app -> app := apo_bu
  on ?aps -> aps := apo_bu
  on ?lock, ?up, ?down -> lock := lock_bu; up := up_bu; down := down_bu
end select
end medium
```

Buffers for transited data
medium Coord {receive apo : nat | send lock; up; down : bool |
receive lp, up, dp : bool | send app : nat |
receive ls, us, ds : bool | send aps : nat} is

perm lock_bu : bool := true, up_bu, down_bu : bool := false, apo_bu : nat := 0

select

on lp, up, dp -> lock_bu := lp; up_bu := up; down_bu := dp
‖ on ls, us, ds -> lock_bu := ls; up_bu := us; down_bu := ds
‖ on apo -> apo_bu := apo
‖ on ?app -> app := apo_bu
‖ on ?aps -> aps := apo_bu
‖ on ?lock, ?up, ?down -> lock := lock_bu ; up := up_bu ; down := down_bu

end select

end medium
Mediums: FCS Example

```plaintext
medium Coord {receive apo : nat | send Coord | receive lp, up, dp : bool | send app : nat | receive ls, us, ds : bool | send aps : nat} is

perm lock_bu : bool := true, up_bu, down_bu : bool := false, apo_bu : nat := 0
select
  on lp, up, dp -> lock_bu := lp; up_bu := up; down_bu := dp
  [] on ls, us, ds -> lock_bu := ls; up_bu := us; down_bu := ds
  [] on apo -> apo_bu := apo
  [] on ?app -> app := apo_bu
  [] on ?aps -> aps := apo_bu
  [] on ?lock, ?up, ?down -> lock := lock_bu; up := up_bu; down := down_bu
end select

end medium
```
medium Coord {receive apo : nat | send app : nat |
receive lp, up, dp : bool | send app : nat |
receive ls, us, ds : bool | send aps : nat} is

perm lock_bu : bool := true, up_bu, down_bu : nat := 0

select
on lp, up, dp  -> lock_bu := lp; up_bu := up; down_bu := dp
[] on ls, us, ds  -> lock_bu := ls; up_bu := us; down_bu := ds
[] on apo  -> apo_bu := apo
[] on ?app  -> app := apo_bu
[] on ?aps  -> aps := apo_bu
[] on ?lock, ?up, ?down  -> lock := lock_bu ; up := up_bu ; down := down_bu

end select

end medium
Environments

Modeling of constraints
- Logical constraints between blocks
- Physical constraints

Activated on demand (passive)

Signal statements to control activation
Environments: FCS Example

environment Conc (out p_tok:bool | out s_tok:bool | out alarm:bool) is
perm p_alive, s_alive:bool := true

if p_alive then
  select
    on ?p_tok -> p_tok := true -- primary responds
    [] p_alive := false -- primary fails
  end select
elsif s_alive then
  select
    on ?s_tok -> s_tok := true -- secondary responds
    [] s_alive := false -- secondary fails
  end select
else
  on ?alarm -> alarm := true
end if
end environment

Safety state of Prim and Sec
Environments: FCS Example

environment Conc (out p_tok:bool | out s_tok:bool | out alarm:bool) is

perm p_alive, s_alive:bool := true

if p_alive then
    select
        on ?p_tok -> p_tok := true -- primary responds
        [] p_alive := false -- primary fails
    end select
elseif s_alive then
    select
        on ?s_tok -> s_tok := true -- secondary responds
        [] s_alive := false -- secondary fails
    end select
else
    on ?alarm -> alarm := true
end if
end environment

Safety state of Prim and Sec

Prim has the priority of control
Environments: FCS Example

\[\text{environment Conc} \begin{align*}
\text{perm } p\text{-alive, } s\text{-alive}\text{:= true} \\
\text{if } p\text{-alive then} \\
\quad \text{select} \\
\quad \quad \text{on } ?p\text{-tok }\rightarrow p\text{-tok }:= \text{true} \quad \text{-- primary responds} \\
\quad \quad \quad \quad [] p\text{-alive }:= \text{false} \quad \text{-- primary fails} \\
\text{end select} \\
\text{elsif } s\text{-alive then} \\
\quad \text{select} \\
\quad \quad \text{on } ?s\text{-tok }\rightarrow s\text{-tok }:= \text{true} \quad \text{-- secondary responds} \\
\quad \quad \quad \quad [] s\text{-alive }:= \text{false} \quad \text{-- secondary fails} \\
\text{end select} \\
\text{else} \\
\quad \text{on } ?\text{alarm }\rightarrow \text{alarm }:= \text{true} \\
\text{end if} \\
\text{end environment}\]
Environments: FCS Example

\[
\text{environment Conc (out } p_{\text{tok}}:\text{bool} | \text{out } s_{\text{tok}}:\text{bool} | \text{out } \text{alarm}:\text{bool} ) \text{ is}
\]

\[
\text{perm } p_{\text{alive}}, s_{\text{alive}}:\text{bool} := \text{true}
\]

\[\text{if } p_{\text{alive}} \text{ then}
\]
\[
\text{select}
\]
\[
\text{on } ?p_{\text{tok}} \rightarrow p_{\text{tok}} := \text{true} \quad -- \text{primary responds}
\]
\[
[] p_{\text{alive}} := \text{false} \quad -- \text{primary fails}
\]
\[
\text{end select}
\]

\[\text{elsif } s_{\text{alive}} \text{ then}
\]
\[
\text{select}
\]
\[
\text{on } ?s_{\text{tok}} \rightarrow s_{\text{tok}} := \text{true} \quad -- \text{secondary responds}
\]
\[
[] s_{\text{alive}} := \text{false} \quad -- \text{secondary fails}
\]
\[
\text{end select}
\]

\[\text{else}
\]
\[
\text{on } ?\text{alarm} \rightarrow \text{alarm} := \text{true}
\]
\[
\text{end if}
\]
\[
\text{end environment}
\]
Composition of blocks, mediums, and environments

No direct connection between blocks

Communication between blocks and mediums (resp., environments) by message-passing rendezvous
Systems: FCS Example

system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is

allocate FBWComp as Prim, FBWCom as Sec , Ail as Ail, Alarmer as Alarmer,
  Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord

temp p_tok : bool, p_pos: nat, p_lck, p_up, p_dwn : bool,
  s_tok : bool, s_pos: nat, s_lck, s_up, s_dwn : bool,
  c_pos, pos : nat, lck, up, dwn : bool, safe, ok: bool

network

  Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
  Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
  Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
  Alarmer (safe; ?alarm)

constrainedby

  Conc (?p_tok | ?s_tok | ?safe),
  Ctrl (c_pos | ?ok)

connectedby

  Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}
end system
**Systems: FCS Example**

```plaintext
system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is

allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,

Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord

temp p_tok : bool, p_pos : nat, p_lck, p_up, p_dwn : bool,

s_tok : bool, s_pos : nat, s_lck, s_up, s_dwn : bool,

c_pos, pos : nat, lck, up, dwn : bool, safe, ok : bool

network

Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},

Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},

Ail (ok; ?c_pos) {lck, up, dwn; ?pos},

Alarmer (safe; ?alarm)

constrainedby

Conc (?p_tok | ?s_tok | ?safe),

Ctrl (c_pos | ?ok)

connectedby

Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}

end system
```
Systems: FCS Example

system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is
allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,
Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord

temp p_tok : bool, p_pos : nat, p_lck, p_up, p_dwn : bool,
s_tok : bool, s_pos : nat, s_lck, s_up, s_dwn : bool,
c_pos, pos : nat, lck, up, dwn : bool, safe, ok : bool

network
    Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
    Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
    Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
    Alarmer (safe; ?alarm)

constrainedby
    Conc (?p_tok | ?s_tok | ?safe),
    Ctrl (c_pos | ?ok)

connectedby
    Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}
end system
Systems: FCS Example

system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is

  allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,

  Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord

  temp p_tok : bool, p_pos : nat, p_lck, p_up, p_dwn : bool,

  s_tok : bool, s_pos : nat, s_lck, s_up, s_dwn : bool,

  c_pos, pos : nat, lck, up, dwn : bool, safe, ok : bool

  network

    Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},

    Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},

    Ail (ok; ?c_pos) {lck, up, dwn; ?pos},

    Alarmer (safe; ?alarm)

  constrainedby

    Conc (?p_tok | ?s_tok | ?safe),

    Ctrl (c_pos | ?ok)

  connectedby

    Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}

end system
system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is
allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,
    Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord
temp p_tok : bool, p_pos: nat, p_lck, p_up, p_dwn : bool,
    s_tok : bool, s_pos: nat, s_lck, s_up, s_dwn : bool,
    c_pos, pos : nat, lck, up, dwn : bool, safe, ok: bool

network
    Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
    Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
    Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
    Alarmer (safe; ?alarm)

constrainedby
    Conc (?p_tok | ?s_tok | ?safe),
    Ctrl (c_pos | ?ok)

connectedby
    Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}
Systems: FCS Example

system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is
  allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,
  Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord
  temp p_tok : bool, p_pos: nat, p_lck, p_up, p_dwn : bool,
  s_tok : bool, s_pos: nat, s_lck, s_up, s_dwn : bool,
  c_pos, pos : nat, lck, up, dwn : bool, safe, ok: bool

network
  Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
  Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
  Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
  Alarmer (safe; ?alarm)

constrainedby
  Conc (?p_tok | ?s_tok | ?safe),
  Ctrl (c_pos | ?ok)

connectedby
  Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}
end system
Systems: FCS Example

system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is

allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,

Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord

temp p_tok : bool, p_pos: nat, p_lck, p_up, p_dwn : bool,

s_tok : bool, s_pos: nat, s_lck, s_up, s_dwn : bool,

c_pos, pos : nat, lck, up, dwn : bool, safe, ok: bool

network

Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
Alarmer (safe; ?alarm)

constrainedby

Conc (?,p_tok | ?s_tok | ?safe),
Ctrl (c_pos | ?ok)

c connectedby

Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}

end system
System: FCS Example

system FlightControlSystem ((p_ord, s_ord : nat, alarm : bool) is
allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,
    Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord
temp p_tok : bool, p_pos : nat, p_lck, p_up, p_dwn : bool,
    s_tok : bool, s_pos : nat, s_lck, s_up, s_dwn : bool,
    c_pos, pos : nat, lck, up, dwn : bool, safe, ok : bool

network
    Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
    Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
    Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
    Alarmer (safe; ?alarm)

constrainedby
    Conc (?p_tok | ?s_tok | ?safe),
    Ctrl (c_pos | ?ok)

connectedby
    Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos | s_lck, s_up, s_dwn | ?s_pos}
end system
Systems: FCS Example

```
system FlightControlSystem (p_ord, s_ord : nat, alarm : bool) is
allocate FBWComp as Prim, FBWCom as Sec, Ail as Ail, Alarmer as Alarmer,
    Conc as Conc, Ctrl [10] as Ctrl, Coord as Coord
    temp p_tok : bool, p_pos: nat, p_lck, p_up, p_dwn : bool,
    s_tok : bool, s_pos: nat, s_lck, s_up, s_dwn : bool,
    c_pos, pos : nat, lck, up, dwn : bool, safe, ok: bool

network
    Prim (p_tok; p_ord) {p_pos; ?p_lck, ?p_up, ?p_dwn},
    Sec (s_tok; s_ord) {s_pos; ?s_lck, ?s_up, ?s_dwn},
    Ail (ok; ?c_pos) {lck, up, dwn; ?pos},
    Alarmer (safe; ?alarm)

constrainedby
    Conc (?p_tok | ?s_tok | ?safe),
    Ctrl (c_pos | ?ok)

connectedby
    Coord {pos | ?lck, ?up, ?dwn | p_lck, p_up, p_dwn | ?p_pos, s_lck, s_up, s_dwn | ?s_pos}
end system
```

Visible from the outside world

Invisible from the outside world
Formal Semantics of GRL

- Labelled transition systems
  - States: union of the memories of blocks, mediums, and environments
  - Initial state: initial values of memories
  - Labels: execution of blocks
    + visible inputs/outputs + visible receives/sends
  - Transition function: atomic execution of blocks with connected mediums and environments

145 rules of static semantics [2]
24 rules of structural operational semantics [2]

[2] available in a technical report of 130 pages
Tools for GRL

GRL2LNT (20,000 lines):
- Parser (2,000 lines): lexical and syntactic analysis
- Automated translator to LNT, input language of CADP
- Accurate and concise LNT
- Improve scalability of model checking

Enabled access to CADP
- More than 40 tools
- Explicit state exploration
- Model-checking, equivalence checking, visual checking
Results for the FCS Example

State space generation
- 2,653 states
- 7,406 transitions

Reduction with branching bisimulation
- 5 states
- 1,287 transitions

Formal verification with CADP enabled
Conclusion: GRL

- Versatile and modular description of
  - Synchronous systems
  - Asynchronous communication
  - Environment constraints

- Expressive and general-purpose
- Close to graphical data flow used in industry
- Easier to learn than full-fledged process algebra
- Efficient verification with CADP
Conclusion: ongoing work

- GRL and GRL2LNT applied on an industrial project
  - Crouzet Automation (Schneider Electric)
  - Networks of Programmable Logic Controllers
    ➔ Positive feedback

- Development of off-the-shelf blocks, mediums, and environments

- Automated GRL generation from industrial tools
  ➔ Automated verification chain

- Connection to synchronous verification tools (future work)
Thank You