Modèle et génération automatique de code

Alexandre Chapoutot

ENSTA Paris

2022-2023

Part I

Lecture 4

Compilation of SCADE models

Compilation of SCADE models

The compiler

Compilation of SCADE 6 – 1

Goal: translating parallel and modular language into sequential code.

Recall of the main steps of a compiler:

- Lexical analysis; Grammatical analysis
- Semantic analysis (e.g. typing, etc.)
- Optimization (e.g. constant folding, function in-lining, etc.)
- Code generation

A node is compiled into one function:

```
X = X0
while (1)
  read(input)
  out, X = step (X, input)
  write(out)
done
```

Remark: step function has to be embedded in a OS-dependent loop.
The SCADE compiler can generate code targeted specific real-time OS as VxWorks

Modular compilation

It is a mandatory feature to be assured for

- the traceability of the compilation process
- the low code size generation

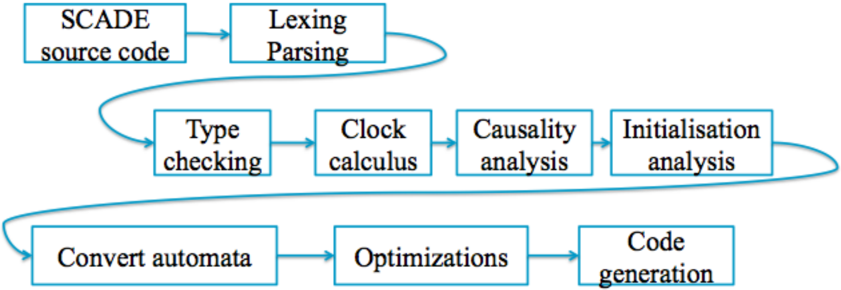
A small problem: it is not possible in the general case

```
node two_copies (a, b: int) returns (x, y: int)
let
   x = a; y = b;
tel;

(x, y) = two_copies(a, x);
```

Solutions:

- in-line all nodes: code size increasing
- a loop between nodes must contain **pre** or **fby** operator

Lustre/SCADE compiler



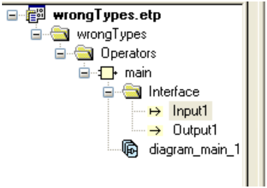
Type checking

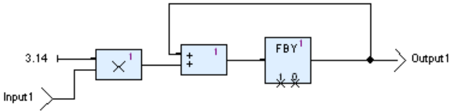
Goal:

- check that all operations are allowed between typed input.
- refuse programs that are not type safe.
- **avoid implicit cast**

```
node cpt (tick: bool) returns (c: int)
var tmp : bool;
let
  c = 0 -> pre c + 3.14;
  tmp = (tick + 1) and c >= tmp;
tel;
```

Example – Bad types

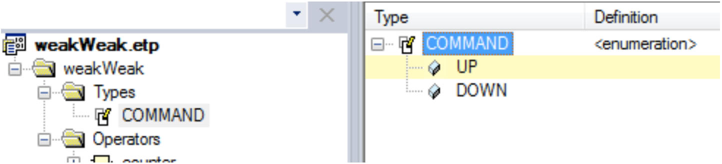




Code Generator			
Information	Log Files	LOGFIL	Log Files
Error	Semantic	ERR_100	main/Output1=: Type mismatch
			This expression has type real but should have type int (output flow mismatch)
Information	Generated Files	GENFIL	KCG- Generated files
Information	Generated Files	GENFIL	Code Generator Generated files

Definition of new types

In SCADE we can define new types, mainly: `arrays`, `enumerations`, `structures`



For example, the process to create `enumeration` type is

- Select project repository in Workspace
- `Insert -> Package Item -> Types`
- Enter the name of the new type; double click on it to get `Type View`
- Select `Definition` tab to choose the category `enumeration`
- Right-clik on the name of the type; choose `Insert -> Definition element` to add new item

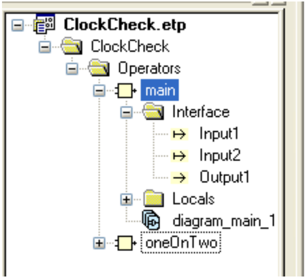
Clock calculus

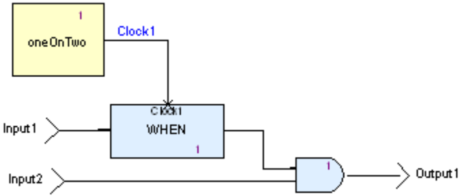
Goal:

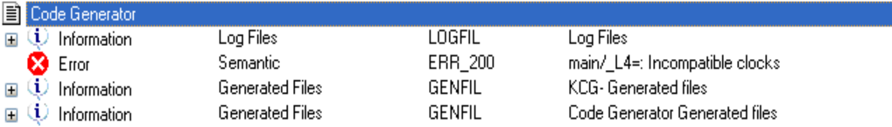
- check that all operations involve elements living on the same clock.
- refuse programs that are not synchronized.
- **guarantee the memory consumption is bounded**

x	x0	x1	x2	x3	x4
y		y0		y1	
x + y	?	?	?	?	?

Example – Clocks







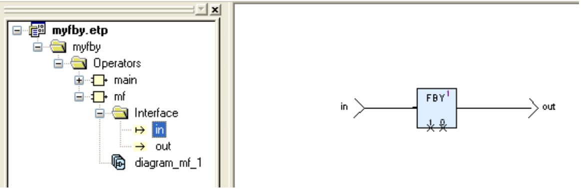
Causality analysis

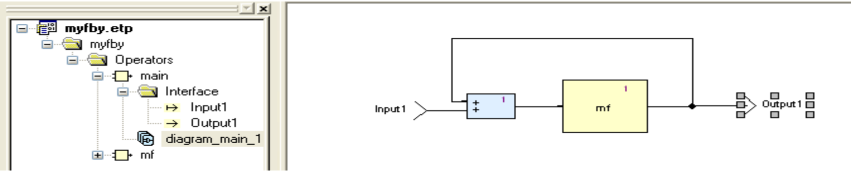
Goal:

- check the absence of causality loop
- **guarantee that the time of execution is bounded**

```
node cpt (tick: bool) returns (c: int)
let
  c = c + 1;
tel;
```

Example – Causality error



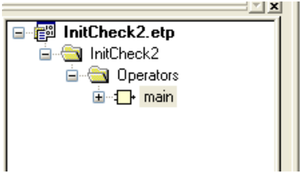


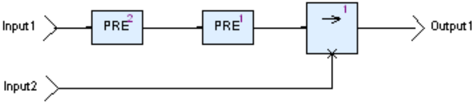
Code Generator			
Information	Log Files	LOGFIL	Log Files
Error	Semantic	ERR_400	main/_L3=: Causality error
Information	Generated Files	GENFIL	KCG- Generated files
Information	Generated Files	GENFIL	Code Generator Generated files

Initialization analysis

Goal:

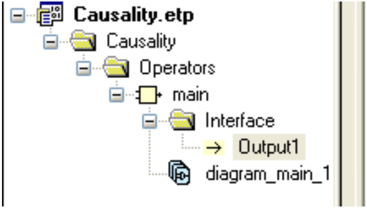
- Check that all memories are initialized
- **guarantee the determinism of the executions**

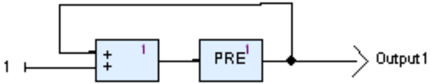




X	X0	X1	X2	X3	X4	…
pre X	nil	X0	X1	X2	X3	…
Y	Y0	Y1	Y2	Y3	Y4	…
Y->X	Y0	X1	X2	X3	X4	…
Y->pre (X)	Y0	X0	X1	X2	X3	…
Y->pre (pre (X))	Y0	nil	X0	X1	X2	…

Example – bad initialization





Code Generator			
Information	Log Files	LOGFIL	Log Files
Error	Semantic	ERR_300	main/_L1=: Initialization error
			main/_L1=: The operator "pre" expects a well-initialized argument
			The pre operator caused a delay
Information	Generated Files	GENFIL	KCG- Generated files
Information	Generated Files	GENFIL	Code Generator Generated files

Intermediate language and C code generation

The code generation is based on an object oriented language:

- very simple class with two methods: step and reset.
- no dynamical allocation, no inheritance.

Translation of each class in a C struct and two functions

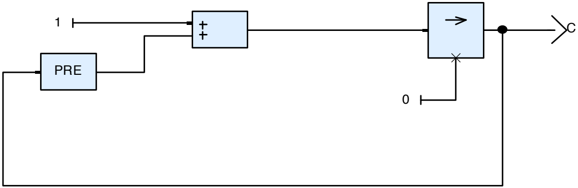
The latest processing

Before generating C code, we have to schedule equations.

Files generated:

```
<root_name>.h, <root_name>.c,
<operator_name>.h, <operator_name>.c,
kcg_types.h, kcg_types.c,
kcg_consts.h, kcg_consts.c, kcg_sensors.h
```

Example – Simple counter




```
/* Compteur.h */

typedef struct {
  kcg_int y;
  kcg_bool init;
} outC_Compteur;
```

Example – Simple counter

```
#include "kcg_consts.h"
#include "kcg_sensors.h"
#include "Compteur.h"

void Compteur_reset(outC_Compteur *outC)
{
  outC->init = kcg_true;
}
```

Example – Simple counter

```
void Compteur(outC_Compteur *outC)
{
  if (outC->init) {
    outC->c = 0;
  }
  else {
    outC->c = 1 + outC->c;
  }
  outC->init = kcg_false;
}
```

Code optimization – Scade 6.4 only

Remark to guarantee traceability model/source code a very limited number of optimization are allowed.

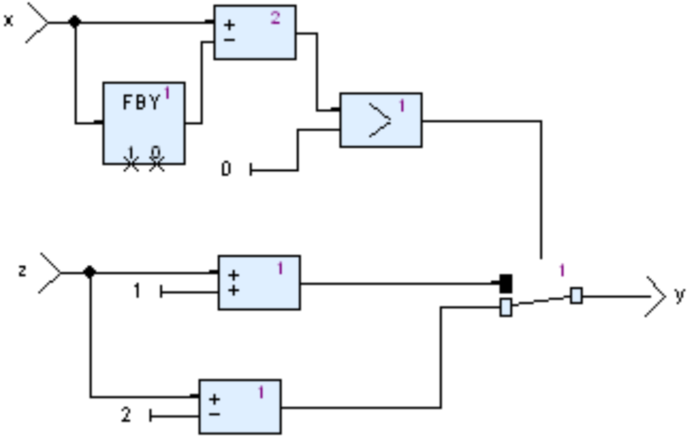
Four levels of optimizations (data-flow part)

- Level 0: no optimization
- Level 1: expressions simplification, equivalent flows simplification, unused variables elimination and expression in-lining
- Level 2 and 3: iterator optimizations

For Level 1 to 3, optimizations on control flow e.g.,

- Reordering control blocks, then merging the content of control structures having the same condition to reduce the number of tests and allow potential additional optimization
- Removing negative conditions in if statements

Example – Switch block



Example – Switch block

Optimization level 0

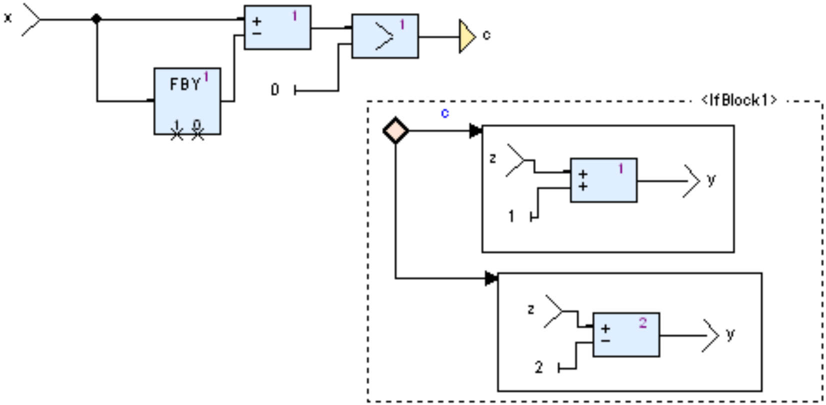
```
if (outC->init) {
   _L10 = 0;
}
else {
   _L10 = outC->_L1;
}
outC->_L1 = inC->x;
_L11 = outC->_L1 - _L10;
_L9 = 2;
_L5 = inC->z;
_L8 = _L5 - _L9;
_L7 = 1;
_L4 = 0;
_L3 = _L11 > _L4;
_L6 = _L5 + _L7;
if (_L3) {
   _L2 = _L6;
}
else {
   _L2 = _L8;
}
outC->y = _L2;
outC->init = kcg_false;
```

Example – Switch block

Optimization level 1

```
void cond(inC_cond *inC, outC_cond *outC)
{
  kcg_int tmp;

  if (outC->init) {
    tmp = 0;
  }
  else {
    tmp = outC->rem_x;
  }
  if (inC->x - tmp > 0) {
    outC->y = inC->z + 1;
  }
  else {
    outC->y = inC->z - 2;
  }
  outC->rem_x = inC->x;
  outC->init = kcg_false;
}
```

Example – If block



Example – If block

Optimization level 0

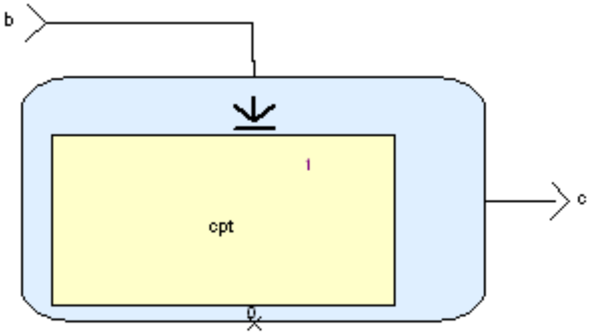
```
if (outC->init) {
  _L4 = 0;
}
else {
  _L4 = outC->_L5;
}
outC->_L5 = inC->x;
_L1 = outC->_L5 - _L4;
_L3 = 0;
_L2 = _L1 > _L3;
c = _L2;
IfBlock1_clock = c;
if (IfBlock1_clock) {
  _L1_IfBlock1 = inC->z;
  _L2_IfBlock1 = 1;
  _L3_IfBlock1 = _L1_IfBlock1 + _L2_IfBlock1;
  y1 = _L3_IfBlock1;
  outC->y = y1;
}
else {
  _L14_IfBlock1 = inC->z;
  _L23_IfBlock1 = 2;
  _L32_IfBlock1 = _L14_IfBlock1 - _L23_IfBlock1;
  y = _L32_IfBlock1;
  outC->y = y;
}
outC->init = kcg_false;
```

Example – If block

Optimization level 1

```
void cond2(inC_cond2 *inC, outC_cond2 *outC)
{
  kcg_int tmp;
  /* cond2::c */ kcg_bool c;

  if (outC->init) {
    tmp = 0;
  }
  else {
    tmp = outC->rem_x;
  }
  c = inC->x - tmp > 0;
  if (c) {
    outC->y = inC->z + 1;
  }
  else {
    outC->y = inC->z - 2;
  }
  outC->rem_x = inC->x;
  outC->init = kcg_false;
}
```

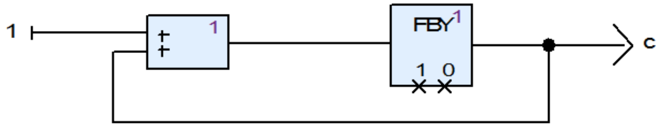
Example – Activate block

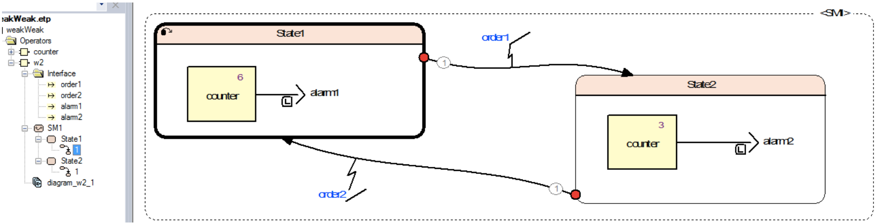


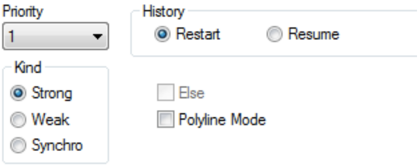
Example – Activate block

```
void activ_reset(outC_activ *outC)
{
  outC->init = kcg_true;
  /* 1 */ Compteur_reset(&outC->Context_1);
}

/* activ */
void activ(inC_activ *inC, outC_activ *outC)
{
  if (inC->b) {
    /* 1 */ Compteur(&outC->Context_1);
    outC->c = outC->Context_1.c;
  }
  else if (outC->init) {
    outC->c = 0;
  }
  outC->init = kcg_false;
}
```

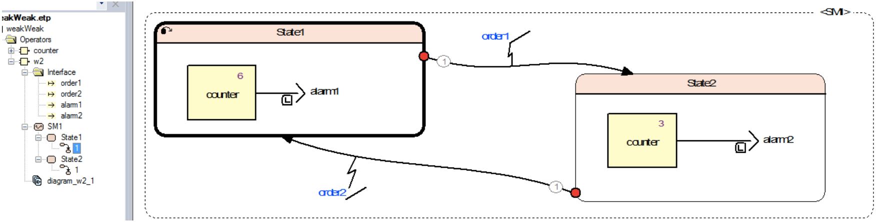
State Machine – Strong transitions – 1







State Machine – Strong transitions – 2



Semantics: the transition is evaluated at the begin of each cycle. If true then evaluate body of the new activiated state ortherwise evaluate the body of the current activated state.

Note by default entering a new state reset all the memory used in this state (`Restart` behavior)
To keep the value of the memory between each activation use `Resume` behavior

State machine – Strong transition .h

```
#include "kcg_types.h"
#include "counter.h"

/* ========================  input structure  ====================== */
typedef struct {
  kcg_bool /* order1/ */ order1;
  kcg_bool /* order2/ */ order2;
} inC_w2;

/* =====================  no output structure  ====================== */

/* ========================  context type  ========================= */
typedef struct {
  /* ---------------------------  outputs  ------------------------- */
  kcg_int32 /* alarm1/ */ alarm1;
  kcg_int32 /* alarm2/ */ alarm2;
  /* -----------------------  no local probes  --------------------- */
  /* ----------------------- local memories  ------------------------- */
  SSM_ST_SM1 /* SM1: */ SM1_state_nxt;
  /* ---------------------  sub nodes' contexts  -------------------- */
  outC_counter /* SM1:State2:_L3=(counter#3)/ */ Context_counter_3;
  outC_counter /* SM1:State1:_L8=(counter#6)/ */ Context_counter_6;
  /* ---------------- no clocks of observable data ------------------ */
} outC_w2;

/* ===========  node initialization and cycle functions  =========== */
/* w2/ */
extern void w2(inC_w2 *inC, outC_w2 *outC);
```

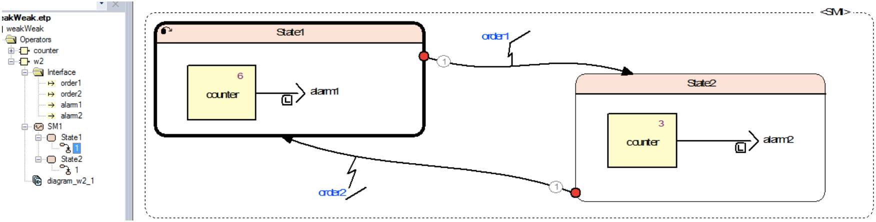
State machine – Strong transition .c – 1

```c
#include "kcg_consts.h"
#include "kcg_sensors.h"
#include "w2.h"

/* w2/ */
void w2(inC_w2 *inC, outC_w2 *outC)
{
  /* SM1: */
  SSM_ST_SM1 SM1_state_act;
  /* SM1: */
  kcg_bool SM1_reset_act;

  /* SM1: */
  switch (outC->SM1_state_nxt) {
    case SSM_st_State2_SM1 :
      if (inC->order2) {
        SM1_state_act = SSM_st_State1_SM1;
      }
      else {
        SM1_state_act = SSM_st_State2_SM1;
      }
      SM1_reset_act = inC->order2;
      break;
    case SSM_st_State1_SM1 :
      if (inC->order1) {
        SM1_state_act = SSM_st_State2_SM1;
      }
      else {
        SM1_state_act = SSM_st_State1_SM1;
      }
      SM1_reset_act = inC->order1;
      break;
    default :
      /* this default branch is unreachable */
      break;
  }
```

State machine – Strong transition .c – 2

```c
/* SM1: */
switch (SM1_state_act) {
  case SSM_st_State2_SM1 :
    if (SM1_reset_act) {
      /* SM1:State2:_L3=(counter#3)/ */ counter_reset(&outC->Context_counter_3);
    }
    /* SM1:State2:_L3=(counter#3)/ */ counter(&outC->Context_counter_3);
    outC->alarm2 = outC->Context_counter_3.c;
    outC->SM1_state_nxt = SSM_st_State2_SM1;
    break;
  case SSM_st_State1_SM1 :
    if (SM1_reset_act) {
      /* SM1:State1:_L8=(counter#6)/ */ counter_reset(&outC->Context_counter_6);
    }
    /* SM1:State1:_L8=(counter#6)/ */ counter(&outC->Context_counter_6);
    outC->alarm1 = outC->Context_counter_6.c;
    outC->SM1_state_nxt = SSM_st_State1_SM1;
    break;
  default :
    /* this default branch is unreachable */
    break;
  }
}
```

State Machine – Weak transitions



Semantics: the transition is evaluated at the end of each cycle after evaluating the body of current active state. If true then activiate the target state at the next cycle ortherwise do nothing.

State machine – Weak transition .c – 1

```
#include "kcg_consts.h"
#include "kcg_sensors.h"
#include "w2.h"

/* w2/ */
void w2(inC_w2 *inC, outC_w2 *outC)
{
  /* SM1: */
  SSM_ST_SM1 SM1_state_sel;

  SM1_state_sel = outC->SM1_state_nxt;
  /* SM1: */
  switch (SM1_state_sel) {
    case SSM_st_State2_SM1 :
      /* SM1:State2:_L3=(counter#3)/ */ counter(&outC->Context_counter_3);
      outC->alarm2 = outC->Context_counter_3.c;
      if (inC->order2) {
        outC->SM1_state_nxt = SSM_st_State1_SM1;
      }
      else {
        outC->SM1_state_nxt = SSM_st_State2_SM1;
      }
      break;
    case SSM_st_State1_SM1 :
      /* SM1:State1:_L8=(counter#6)/ */ counter(&outC->Context_counter_6);
      outC->alarm1 = outC->Context_counter_6.c;
      if (inC->order1) {
        outC->SM1_state_nxt = SSM_st_State2_SM1;
      }
      else {
        outC->SM1_state_nxt = SSM_st_State1_SM1;
      }
      break;
    default :
      /* this default branch is unreachable */
      break;
  }
```