A CHR-Based Solver for Weak Memory Behaviors

CSTVA 2016 — **Allan Blanchard**[1,2] Nikolai Kosmatov[1] Frédéric Loulergue[2]

[1] CEA LIST - Software Reliability Laboratory
[2] Univ Orléans, INSA Centre Val de Loire, LIFO

Memory model

Reasoning about concurrency

Memory models define :

- how threads interact with memory
- in particular on shared data

They can be defined by :

- processor architectures
- languages

Sequential Consistency

Lamport 1979

The semantics of the parallel composition of two programs is given by the interleavings of the executions of both programs.

A simple program

$$\mathtt{x} := \Omega;\ \mathtt{y} := \Omega$$

Thread 0	Thread 1
$\mathtt{x} := 1;$	$\mathtt{y} := 1;$
$\mathtt{r}_0 := \mathtt{y};$	$\mathtt{r}_1 := \mathtt{x};$

Possible results :

- $\mathtt{r}_0 = 1 \wedge \mathtt{r}_1 = 1$
- $\mathtt{r}_0 = \Omega \wedge \mathtt{r}_1 = 1$
- $\mathtt{r}_0 = 1 \wedge \mathtt{r}_1 = \Omega$

Impossible result :

- $\mathtt{r}_0 = \Omega \wedge \mathtt{r}_1 = \Omega$

Weak behaviors

Modern architectures

For example, x86-TSO or ARM do not respect SC :

- hard to ensure such synchronization in hardware,
- and far too costly.

Examples of weak behaviors

- out-of-order execution,
- store buffering,
- speculative execution,
- ...

Example with store buffering

Thread 0	Thread 1
$\mathtt{x} := 1;$	$\mathtt{y} := 1;$
$\mathtt{r}_0 := \mathtt{y};$	$\mathtt{r}_1 := \mathtt{x};$
$r_0 =$	$r_1 =$

Proc 0 : Store Buffer
…

Proc 1 : Store Buffer
…

Global Memory

…	$\mathtt{x} = \Omega$	$\mathtt{y} = \Omega$	…

Example with store buffering

Thread 0	Thread 1
$\longrightarrow$ $\mathbf{x} := \mathbf{1};$	$y := 1;$
$r_0 := y;$	$r_1 := x;$
$r_0 =$	$r_1 =$

Proc 0 : Store Buffer		
…	**$\mathbf{x} = \mathbf{1}$**	…

Proc 1 : Store Buffer
…

Global Memory

…	$x = \Omega$	$y = \Omega$	…

Example with store buffering

Thread 0	Thread 1
$x := 1;$	$\longrightarrow$ $\mathbf{y} := \mathbf{1};$
$r_0 := y;$	$r_1 := x;$
$r_0 =$	$r_1 =$

Proc 0 : Store Buffer		
…	$x = 1$	…

Proc 1 : Store Buffer		
…	$\mathbf{y = 1}$	…

Global Memory

…	$x = \Omega$	$y = \Omega$	…

Example with store buffering

Thread 0	Thread 1
$x := 1;$	$y := 1;$
$\longrightarrow \mathbf{r_0} := \mathbf{y};$	$r_1 := x;$
$\mathbf{r_0} = \boldsymbol{\Omega}$	$r_1 =$

Proc 0 : Store Buffer		
...	$x = 1$	...

Proc 1 : Store Buffer		
...	$y = 1$	...

Global Memory

...	$x = \Omega$	$\mathbf{y} = \boldsymbol{\Omega}$	...

Example with store buffering

Thread 0	Thread 1
$x := 1;$	$y := 1;$
$r_0 := y;$	$\longrightarrow \mathbf{r_1} := \mathbf{x};$
$r_0 = \Omega$	$\mathbf{r_1} = \boldsymbol{\Omega}$

Proc 0 : Store Buffer		
...	$x = 1$	...

Proc 1 : Store Buffer		
...	$y = 1$	...

Global Memory

...	$\mathbf{x} = \boldsymbol{\Omega}$	$y = \Omega$	...

Example with store buffering

Thread 0	Thread 1
$x := 1;$	$y := 1;$
$r_0 := y;$	$r_1 := x;$
$r_0 = \Omega$	$r_1 = \Omega$

Proc 0 : Store Buffer		
…	$x = 1$	…

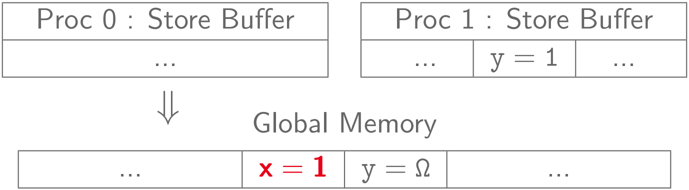
Proc 1 : Store Buffer		
…	$y = 1$	…

Global Memory

…	$x = \Omega$	$y = \Omega$	…

Example with store buffering

Thread 0	Thread 1
x := 1;	y := 1;
r_0 := y;	r_1 := x;
$r_0 = \Omega$	$r_1 = \Omega$



Proc 0 : Store Buffer
…

Proc 1 : Store Buffer		
…	$y = 1$	…

⇓

Global Memory

…	$\mathbf{x = 1}$	$y = \Omega$	…

Example with store buffering

Thread 0	Thread 1
$\mathtt{x} := 1;$	$\mathtt{y} := 1;$
$\mathtt{r}_0 := \mathtt{y};$	$\mathtt{r}_1 := \mathtt{x};$
$r_0 = \Omega$	$r_1 = \Omega$

Proc 0 : Store Buffer
…

Proc 1 : Store Buffer
…

⇓

Global Memory

…	$\mathtt{x} = 1$	**$\mathtt{y} = 1$**	…

Example with store buffering

Thread 0	Thread 1
$x := 1;$	$y := 1;$
$r_0 := y;$	$r_1 := x;$
$\mathbf{r_0 = \Omega}$	$\mathbf{r_1 = \Omega}$

Proc 0 : Store Buffer
...

Proc 1 : Store Buffer
...

Global Memory

...	$\mathbf{x = 1}$	$\mathbf{y = 1}$	...

This behavior is allowed on ARM and TSO processors

We need to understand weak behaviors

Reasoning about programs

We have more and more multi-core software but

- it is hard to reason about them
- most analysis techniques are not aware of weak behaviors

Existing dedicated tools

CPPMem (Batty et al. 2010)

Program executions under C++11 model

Herding cats (Alglave et al. 2014)

Generic framework for weak behaviors

- written in OCaml
- provides a language to specify memory models

JMMSolve (Schrijvers 2004) based on CCMM (Saraswat 2004)

Program executions under Java Memory Model

- based on Concurrent Constraint-based Memory Machines
- written using Constraint Handling Rules (CHR)

Prolog and CHR

Prolog

Declarative language for logic programming

Constraint Handling Rules

Declarative language for constraint programming

- maintains a store of constraints ($\sim$ terms)
- handled by rules that will add or remove constraints

Goals of our solver

To identify allowed executions

- for a given parallel program
- according to a given memory model

Additional goals

- possibility to add new memory models
- support of specific instructions

Basic relations

(st, x, Ω)

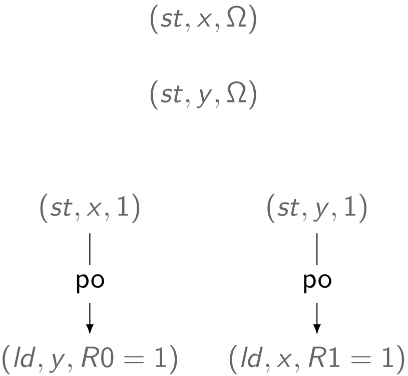
(st, y, Ω)

$(st, x, 1)$ $(st, y, 1)$

- Program-Order : PO
- Coherency-Order : CO
- Reads-From : RF

$(ld, y, R0 = 1)$ $(ld, x, R1 = 1)$

Basic relations



(st, x, Ω)

(st, y, Ω)

$(st, x, 1)$

po

$(ld, y, R0 = 1)$

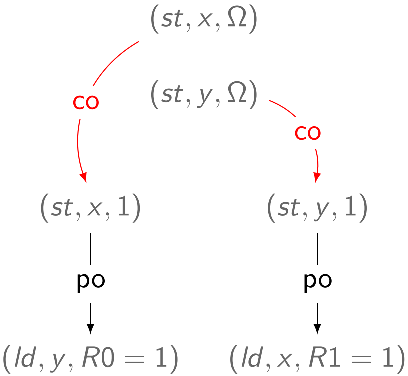
$(st, y, 1)$

po

$(ld, x, R1 = 1)$

- Program-Order : PO
- Coherency-Order : CO
- Reads-From : RF

Basic relations



(st, x, Ω)

(st, y, Ω)

co

co

$(st, x, 1)$

$(st, y, 1)$

po

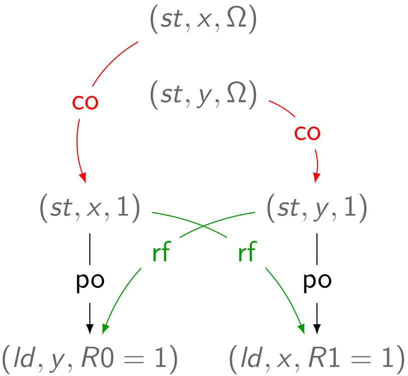
po

$(ld, y, R0 = 1)$

$(ld, x, R1 = 1)$

- Program-Order : PO
- Coherency-Order : CO
- Reads-From : RF

Basic relations



(st, x, Ω)

(st, y, Ω)

co

co

$(st, x, 1)$

$(st, y, 1)$

po

rf

rf

po

$(ld, y, R0 = 1)$

$(ld, x, R1 = 1)$

- Program-Order : PO
- Coherency-Order : CO
- Reads-From : RF

Chosen approach

Generate all candidate executions

An execution is represented by ordering relations :

- CO: for each location *l*, a total ordering of every store to *l*
- RF: for each load, a store having written the value being read
- we combine all permutations of CO and RF using backtracking

Filter out forbidden executions

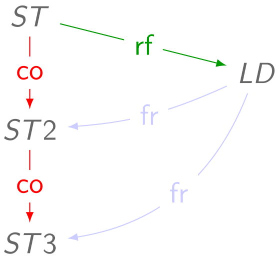
- apply model rules to deduce more ordering relations
- incoherent execution: an action must happen before itself (which means that some relations exhibits a cycle)

Express and derive relations with CHR

Relation between 2 instructions	CHR constraint
$(st, x, \Omega) \xrightarrow{CO} (st, x, 1)$	$\mathtt{co}((\mathtt{st}, \mathtt{x}, \mathtt{undefined}), (\mathtt{st}, \mathtt{x}, 1))$
$(st, x, 2) \xrightarrow{RF} (ld, x, R)$	$\mathtt{rf}((\mathtt{st}, \mathtt{x}, 2), (\mathtt{ld}, \mathtt{x}, 2))$

CHR rules to derive new relations :

- $\mathtt{rf}(\mathtt{ST}, \mathtt{LD}), \mathtt{co}(\mathtt{ST}, \mathtt{ST2}) \Rightarrow \mathtt{fr}(\mathtt{LD}, \mathtt{ST2}).$
- $\mathtt{fr}(\mathtt{LD}, \mathtt{ST2}), \mathtt{co}(\mathtt{ST2}, \mathtt{ST3}) \Rightarrow \mathtt{fr}(\mathtt{LD}, \mathtt{ST3}).$

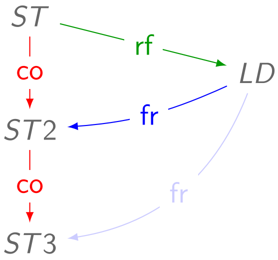


Express and derive relations with CHR

Relation between 2 instructions	CHR constraint
$(st, x, \Omega) \xrightarrow{CO} (st, x, 1)$	$\texttt{co}((\texttt{st}, \texttt{x}, \texttt{undefined}), (\texttt{st}, \texttt{x}, 1))$
$(st, x, 2) \xrightarrow{RF} (ld, x, R)$	$\texttt{rf}((\texttt{st}, \texttt{x}, 2), (\texttt{ld}, \texttt{x}, 2))$

CHR rules to derive new relations :

- $\texttt{rf}(\texttt{ST}, \texttt{LD}), \texttt{co}(\texttt{ST}, \texttt{ST2}) \Rightarrow \texttt{fr}(\texttt{LD}, \texttt{ST2}).$
- $\texttt{fr}(\texttt{LD}, \texttt{ST2}), \texttt{co}(\texttt{ST2}, \texttt{ST3}) \Rightarrow \texttt{fr}(\texttt{LD}, \texttt{ST3}).$

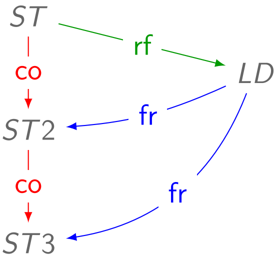


Express and derive relations with CHR

Relation between 2 instructions	CHR constraint
$(st, x, \Omega) \xrightarrow{CO} (st, x, 1)$	$\texttt{co((st, x, undefined), (st, x, 1))}$
$(st, x, 2) \xrightarrow{RF} (ld, x, R)$	$\texttt{rf((st, x, 2), (ld, x, 2))}$

CHR rules to derive new relations :

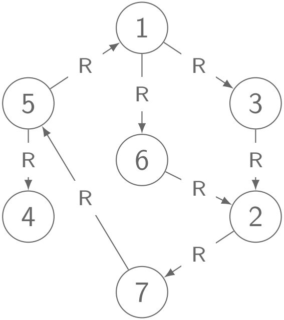
- $\texttt{rf(ST, LD), co(ST, ST2)} \Rightarrow \texttt{fr(LD, ST2).}$
- $\texttt{fr(LD, ST2), co(ST2, ST3)} \Rightarrow \texttt{fr(LD, ST3).}$



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

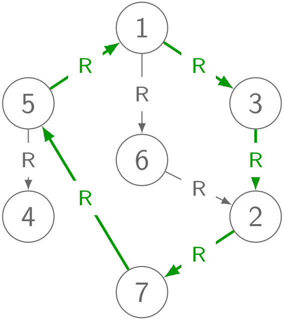
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
   true.

tc(B,E), r(E,B) <=>
   cycle(B).

tc(B,E), r(E,N) ==>
   inf(B,N) | tc(B,N).

r(I,J) ==>
   inf(I,J) | tc(I,J).
```



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

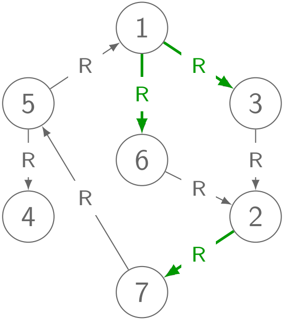
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

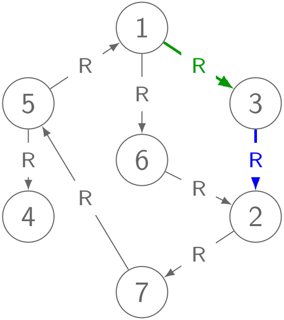
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

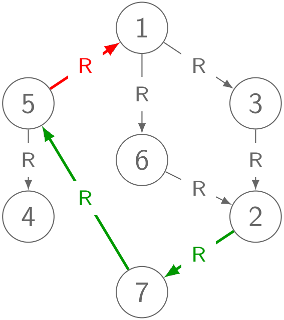
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

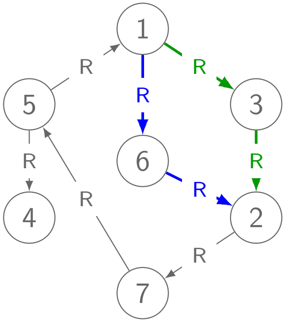
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



Detection of incoherent execution

Reminder : incoherent execution = cycle in the perserved relations

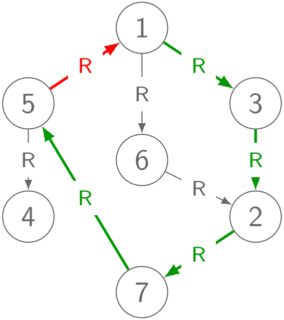
```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



Detection of incoherent execution

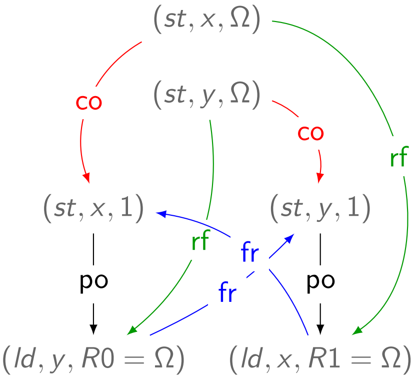
Reminder : incoherent execution = cycle in the perserved relations

```
:- chr_constraint r/2, tc/2, cycle/1.

tc(B,E) \ tc(B,E) <=>
  true.

tc(B,E), r(E,B) <=>
  cycle(B).

tc(B,E), r(E,N) ==>
  inf(B,N) | tc(B,N).

r(I,J) ==>
  inf(I,J) | tc(I,J).
```



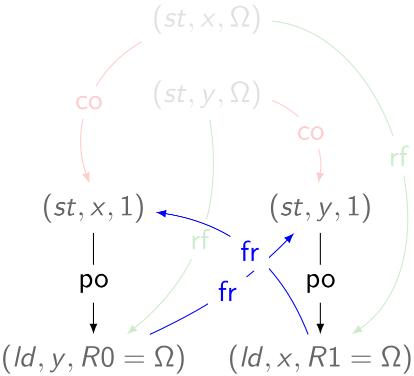
Definition of a model : SC

```
co(I,J) ==> sc(I,J).
rf(I,J) ==> sc(I,J).
fr(I,J) ==> sc(I,J).
po(I,J) ==> sc(I,J).
```



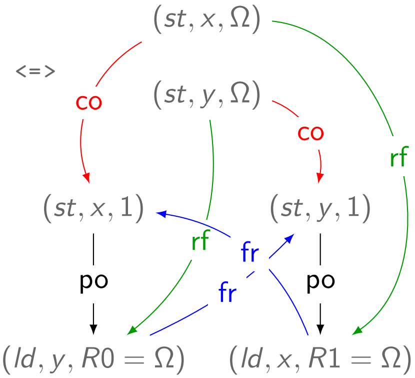
Definition of a model : SC

```
co(I,J) ==> sc(I,J).
rf(I,J) ==> sc(I,J).
fr(I,J) ==> sc(I,J).
po(I,J) ==> sc(I,J).
```



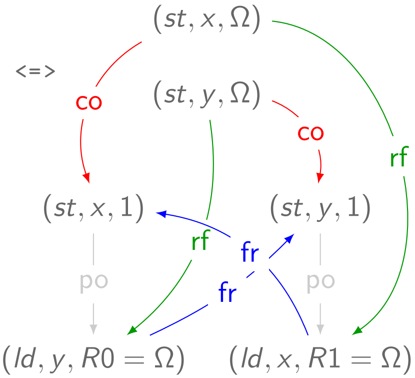
Definition of a model: TSO (simplified)

```
ppo((st,_,_), (ld,_,_)) <=>
    true.
ipo(I,J) ==> ppo(I,J).

bar(I,J) ==> tso(I,J).
ppo(I,J) ==> tso(I,J).
rf(I,J) ==> tso(I,J).
co(I,J) ==> tso(I,J).
fr(I,J) ==> tso(I,J).
```



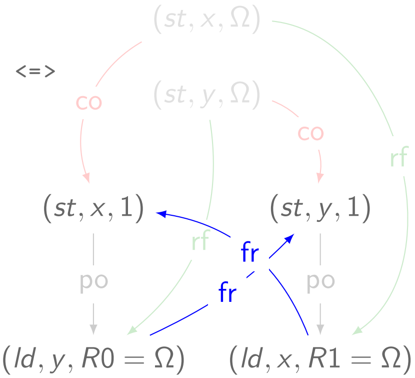
Definition of a model: TSO (simplified)

```
ppo((st,_,_), (ld,_,_)) <=>
    true.
ipo(I,J) ==> ppo(I,J).

bar(I,J) ==> tso(I,J).
ppo(I,J) ==> tso(I,J).
rf(I,J) ==> tso(I,J).
co(I,J) ==> tso(I,J).
fr(I,J) ==> tso(I,J).
```



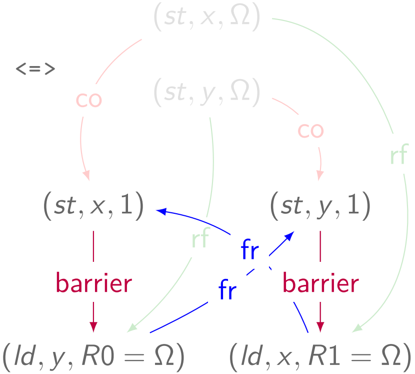
Definition of a model: TSO (simplified)

```
ppo((st,_,_), (ld,_,_)) <=>
    true.
ipo(I,J) ==> ppo(I,J).

bar(I,J) ==> tso(I,J).
ppo(I,J) ==> tso(I,J).
rf(I,J) ==> tso(I,J).
co(I,J) ==> tso(I,J).
fr(I,J) ==> tso(I,J).
```



Definition of a model: TSO (simplified)

```
ppo((st,_,_), (ld,_,_)) <=>
    true.
ipo(I,J) ==> ppo(I,J).

bar(I,J) ==> tso(I,J).
ppo(I,J) ==> tso(I,J).
rf(I,J) ==> tso(I,J).
co(I,J) ==> tso(I,J).
fr(I,J) ==> tso(I,J).
```



Implemented Memory Models

- SC
- TSO
- PSO
- ARM (Almost finished)

Correctness tests

Program samples from Herd

We tested the implementation on 18 examples of litmus programs.

- message passing,
- message passing by address passing,
- basic uniproc relations,
- store buffering,
- ...

We observe the same results found by Herd.

`http://virginia.cs.ucl.ac.uk/herd/`

Performance tests

Multiple message passing

Example	Model	#exec	Herd	CHR Solver
MP3	Generic	147 436	1.2s	3.3s
	PSO	2 258	3.8s	6.4s
	TSO	800	4.1s	3.2s
	SC	678	5.5s	3.3s
MP4	Generic	255 000 000	1405s	$>$ 1h
	PSO	516 030	$>$ 1h	2796s
	TSO	96 498	$>$ 1h	752s
	SC	81 882	$>$ 1h	747s

Early pruning makes the solver efficient for “long” programs.

Conclusion

A CHR based solver for weak memory behaviors

- currently for SC, TSO and PSO
- easy to use and to extend for new memory models
- it was not so hard to implement
- despite the (awful) complexity, execution time is not so bad

Future work

- Features :
  - finalize ARM
  - add read-modify-write operations and branching
- Evaluate on further benchmarks

Thank you ! Questions ?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA Tech List
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
`www-list.cea.fr`

Etablissement public à caractère industriel et commercial — RCS Paris B 775 685 019