Adding CSPm Functions and Data Types to CSP++

Daniel GARNER, Markus ROGGENBACH,
Bill GARDNER

CPA 2015

DITTO

Motivation: Fault-tolerant computer of the ISS

1. Protocol verified by Lamport (1980ties)
2. Implementation in Occam (1990ties)
3. Verification of Occam programs by abstraction to CSP (1990ties)

Buth et al. report on their verification:

- “seven deadlock situations were uncovered”
- “about five livelocks were detected”

CSP++ methodology

- gain an understanding of the system
- specify & analyse communication structure in CSP
- fully automatic translation to C++
- enrich the system with user coded functions

Overview

A puzzle

Modelling and Verification using CSP

Code generation with CSP++

(Sorry, no user coded functions)

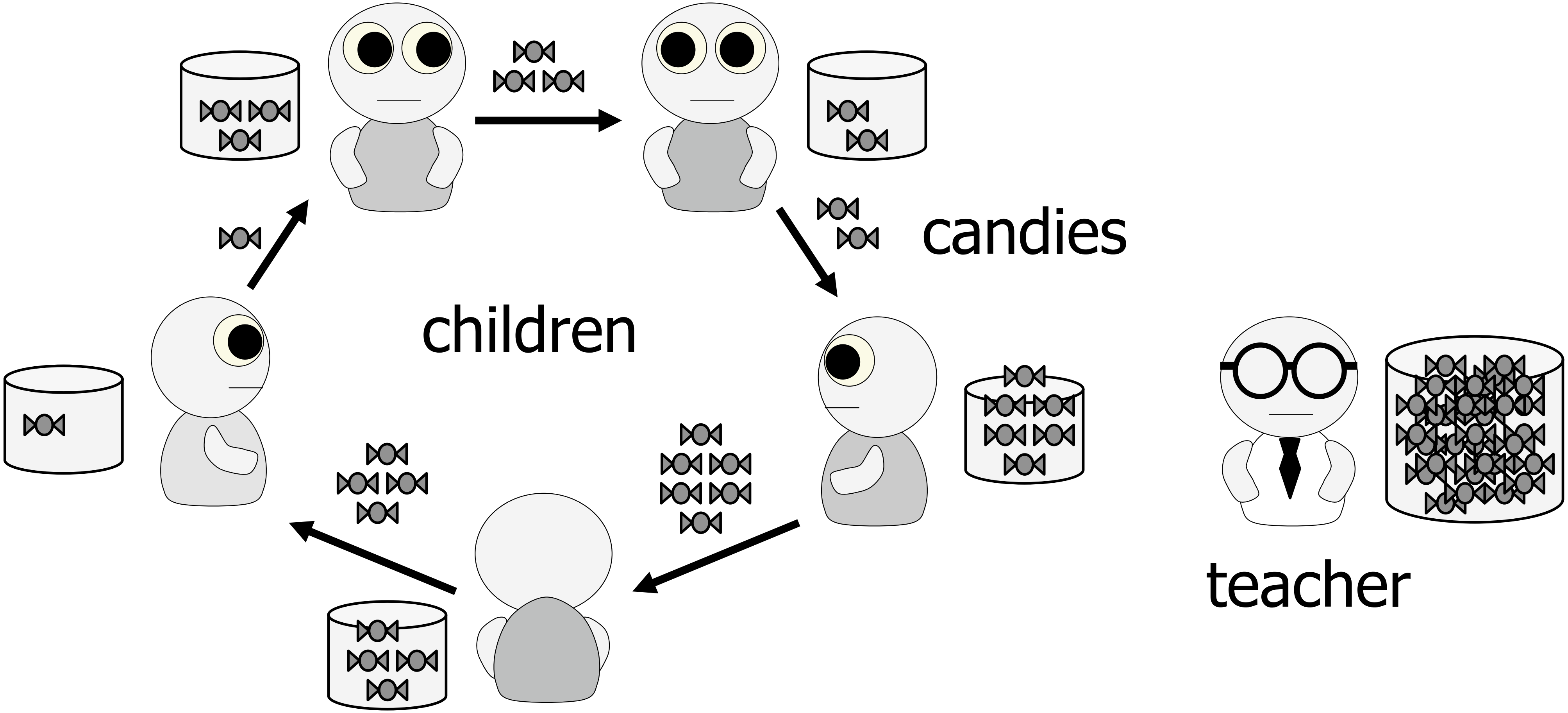
A mathematical puzzle

The children & candy puzzle

There are n children sitting in a circle,
each with an even number of candies.

The following two steps are repeated indefinitely:

- every child passes half of their candies to the child on their left;
- any child who ends up with an odd number of candies is given another candy by the teacher.


candies
children
teacher

Some natural questions on the system

- Will the teacher keep handing out more and more candies?
- Will an unequal distribution of candies eventually become an equal one?

With some mathematical analysis one can establish:

- The maximum number of candies held by a single child never increases.

  Consequence:
  The teacher must eventually stop handing out candies.

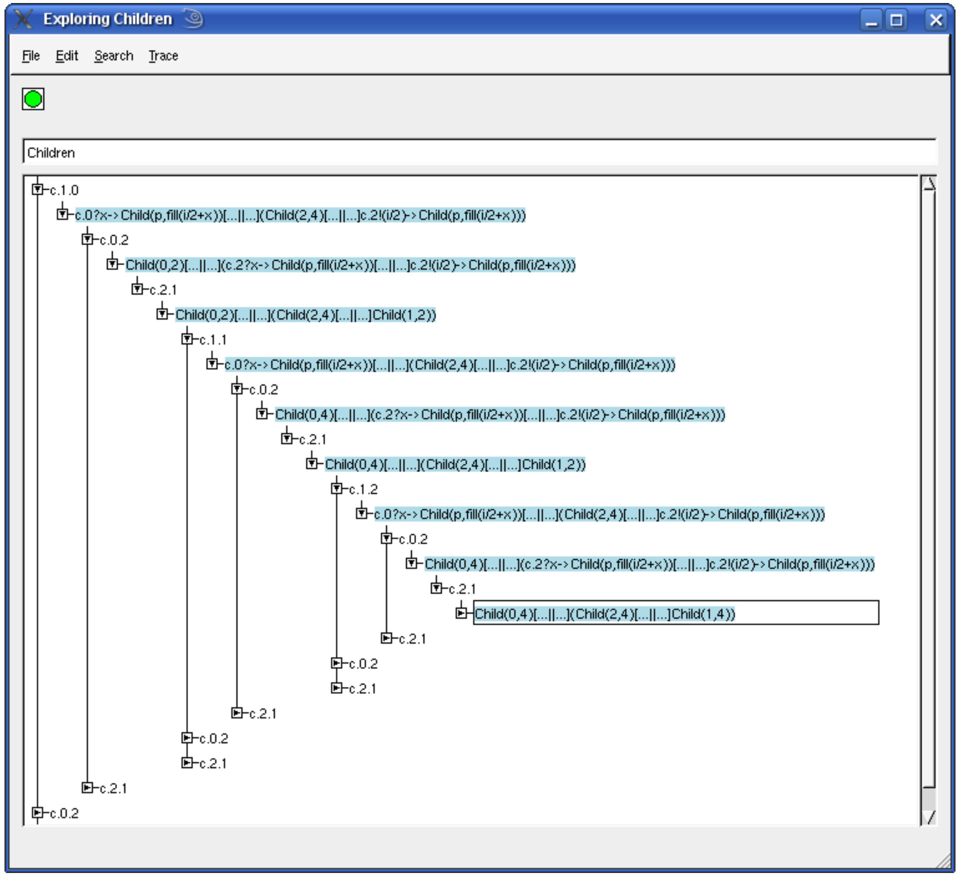
- Eventually,
  all children will hold the same number of candies.

Modelling, Simulation, Model-Checking, Theorem-Proving

Asynchronous model of the puzzle in CSP

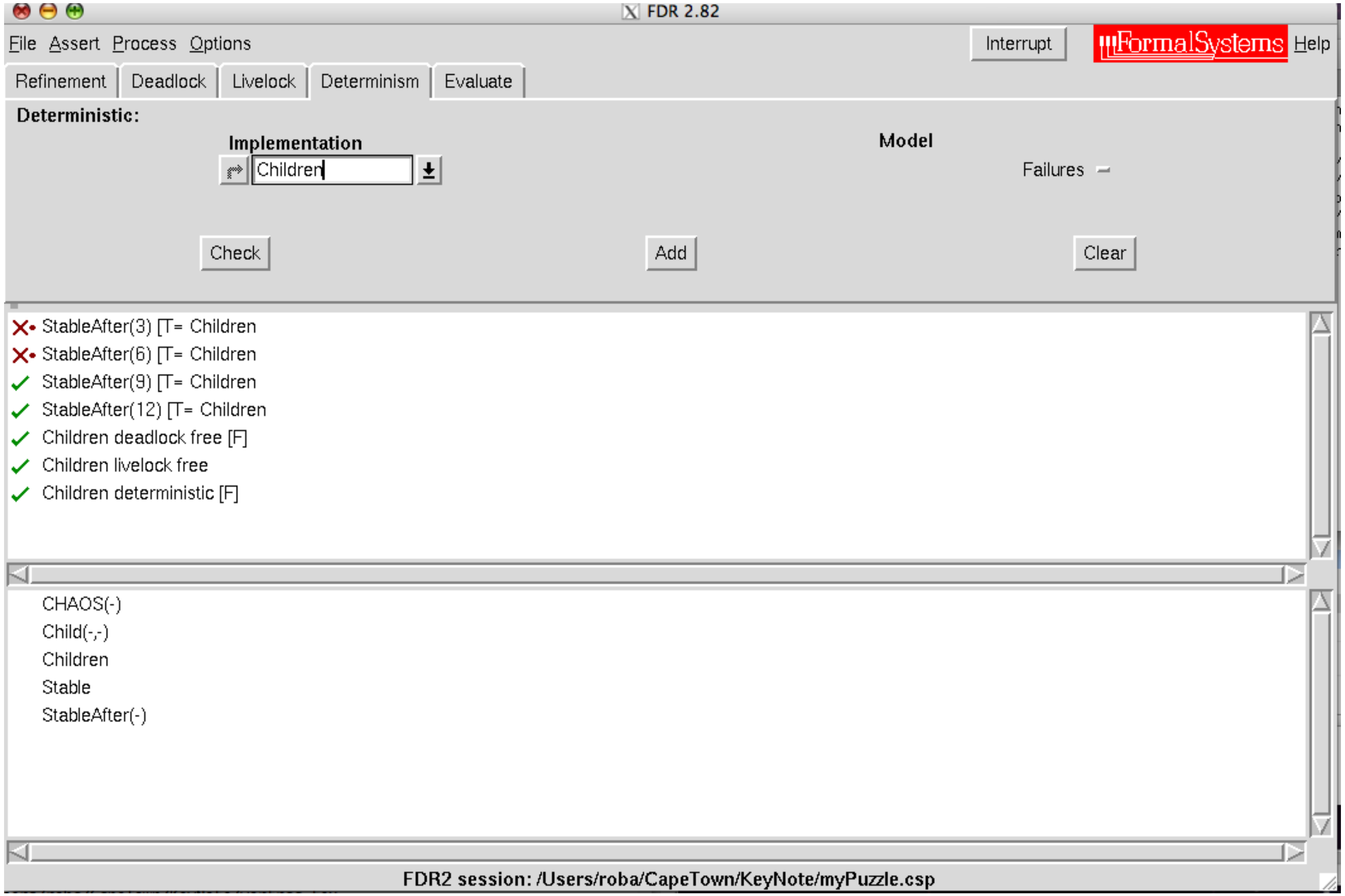
```
channel c : {0..2}.{0..4}
channel d : {0..2}.{0..4}

pragma cspt function
leftof(i) = (i+1)%3

pragma cspt function
fill(n) = if (n % 2 == 0) then n else n + 1

Child(i,x) =
   c.leftof(i)!x/2 -> d.leftof(i).x/2 -> c.i?y -> Child(i,fill((x/2) + y))
   []
   c.i?y -> c.leftof(i)!x/2 -> d.leftof(i).x/2 -> Child(i,fill((x/2) + y))

SYS = (Child(0,0) [|{|c.1|}|] Child(1,2)) [|{|c.0,c.2|}|] Child(2,4)
```

Simulation with ProBE



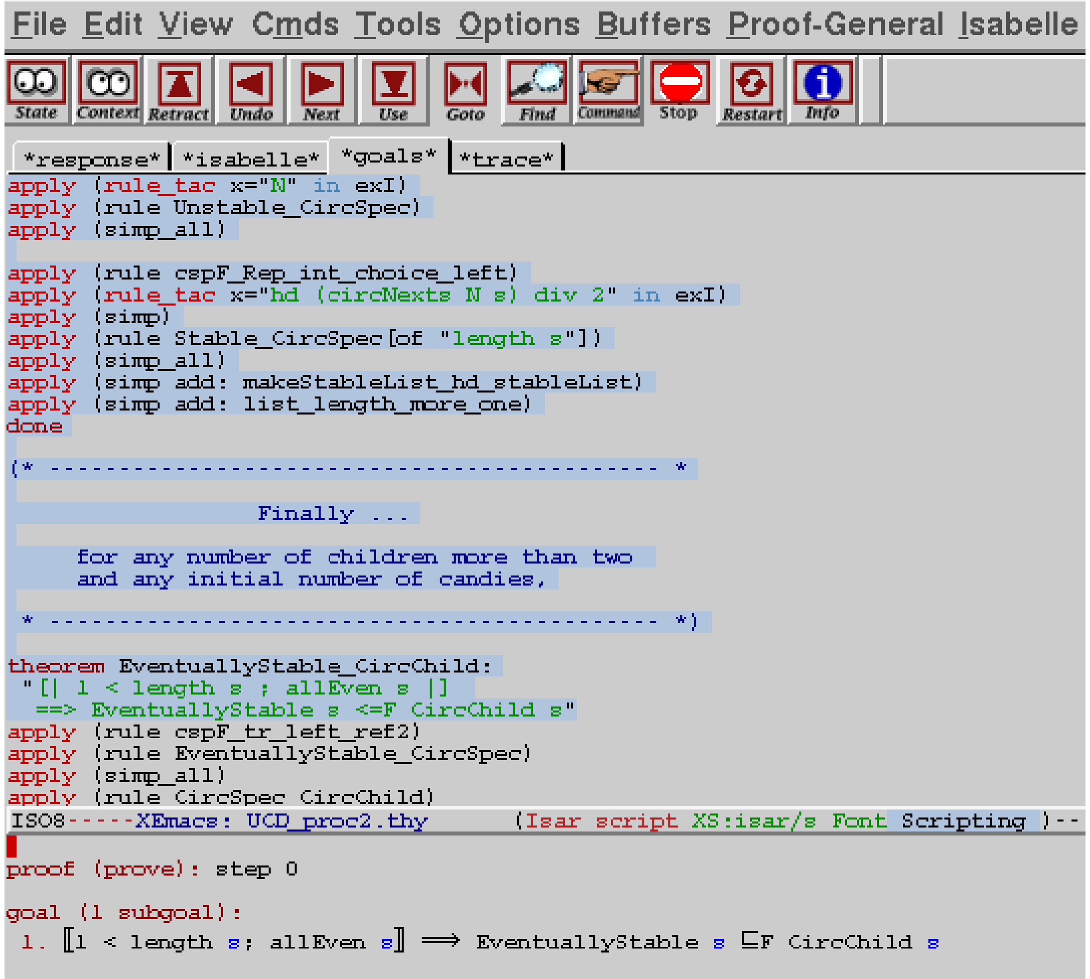
Simulate runs of a single instance and check that in these runs the puzzle stabilise.

Model-checking with FDR



Verify that a single instance of our puzzle stabilises.

Proof with CSP-Prover



Verify that all instances of our puzzle stabilise.

Code generation with CSP++

Versions 4.2 till 5.1

nothing but error messages
on the shown CSPm script

Reason:

- only CSP operators are supported;
- however, the functional programming language of CSPm has nearly no support.

The new Version 5.2

```
carmel ~/workspace/puzzle 0> ./puzzle > log
^C
carmel ~/workspace/puzzle 1> head -12 log
Action: d.1.0
Action: d.2.1
Action: d.0.2
Action: d.1.1
Action: d.2.1
Action: d.0.2
Action: d.1.2
Action: d.2.1
Action: d.0.2
Action: d.1.2
Action: d.2.2
Action: d.0.2
```

New in V5.2: Support for data types

- Sets + standard functions such as union, intersection, . . .
- Sequences + standard functions such as size, front . . .
- User defined functions:

```
pragma cspt function
fill(n) = if (n % 2 == 0) then n else n + 1
```

- User defined constants

CSP++ in a nutshell

Methodology:

- specify & analyse communication structure in CSP
- fully automatic translation to C++
- enrich the system with user coded functions

Technological basis:

- GNU Portable Threads

Relationship between CSP specification and code:

- trace refinement

Covered sub-language of CSP

CSP$_M$	JCSP	FSPJ	CSP++
STOP	✓	✓	✓
SKIP	✓	Not covered in FSP	✓
$a \to P$	(✓)	(✓)	(✓)
$P \mathbin{;} Q$	✓	×	✓
$P \setminus a$	×	×	✓
$P \Box Q$	✓	✓	✓
$P \sqcap Q$	×	Not covered in FSP	×
$b \& p$	×	×	×
$P\,[a \leftarrow b]$	×	×	✓
$P \mid\mid\mid Q$	(✓)	(✓)	✓
$P \mid[\alpha]\mid Q$	×	×	✓
$P \mid[\alpha \mid \beta]\mid Q$	✓	✓	×
$\mathbin{;} x : s @ P$	×	×	×
$\Box x : a @ P$	×	×	×
$\sqcap x : a @ P$	×	×	×
$\mid\mid\mid x : a @ P$	×	×	×
$\mid[\alpha]\mid x : a @ P$	×	×	×
$\mid\mid x : a @ [\alpha_a] P$	×	×	×

from: T Davies, CSP Implementation Techniques, Swansea 2012.

Conclusion

Summary & Future Work

CSP++

- provides fully automatic code generation from CSP
- has now wider support for data types

Future work:

- extend to cover more CSP operators
- further case studies