

Tools for teaching programming concepts @ DTU

Alberto Lluch Lafuente — Technical University of Denmark

Tools for teaching ~~programming concepts~~ @ DTU

Formal methods?

Alberto Lluch Lafuente — Technical University of Denmark

Some disclaimers...

- > This is not a commercial talk
- > Talk duration [??..40 min]
- > Credits go to several people (details will follow)

Credits

Many people have contributed to the tools in this talk

Panagiotis Vasilikos

Oliver Emil Bøving

Mike Castro Lundin

Martin Hansen

Maliina Hammeken

Kasper Laursen

Ioannis Karras

Hanne Riis Nielson

Flemming Nielson

Christoph Matheja

Camilla Færch

Alberto Lluch Lafuente

...

A bit of context...

Computer Science Modelling: a key course in CS/IT educations at DTU



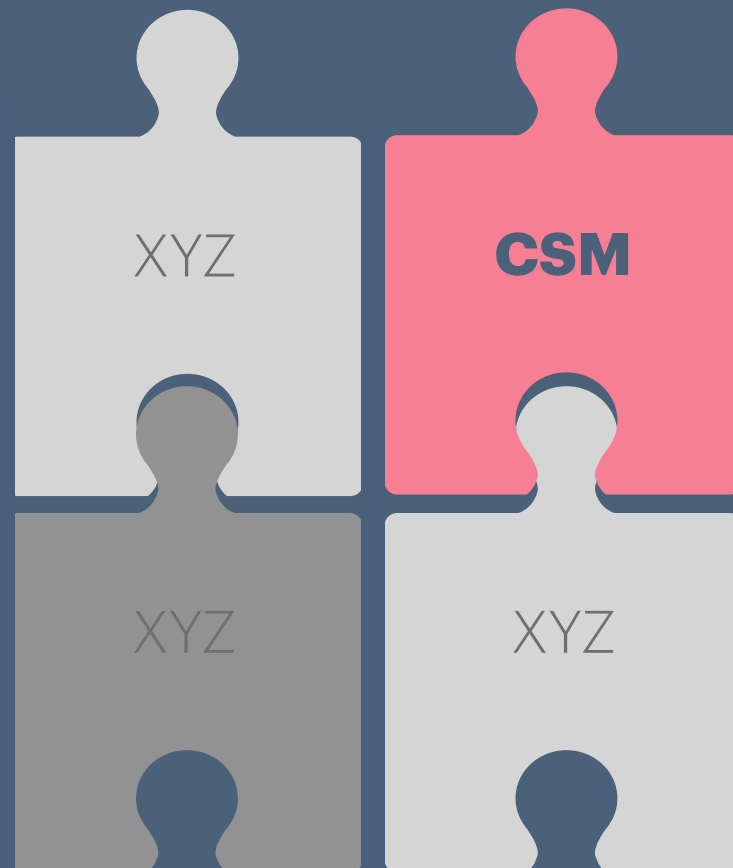
TOPICS

- > Formal languages (automata, grammars, ...)
- > Compilers & interpreters (semantics, parsing,...)
- > Formal methods (verification, static analysis, ...)

A bit of context...

Computer Science Modelling: a key course in CS/IT educations at DTU

BSc



TOPICS

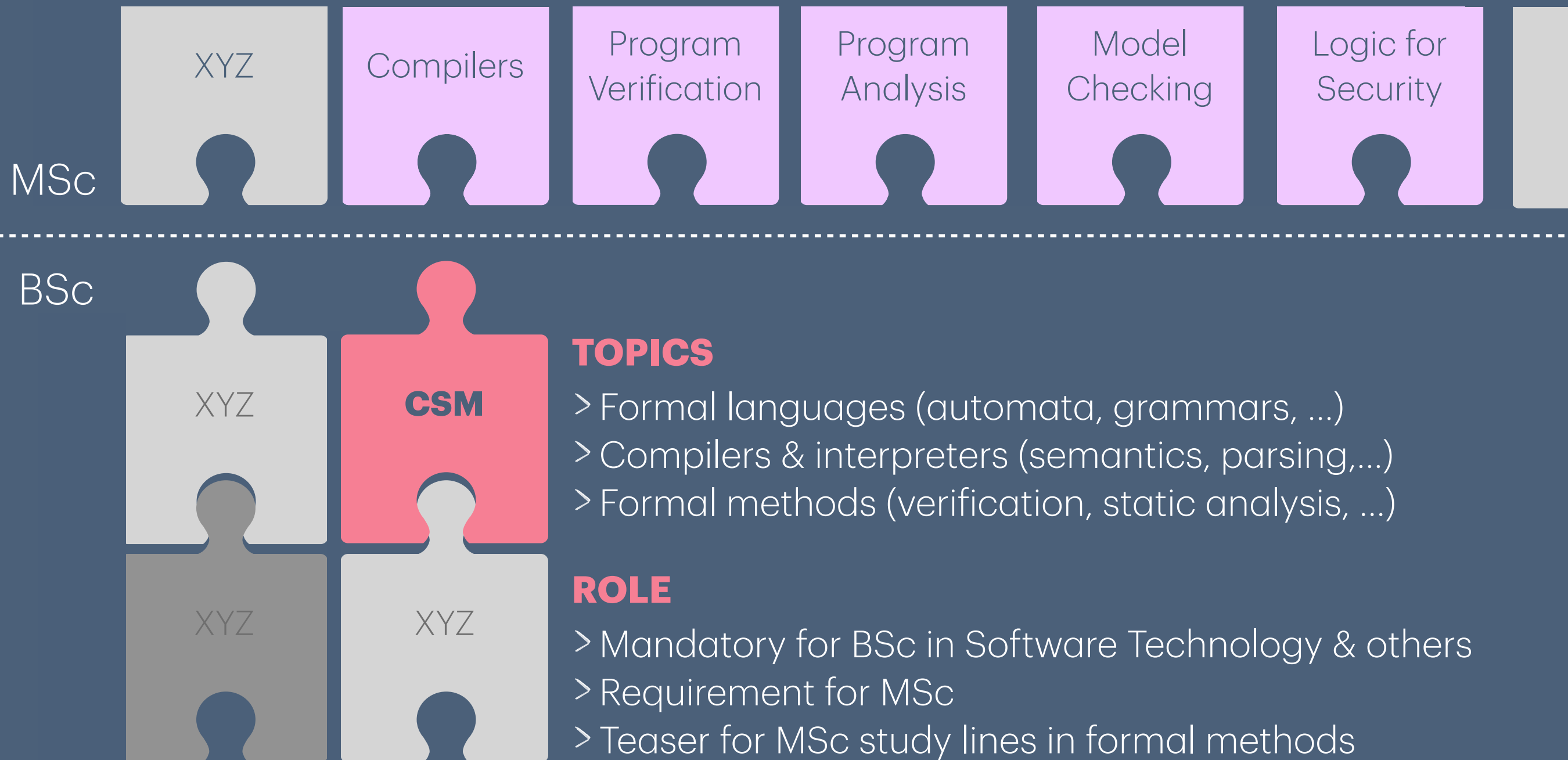
- > Formal languages (automata, grammars, ...)
- > Compilers & interpreters (semantics, parsing,...)
- > Formal methods (verification, static analysis, ...)

ROLE

- > Mandatory for BSc in Software Technology & others

A bit of context...

Computer Science Modelling: a key course in CS/IT educations at DTU



3 key “meta” learning objectives

APPLY METHODS

e.g. use a provided flow analyser to reason about security

IMPLEMENT METHODS

e.g. implement flow analyser

UNDERSTAND METHODS

e.g. information flow analysis
& non-interference

3 key “meta” learning objectives

APPLY METHODS

e.g. use a provided flow analyser to reason about security

UNDERSTAND METHODS

e.g. information flow analysis
& non-interference



IMPLEMENT METHODS

e.g. implement flow analyser

3 key “meta” learning objectives

APPLY METHODS

e.g. use a provided flow analyser to reason about security

UNDERSTAND METHODS

e.g. information flow analysis
& non-interference



IMPLEMENT METHODS

e.g. implement flow analyser

Role of tools:

> Enable feedback loops between LOs

3 key “meta” learning objectives

APPLY METHODS

e.g. use a provided flow analyser to reason about security

UNDERSTAND METHODS

e.g. information flow analysis
& non-interference



IMPLEMENT METHODS

e.g. implement flow analyser

Role of tools:

- > Enable feedback loops between LOs
- > Empower students to self-assess LOs
- > Empower teachers to self-assess LOs



SCALE!

Programs as a case-study

Program

[+ specification]



Something interesting
about the program



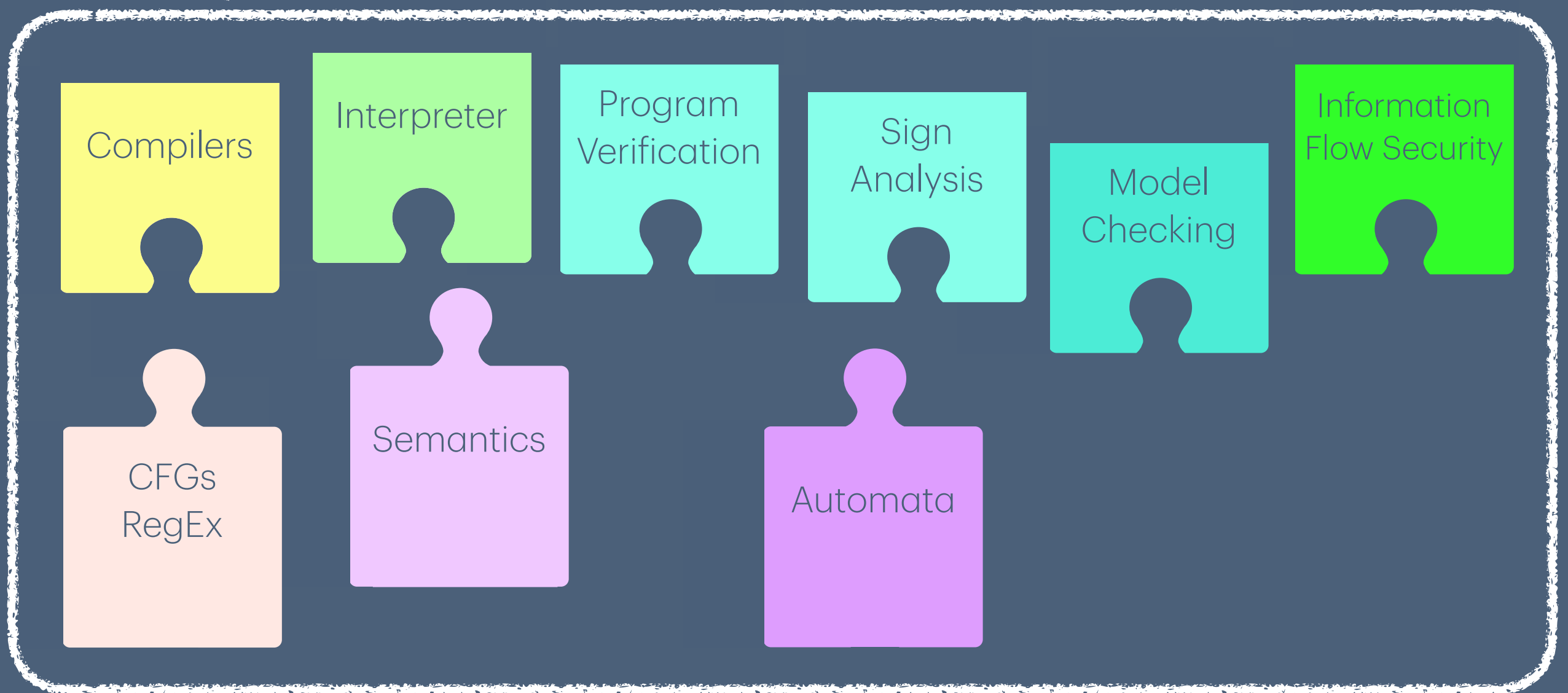
Programs as a case-study

Program

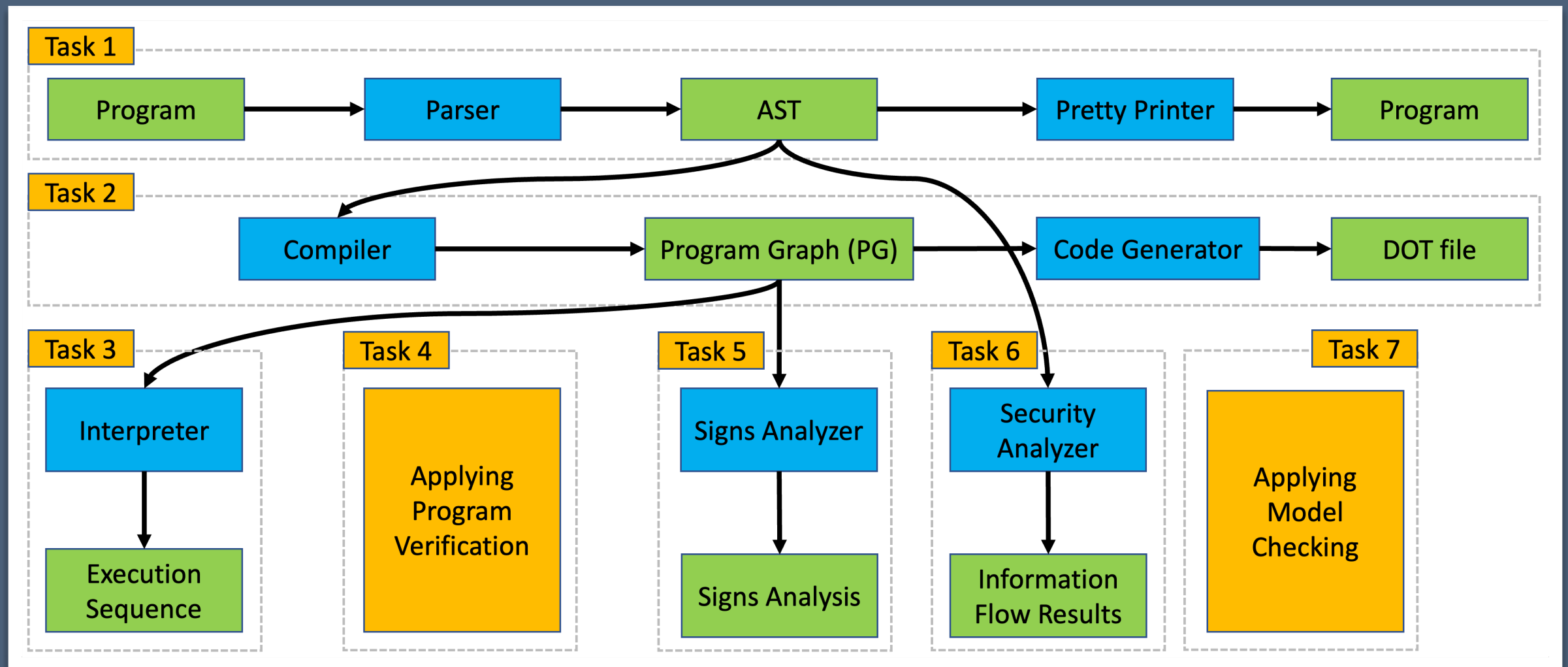
[+ specification]



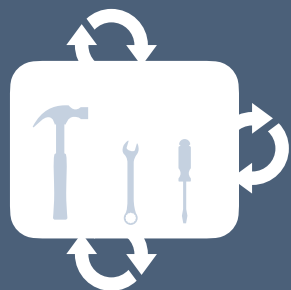
Something interesting
about the program



Programs as a case-study



APPLY METHODS



IMPLEMENT

- > Follow formal definitions from teaching material
- > Match behaviour of reference implementations
- > Boilerplate provided (e.g. UI), focus on fundamentals

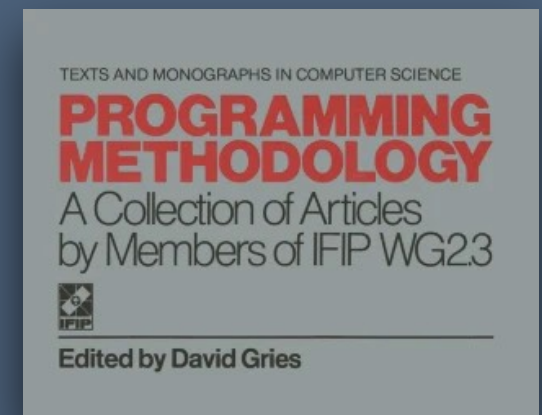
UNDERSTAND

Programs as a case-study

Programming Language: a variant of Guarded Command Language
[Dijkstra, Commun. ACM 18 (1975), 8: 1975)]

This is how factorial looks:

```
y:=1;  
do x>0 -> y:=x*y;  
          x:=x-1  
od
```

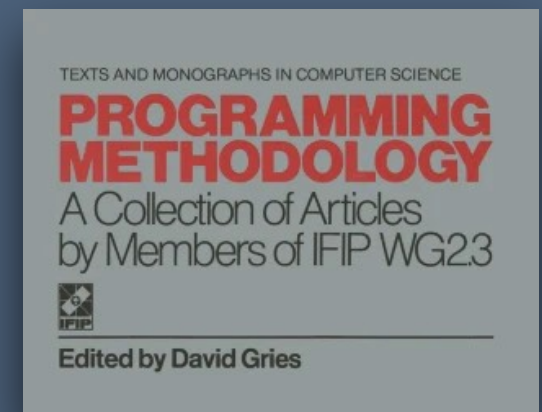


Programs as a case-study

Programming Language: a variant of Guarded Command Language
[Dijkstra, Commun. ACM 18 (1975), 8: 1975)]

This is how factorial looks:

```
y:=1;  
do x>0 -> y:=x*y;  
          x:=x-1  
od
```



Main features:

- > Only two type of variables: Integers & integer arrays
- > Basic arithmetic & Boolean expressions
- > Control flow constructs: if & loops
- > Non-determinism

Formal syntax & semantics in teaching material :)

Non-det. sort

```
do  
    a<b -> x := a; a:=b; b:=x  
[] b<c -> x := b; b:=c; c:=x  
[] c<d -> x := c; c:=d; d:=x  
od
```

BROWSER VERSION

- > Immediate feedback “as-you-type”
 - > Random test-case generation
 - > Good-looking browser-based UIs
 - > Zero-to-minimal installations needed
- > Empower students to self-assess the LOs
- > Avoid scaring away students from FM!

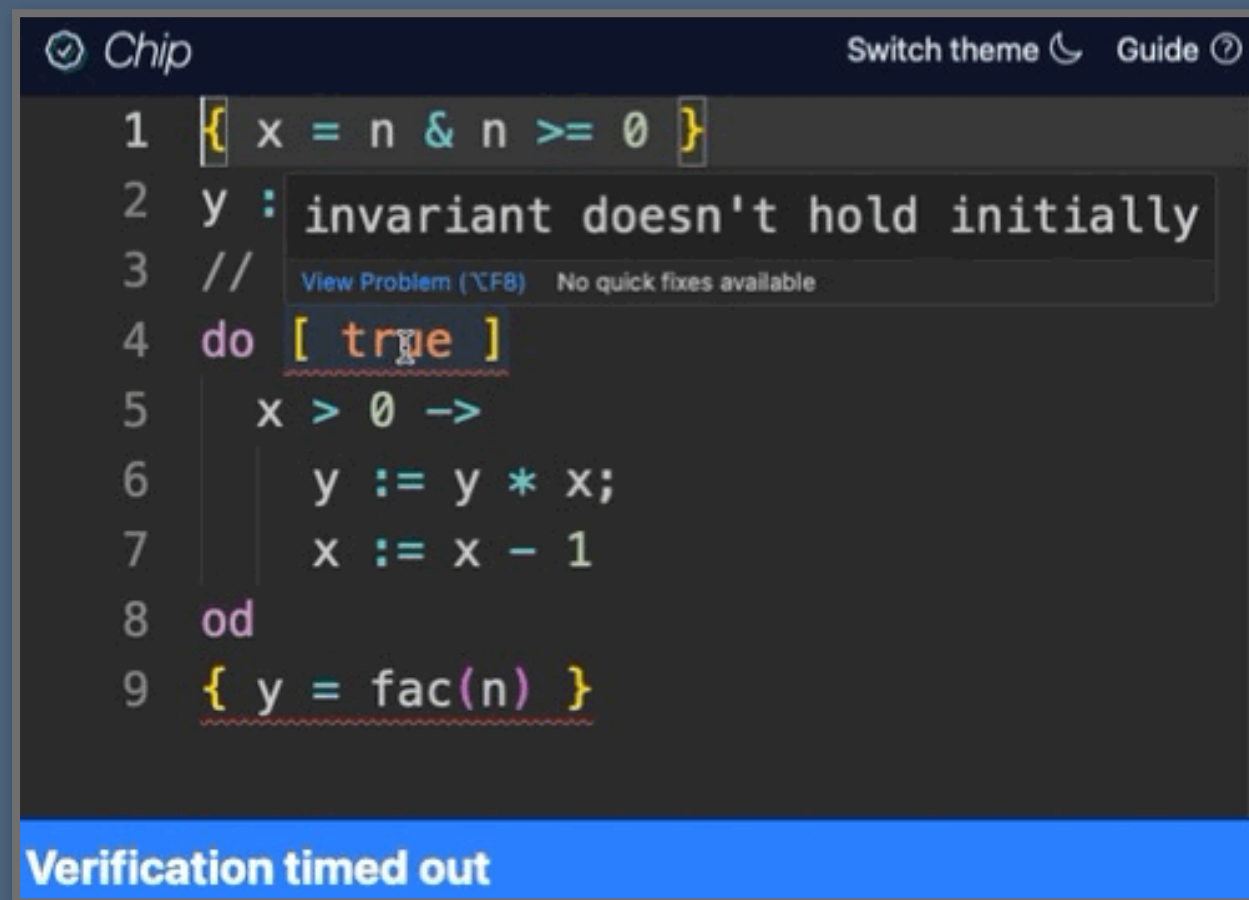
The screenshot shows the Chip IDE interface. At the top, there's a header with the 'Chip' logo, a 'Switch theme' button, and a 'Guide' link. The main area displays a code snippet with line numbers 1 through 9. The code is as follows:

```
1 { x = n & n >= 0 }
2 y : invariant doesn't hold initially
3 //
4 do [ true ]
5   x > 0 ->
6     y := y * x;
7     x := x - 1
8 od
9 { y = fac(n) }
```

A tooltip is visible over the code, displaying 'View Problem (^F8)' and 'No quick fixes available'. At the bottom of the IDE, a blue status bar indicates 'Verification timed out'.

BROWSER VERSION

- > Immediate feedback “as-you-type”
 - > Random test-case generation
 - > Good-looking browser-based UIs
 - > Zero-to-minimal installations needed
- > Empower students to self-assess the LOs
- > Avoid scaring away students from FM!

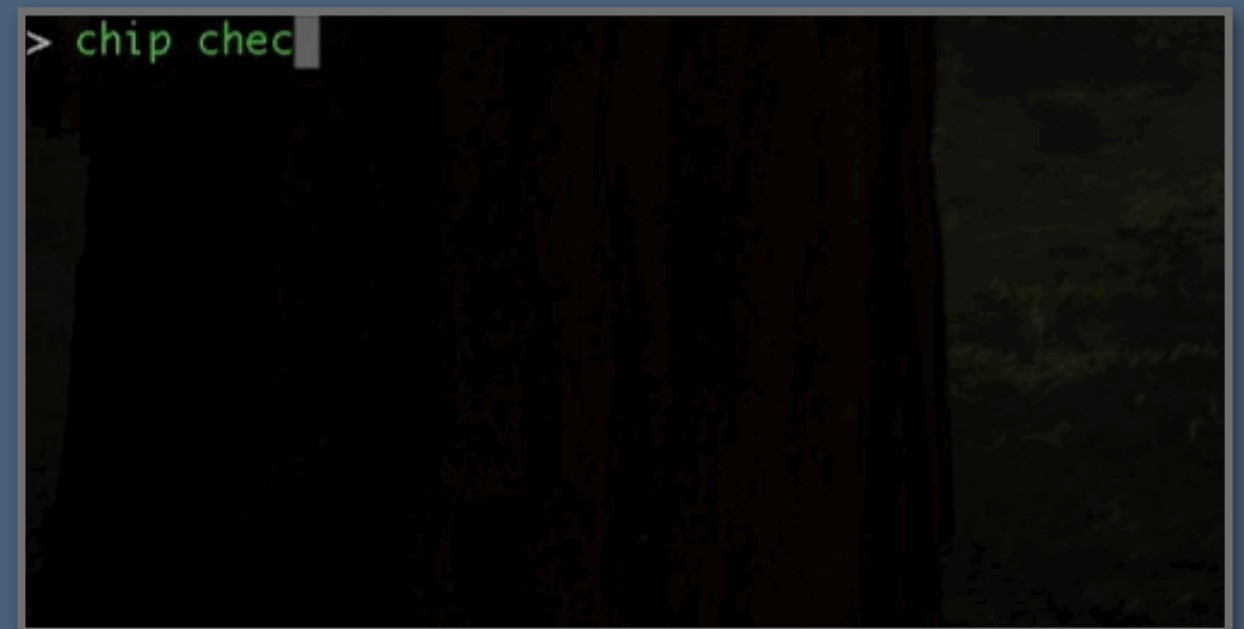


The screenshot shows the Chip browser-based verification tool. The interface has a dark theme. At the top, there's a header with the 'Chip' logo, a 'Switch theme' button, and a 'Guide' link. The main area displays a code snippet in a programming language that uses curly braces for assertions and loops. The code is as follows:

```
1 { x = n & n >= 0 }
2 y : invariant doesn't hold initially
3 //
4 do [ true ]
5   x > 0 ->
6     y := y * x;
7     x := x - 1
8 od
9 { y = fac(n) }
```

Below the code, there's a blue status bar that reads "Verification timed out". A tooltip is visible over the code, showing "View Problem (^F8)" and "No quick fixes available".

COMMAND-LINE VERSION



The screenshot shows the command-line version of the Chip tool. It features a dark background with a green prompt character followed by the text "chip chec".



Empower teachers to grade/assess the LOs

fm4fun - the original toolset

GCL
program



Program
Graph



fm4fun About Download Environments

Input Program

Extended Guarded Commands Code ☒

```
y:=1;  
do x>0 -> y:=x*y;  
           x:=x-1  
od
```

Step-wise Execution

Explore how the memory changes, upon execution of the actions. Recall that if more guard is deterministic which action to take.

When using the **Show Trace** feature, non-deterministic choices will be shown in the first configuration. When using the **Show Resulting Configurations** feature, all non-deterministic choices will be shown in the first configuration. A short summary of the final configurations is the input code is highly non-deterministic, and the number of steps is high, the tree of configurations grows big (and therefore cause long execution).

For both outputs, all non-terminated and non-stuck configurations can be extended with the **Show Resulting Configurations** feature.

Initialization of Variables and Arrays

→ x = 3, y = 0

Number of Steps

100

Show Trace Show Resulting Configurations

Action	Node	Memory
	q0	3
y:=1	q1	3
x>0	q2	3
y:=x*y	q3	3
x:=x-1	q1	2
x>0	q2	2
y:=x*y	q3	2
x:=x-1	q1	1
x>0	q2	1
y:=x*y	q3	1
x:=x-1	q1	0
!(x>0)	q0	0

Terminated Successfully terminated in 11 steps.

Program Graph

Non-det. Det.

fm4fun About Download Environments

Input Program

Extended Guarded Commands Code ☒

```
i:=0;  
x:=0;  
y:=0;  
do i<10 -> if A[i]>=0 ->  
           x:=x+A[i];  
           i:=i+1  
           [] A[i]<0 -> i:=i+1;  
           break  
fi;  
y:=y+1  
od
```

Deterministic Model ☒

Non-deterministic Deterministic

Specify Security Lattice

Templates public < private ☒

Security Classification for Variables and Arrays

→ i = public, x = public, y = public, A = private ☒

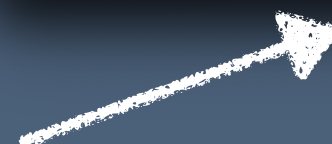
Program Graph

Non-det. Det.

Show Security Analysis

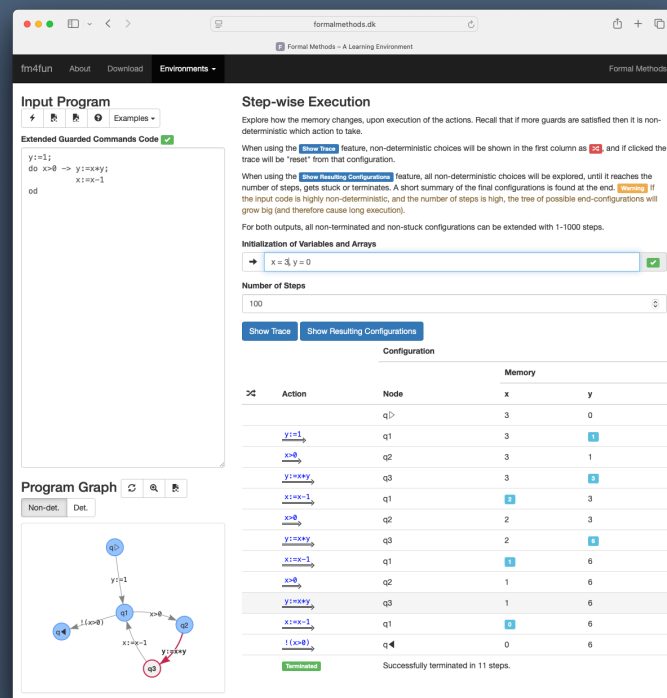
Flows	
Actual	A → i, A → x, i → i, i → x, i → y, x → x, y → y
Allowed	A → A, i → A, i → i, i → x, i → y, x → A, x → i, x → x, x → y, y → A, y → i, y → x, y → y
Violations	A → i, A → x
Result	Not Secure

Analysis (e.g. executions, security)



Tool & Teaching material

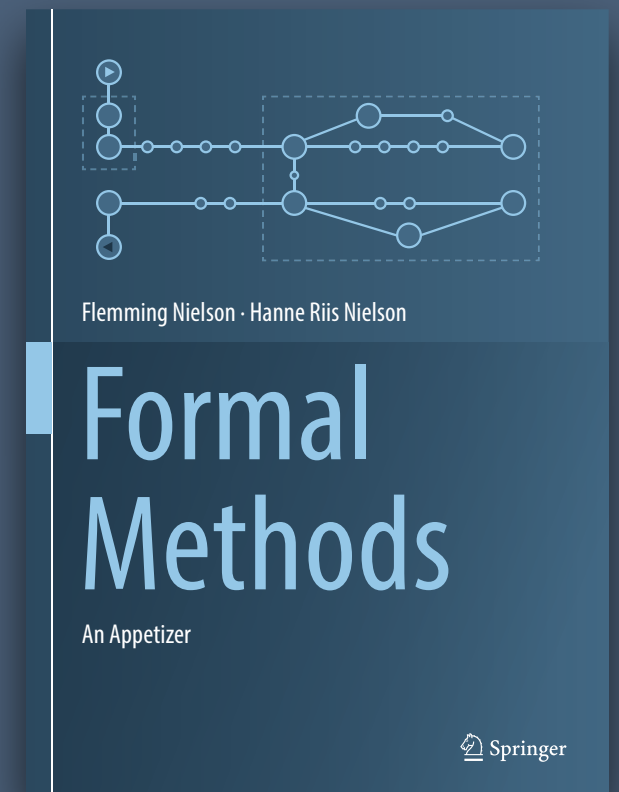
<https://formalmethods.dk>



Tool support for GCL programs:

- > Parser
- > Program Graph representation
- > Interpreter
- > Information flow analysis
- > Sign analysis
- > Concurrency
- > ...

Teaching Material



+ videos
+ slides

<https://team-checkr.github.io/chip>

Program Verification for GCL

Annotated GCL
program

The screenshot shows a web browser window with the address bar displaying `team-checkr.github.io`. The page title is "Chip". The interface includes a "Switch theme" button and a "Guide" link. The main content area displays a GCL program with line numbers 1 through 9. The program is annotated with preconditions, invariants, and postconditions. The verification result is shown at the bottom in a green bar, indicating that the program is verified but not fully annotated.

```
1 { x = n & n >= 0 }
2 y := 1;
3 do [ x >= 0 & y * fac(x) = fac(n) ]
4   x > 0 ->
5     y := y * x;
6     x := x - 1
7 od
8 { y = fac(n) }
```

Verified The program is *not* fully annotated

Precondition

Invariant

Postcondition

Verification result

<https://team-checkr.github.io/moka>

An LTL model checker for parallel GCL

Parallel GCL
Program

LTL spec

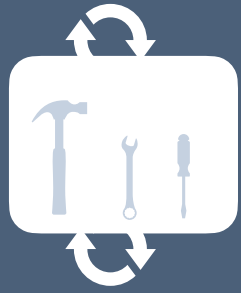
Transition
System

The screenshot displays the Moka web interface with the following components:

- Parallel GCL Program:** A code editor on the left containing a parallel GCL program. It includes variables like `n`, `turn`, `wants_to_enter_0`, `wants_to_enter_1`, `in_0`, and `in_1`. The program uses `par` blocks for parallel execution and `do` blocks for guarded commands. A comment at the bottom states "LTL property does not hold".
- LTL spec:** A table at the bottom left showing a sequence of states (Step, in_0, in_1, n, turn, wants_to_enter_0, wants_to) used as counterexamples. The states are numbered 1 through 7.
- Transition System:** A complex state transition graph on the right, representing the model checked by Moka. It consists of numerous states connected by transitions, with some states highlighted in orange.
- Result:** A red bar at the bottom of the interface with the text "Error".

Result: yes / no + counterexamples

APPLY METHODS



Inspectify

UNDERSTAND

checkr.compute.dtu.dk

Inspectify Calculator Parser Compiler Interpreter Security Sign

Show reference ☒

Generate

```
y:=1;  
do x>0 -> y:=x*y;  
    x:=x-1  
od
```

Initialization of variables and arrays

x	3
y	0

Options

Number of steps 14

Determinism Deterministic NonDeterministic

Jobs:

Action	Node	x	y
	q▷	3	0
y := 1	q ₁	3	1
(x > 0)	q ₂	3	1
y := (x * y)	q ₃	3	3
x := (x - 1)	q ₁	2	3
(x > 0)	q ₂	2	3
y := (x * y)	q ₃	2	6
x := (x - 1)	q ₁	1	6
(x > 0)	q ₂	1	6
y := (x * y)	q ₃	1	6
x := (x - 1)	q ₁	0	6
!(x > 0)	q◀	0	6

Terminated

v0.2.11

APPLY METHODS

Inspectify



IMPLEMENT

UNDERSTAND

Trace

Program
Graph

GCL program

```
47 let rec evalBool expr : boolExpr memory : InterpreterMemory : bool = /
48   match expr with
49   | Bool b : bool -> b
50   | Negation b : boolExpr -> not (evalBool expr = b memory)
51   | LogicExpr(lhs : boolExpr, op : BLOp, rhs : boolExpr) ->
52     let evalLhs : bool = evalBool expr = lhs memory
53     let evalRhs : bool = evalBool expr = rhs memory
54     match op with
55     | SAnd -> evalLhs && evalRhs
56     | SOr -> evalLhs || evalRhs
57     | And -> evalLhs && evalRhs
58     | Or -> evalLhs || evalRhs
59   | Comparison(lhs : arithExpr, op : BROp, rhs : arithExpr) ->
60     let evalLhs : int = evalArith expr = lhs memory
61     let evalRhs : int = evalArith expr = rhs memory
62     match op with
63     | Eq -> evalLhs = evalRhs
64     | NEq -> evalLhs <> evalRhs
65     | GEq -> evalLhs >= evalRhs
66     | LEq -> evalLhs <= evalRhs
67     | Gt -> evalLhs > evalRhs
68     | Lt -> evalLhs < evalRhs
69
70 // Semantic function 'S', which specifies how the memory is affected
71 // (labels in the program graph) - evaluation of commands
```

student's code

Inspectify Calculator Parser Compiler Interpreter Security Sign

Generate

```
if
true || 1/0=1/0 -> skip
fi
```

Initialization of variables and arrays

Options

Number of steps 14

Determinism Deterministic NonDeterministic

Input JSON Output Output JSON Reference Output Validation ✗

No stuck execution found See output

Jobs: 14 succeeded 22 failed v0.2.6

testing vs reference implementation

Starter videos

Book
(formal spec)

The collage consists of three overlapping images. The leftmost image is a screenshot of the VS Code editor with the 'Compiler.fs' file open, showing F# code for a compiler. The middle image is a screenshot of a PDF document titled 'book - 2019_Book_FormalMethods.pdf', showing a formal specification for edges. The rightmost image is a screenshot of the 'Inspectify' web application, showing a task description for implementing a function and a generated graph with nodes 'qS' and 'qF' connected by an edge labeled 'skip'.

```
edges( $q_0 \rightsquigarrow q_1$ )[ $x \cdot a$ ] = {( $q_0, x \cdot a, q_1$ )}
```

```
edges( $q_0 \rightsquigarrow q_1$ )[skip] = {( $q_0, skip, q_1$ )}
```

```
let analysis (input: Input) : Output =  
  failWith "Compiler not yet implemented" // TODO: start here
```

```
graph LR  
  qS((qS)) -- skip --> qF((qF))
```

Task
description
(Informal)

coding

inspectify

Inspectify - scoreboard

Checko

Last update: 23/09/2025, 15:13:39

	Parser	Compiler	Interpreter	Security	Sign
Yogurt-Yaks	Finished - be7a95e	Finished - be7a95e	Finished - be7a95e	Finished - be7a95e	Finished - be7a95e
Tortilla-Toads	Finished - 125b041	Finished - 125b041	Finished - 125b041	Finished - 125b041	Finished - 125b041
Tangerine-Tarantulas	Finished - 88a7d49	Finished - 88a7d49	Finished - 88a7d49	Finished - 88a7d49	Finished - 88a7d49
Strudel-Stingrays	Finished - b2edb4e	Finished - b2edb4e	Finished - b2edb4e	Finished - b2edb4e	Finished - b2edb4e
Strudel-Skunks	Finished - c7c3fa3	Finished - c7c3fa3	Finished - c7c3fa3	Finished - c7c3fa3	Finished - c7c3fa3
Sprout-Sparrows	Finished - f48c8da	Finished - f48c8da	Finished - f48c8da	Finished - f48c8da	Finished - f48c8da
Sorbet-Seagulls	Finished - cb82c68	Finished - cb82c68	Finished - cb82c68	Finished - cb82c68	Finished - cb82c68
Scone-Scorpions	Finished - bad5c7f	Finished - bad5c7f	Finished - bad5c7f	Finished - bad5c7f	Finished - bad5c7f
Samosa-Sharks	Finished - d373b82	Finished - d373b82	Finished - d373b82	Finished - d373b82	Finished - d373b82
Raisin-Rats	Finished - a70a6f8	Finished - a70a6f8	Finished - a70a6f8	Finished - a70a6f8	Finished - a70a6f8
Quiche-Quokkas	Finished - 5c18d45	Finished - 5c18d45	Finished - 5c18d45	Finished - 5c18d45	Finished - 5c18d45
Pickle-Pumas	Finished - be9ef7c	Finished - be9ef7c	Finished - be9ef7c	Finished - be9ef7c	Finished - be9ef7c
Pesto-Pigeons	Finished - 28e8bb4	Finished - 28e8bb4	Finished - 28e8bb4	Finished - 28e8bb4	Finished - 28e8bb4
Peanut-Butter-Parrots	Finished - 84231f1	Finished - 84231f1	Finished - 84231f1	Finished - 84231f1	Finished - 84231f1
Omelet-Owls	Finished - 3e8cb1e	Finished - 3e8cb1e	Finished - 3e8cb1e	Finished - 3e8cb1e	Finished - 3e8cb1e
Noodle-Narwhals	Finished - 87449f5	Finished - 87449f5	Finished - 87449f5	Finished - 87449f5	Finished - 87449f5
Miso-Moles	Finished - 1386664	Finished - 1386664	Finished - 1386664	Finished - 1386664	Finished - 1386664
Meatball-Mice	Finished - 980f271	Finished - 980f271	Finished - 980f271	Finished - 980f271	Finished - 980f271
Marshmallow-Monkeys	Finished - a1f4cfa	Finished - a1f4cfa	Finished - a1f4cfa	Finished - a1f4cfa	Finished - a1f4cfa
Mango-Moose	Finished - 2fbf8c8	Finished - 2fbf8c8	Finished - 2fbf8c8	Finished - 2fbf8c8	Finished - 2fbf8c8
Macaron-Moles	Finished - 7b3196b	Finished - 7b3196b	Finished - 7b3196b	Finished - 7b3196b	Finished - 7b3196b
Jellybean-Jackals	Finished - 3d37ea1	Finished - 3d37ea1	Finished - 3d37ea1	Finished - 3d37ea1	Finished - 3d37ea1
Hazelnut-Hippos	Finished - 7c5973f	Finished - 7c5973f	Finished - 7c5973f	Finished - 7c5973f	Finished - 7c5973f
Gnocchi-Gophers	Finished - 7ac30ee	Finished - 7ac30ee	Finished - 7ac30ee	Finished - 7ac30ee	Finished - 7ac30ee
Gingerbread-Goats	Finished - ea2bc73	Finished - ea2bc73	Finished - ea2bc73	Finished - ea2bc73	Finished - ea2bc73
Fondue-Ferrets	Finished - 8a98bf7	Finished - 8a98bf7	Finished - 8a98bf7	Finished - 8a98bf7	Finished - 8a98bf7
Dumpling-Ducks	Finished - 67bdede	Finished - 67bdede	Finished - 67bdede	Finished - 67bdede	Finished - 67bdede
Cucumber-Cougars	Finished - a70211d	Finished - a70211d	Finished - a70211d	Finished - a70211d	Finished - a70211d
Cookie-Kangaroos	Finished - 3c0fe04	Finished - 3c0fe04	Finished - 3c0fe04	Finished - 3c0fe04	Finished - 3c0fe04
Coconut-Cobras	Finished - 3575a4f	Finished - 3575a4f	Finished - 3575a4f	Finished - 3575a4f	Finished - 3575a4f
Churro-Chickens	Finished - d6ebc9c	Finished - d6ebc9c	Finished - d6ebc9c	Finished - d6ebc9c	Finished - d6ebc9c
Cheesecake-Chimpanzees	Finished - 3184305	Finished - 3184305	Finished - 3184305	Finished - 3184305	Finished - 3184305
Cabbage-Crabs	Finished - 80107a7	Finished - 80107a7	Finished - 80107a7	Finished - 80107a7	Finished - 80107a7
Burrito-Llamas	Finished - 936d296	Finished - 936d296	Finished - 936d296	Finished - 936d296	Finished - 936d296
Bagel-Bugs	Finished - 4717d19	Finished - 4717d19	Finished - 4717d19	Finished - 4717d19	Finished - 4717d19
Cereal-Snails	Finished - 47dfac8	Finished - 47dfac8	Finished - 47dfac8	Finished - 47dfac8	Finished - 47dfac8
Baguette-Beetles	Finished - e2f08c2	Finished - e2f08c2	Finished - e2f08c2	Finished - e2f08c2	Finished - e2f08c2
Salsa-Snakes	Finished - 5881a3a	Finished - 5881a3a	Finished - 5881a3a	Finished - 5881a3a	Finished - 5881a3a
Croissant-Coyotes	Finished - 8d8aa9d	Finished - 8d8aa9d	Finished - 8d8aa9d	Finished - 8d8aa9d	Finished - 8d8aa9d
Cabbage-Cobras	Finished - 97f0f44	Finished - 97f0f44	Finished - 97f0f44	Finished - 97f0f44	Finished - 97f0f44
Radish-Rattlesnakes	Finished - 2c442a4	Finished - 2c442a4	Finished - 2c442a4	Finished - 2c442a4	Finished - 2c442a4
Cantaloupe-Camels	Finished - 9189445	Finished - 9189445	Finished - 9189445	Finished - 9189445	Finished - 9189445
Ravioli-Raccoons	Finished - 432d865	Finished - 432d865	Finished - 432d865	Finished - 432d865	Finished - 432d865
Cranberry-Coyotes	Finished - 2d5dde3	Finished - 2d5dde3	Finished - 2d5dde3	Finished - 2d5dde3	Finished - 2d5dde3
Croquette-Crows	Finished - 98d6ec7	Finished - 98d6ec7	Finished - 98d6ec7	Finished - 98d6ec7	Finished - 98d6ec7
Muffin-Meerkats	Finished - 03c9f7d	Finished - 03c9f7d	Finished - 03c9f7d	Finished - 03c9f7d	Finished - 03c9f7d
Chowder-Chinchillas	Finished - 37d5cf7	Finished - 37d5cf7	Finished - 37d5cf7	Finished - 37d5cf7	Finished - 37d5cf7
Gumbo-Gorillas	Finished - 091e417	Finished - 091e417	Finished - 091e417	Finished - 091e417	Finished - 091e417
Chili-Crows	Finished - 0865ad6	Finished - 0865ad6	Finished - 0865ad6	Finished - 0865ad6	Finished - 0865ad6
Popcorn-Penguins	Finished - efa4eaa	Finished - efa4eaa	Finished - efa4eaa	Finished - efa4eaa	Finished - efa4eaa
Pancake-Pandas	Finished - 9b50c68	Finished - 9b50c68	Finished - 9b50c68	Finished - 9b50c68	Finished - 9b50c68
Gumbo-Gulls	Finished - 3fb014c	Finished - 3fb014c	Finished - 3fb014c	Finished - 3fb014c	Finished - 3fb014c
Blueberry-Buffalos	Finished - d588edb	Finished - d588edb	Finished - d588edb	Finished - d588edb	Finished - d588edb
Pizza-Pigeons	Finished - eb5fc33	Finished - eb5fc33	Finished - eb5fc33	Finished - eb5fc33	Finished - eb5fc33
Coltslaw-Cockatoos	Finished - 46a3e68	Finished - 46a3e68	Finished - 46a3e68	Finished - 46a3e68	Finished - 46a3e68
Cinnamon-Cicadas	Finished - 2338f6e	Finished - 2338f6e	Finished - 2338f6e	Finished - 2338f6e	Finished - 2338f6e
Waffle-Walruses	Finished - da13548	Finished - da13548	Finished - da13548	Finished - da13548	Finished - da13548

We periodically pull student’s code, run tests and update the scoreboard (gamification)

Links

<https://formalmethods.dk>

<https://team-checkr.github.io/chip>

<https://team-checkr.github.io/moka>

<https://github.com/team-checkr/checkr>

Questions?

albl@dtu.dk