CoCoA 5

Giovanni Lagorio (lagorio@disi.unige.it)

COCOA 2009

International School on Computer Algebra

Barcelona, 8-12 June 2009

To set the record straight

- I'm a computer scientist
- I cannot tell apart
  - polynomials and
  - quinces

x^2+5x-1

- I have write access to the CoCoA source repository
- If you're not scared, you were not paying attention ☺

Luckily…

- My work and this talk are about the *CoCoA programming language*
- So far, I haven’t done much damage
  - At least, that what I’d like to think ☺
- Key decisions were already made
  - Can’t get the blame or praise

Plan of the talk

- Why we need backward *in*compatibility
- CoCoA 5
- The transition path
- Conclusions

CoCoA 4 $\not\subset$ CoCoA 5

- CoCoA 4 is an incredibly flexible language
- Easy to use!
- Easy to misuse! ☹
- As a newbie, I find that
  - some constructs have a "funny semantics"
    (they're probably ok when used properly, but beginners tend to think outside the box ☺)
  - error reporting is rather bad
- CoCoA 5 will be
  - still easy to use, but
  - way harder to misuse
- The price to pay? It won't be 100% backward compatible

An warming-up example

- Two := 2; -- Assignment of an integer
- L := [1, 2, 3]; -- Assignment of a list
- 2 [1, 2, 3]; -- Multiplication, yields the list [2, 4, 6]
- [1, 2, 3] 2; -- Same here
- 2 L; -- Variables and values can be mixed
- L 2; -- and matched as expected
- Two L; -- ~~Obviously~~, yielding the same result
- L Two; -- ...

sometimes

- Two [1, 2, 3]; -- ... → ERROR: Bad parameters
- [1, 2, 3] Two; -- ... CONTEXT: Two[1][2][3]

Problem: lack of uniformity

- if **operator** [] allows accessing the n-th element of a list, why [**2, 3, 5**] [N] doesn't work? Remember: **L**[N] does work
- (quoting from the manual) "For multiplication, one may use *, parentheses, or just a space". Why **L** [N] doesn't multiply **L** and [N], yet [**1, 2, 3**]N does multiply them?
- Why **xX** is a product but **Xx** is a single identifier?
- **x2** is a product, so they are **2x** and **2X**, yet **X2** is a single identifier
- …

A peculiar function definition

```
Define F(F)
  If F (F-1) (F) = 0 Then
      Return 1;
  Else
      Return F(F-1)(F);
  Endif
EndDefine;
F := 5;
-(-1  F)F(F  -1);
```

I’d like to point out that

- It’s the definition of a pretty well-known function
  - and an example of using it
- Everything is 100% legit CoCoA 4 code (that is, I’m not exploiting a bug of the interpreter)
- I do know that no one in their right mind would ever write code like this
  - Unless she/he wants to prove a point
  - ...and I do ☺

12 occurrences of F; 3 defs, 9 usages

```
Define F(F)
  If F (F-1) (F) = 0 Then
       Return 1;
  Else
       Return F(F-1)(F);
  Endif
EndDefine;
F := 5;
-(-1  F)F(F  -1);
```

So, in an expression, what does *F* mean?
Which *F* is which?

So similar, yet so different…

```
Define F(F)
  If F (F-1) (F) = 0 Then
       Return 1;
  Else
       Return F(F-1)(F);
  Endif
EndDefine;
F := 5;
-(-1  F)F(F  -1);
```

Why multiplication is not commutative?

```
Define F(F)
  If F (F-1) (F) = 0 Then
      Return 1;
  Else
      Return F(F-1)(F);
  Endif
EndDefine;
F := 5;
-(-1  F)F(F  -1);
```

Anyway, here it is the **factorial** function:

```
Define F(F)
  If F (F-1) (F) = 0 Then
        Return 1;
  Else
        Return F(F-1)(F);
  Endif
EndDefine;
F := 5;
-(-1  F)F(F  -1); -- as expected, 120, that is, 5!
```

Bottom line

- CoCoA 4 silently accepts
  - *dangerous code*: every piece of code whose semantics depends on the presence or absence of a blank is a bomb waiting to explode
  - *suspicious code*: does 1/2*x mean (1/2)*x or 1/(2*x)?
- CoCoA 5 won't. It will
  - reject suspicious constructs (depending on severity, warnings or errors will be issued)
    - This helps users to avoid common errors and pitfalls
  - have a single namespace for variables, functions and indeterminates (x2, xyz, A42, foo, Bar … will be valid identifier for any of those)

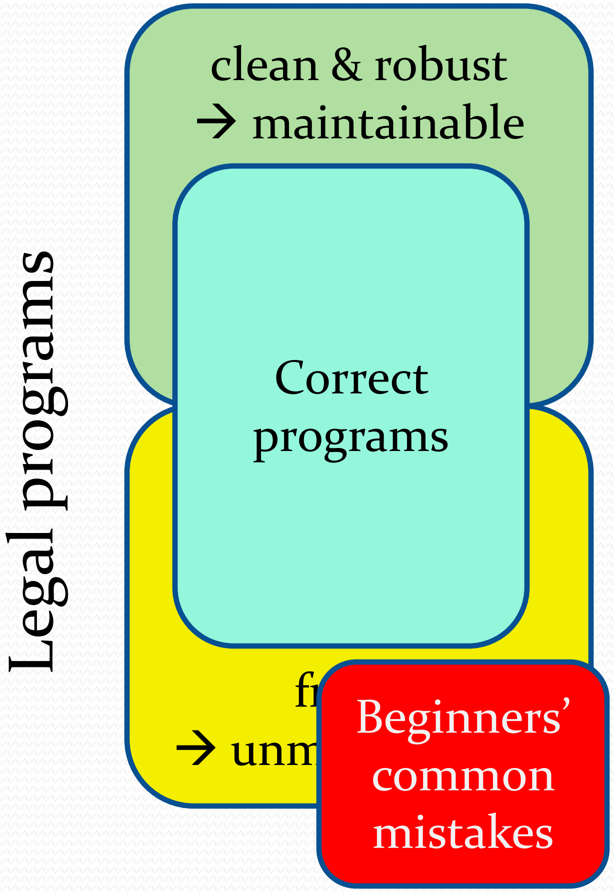
Polynomials are special

- **5x^2+3xy+1** looks better than **5*x^2+3*x*y+1**
- In CoCoA 5 special parentheses allow to use implict multiplication in well-marked regions; for instance,
  P := ${ 5x^2+3xy+1 }$; -- might not be the final syntax
- This is an expression-level construct
- Still, not 100% compatible:
  2 x is equivalent to **x 2** in CoCoA 4; but
  x 2 is rejected by CoCoA 5 (does it mean **x*2** or **x^2**?)

Interactive input is special too

- A context-sensitive prompt helps the user to understand what’s going on
  - Is the interpreter waiting for a new command or for a closing quote/comment?
- Line numbers in error reporting are not particularly helpful
- The error recovery strategy can be (and it is) different

CoCoA 4 vs CoCoA 5



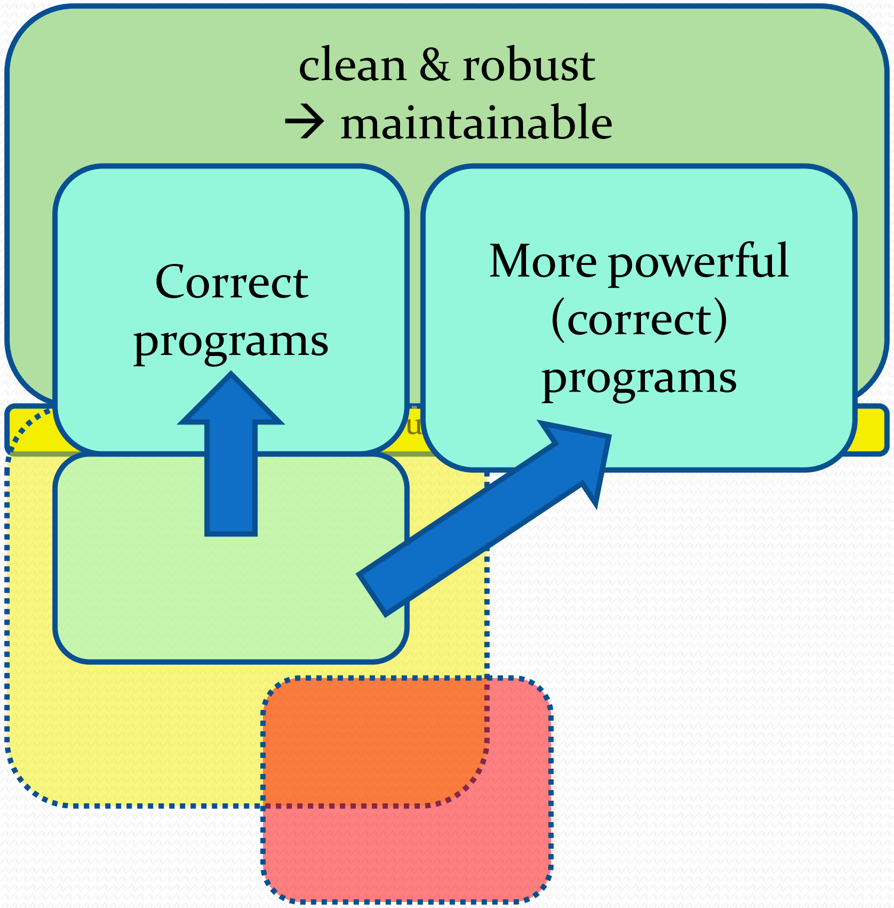
Legal programs

clean & robust
→ maintainable

Correct programs

fr
→ unm

Beginners' common mistakes



clean & robust
→ maintainable

Correct programs

More powerful (correct) programs

u

The transition path

- We’re writing a document with the (very original) title: **Differences between CoCoA-4 and CoCoA-5**
- Today I’ll give you the idea
  - Details are (or should be there)

Identifiers and Keywords

- Only *one* namespace: when you see a name, you know it can only refer to one entity (at a time)
- No special casing
- Reserved words
  - are actually *reserved*
    - Most of them are the same they were before
  - Case insensitive (yet, there are preferred casing); note that ciao is a single reserved keyword (it's not c*i*a*o)

Removed features

- *Implicit multiplication* except inside ${ … }$
- *Cond* expressions
- *Time* expression (but there is now a *Time* statement)
- “functions” *Print*/*PrintLn*
- the @ operator
- *NewLine*
- trailing *If*
- *Repeat*/*EndRepeat*
- *Help* and *Eof*

Conclusions

- We can’t forget the large user base: a smooth transition path is provided
- Every correct CoCoA 4 program will be either:
  - accepted and have the exact same semantics
  - rejected (the interpreter will tell you why)
- Restrictions are not artificial: every “clean” CoCoA 4 code should run fine
  - Once polynomials (using implicit multiplication) are parenthesized