KOMPUTER-ALGEBRA RENDSZEREK VERIFIKÁCIÓJA

Szoftver Verifikáció és Validáció, 2015 Ősz

Vaitkus Márton

Tartalom

- Motiváció
- Maple → MiniMaple
- MiniMaple típusellenőrzése
- MiniMaple formális specifikációja
- MiniMaple verifikációja Why3-ben

Motiváció

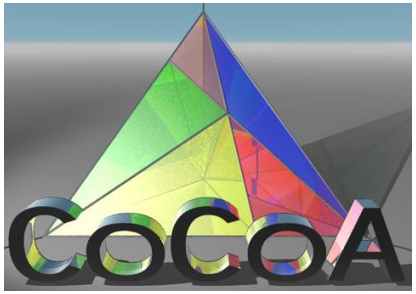
- Komputer-algebra rendszerek
  - Szimbolikus számítások automatizálása
    - Egyenletrendezés, egyszerűsítés
    - Paraméteres egyenletmegoldás (algebrai/ODE/PDE)
    - Deriválás, integrálás
    - Absztrakt algebra, számelmélet
    - Vizualizáció, interaktív ábrák
    - Stb.
  - Kezdetek: 60-as évek, LISP-alapon
    - MATHLAB, muMATH, Reduce, Derive, Macsyma (Maxima)

Motiváció

- Komputer-algebra rendszerek - kommerciális

Motiváció

- Komputer-algebra rendszerek - akadémiai



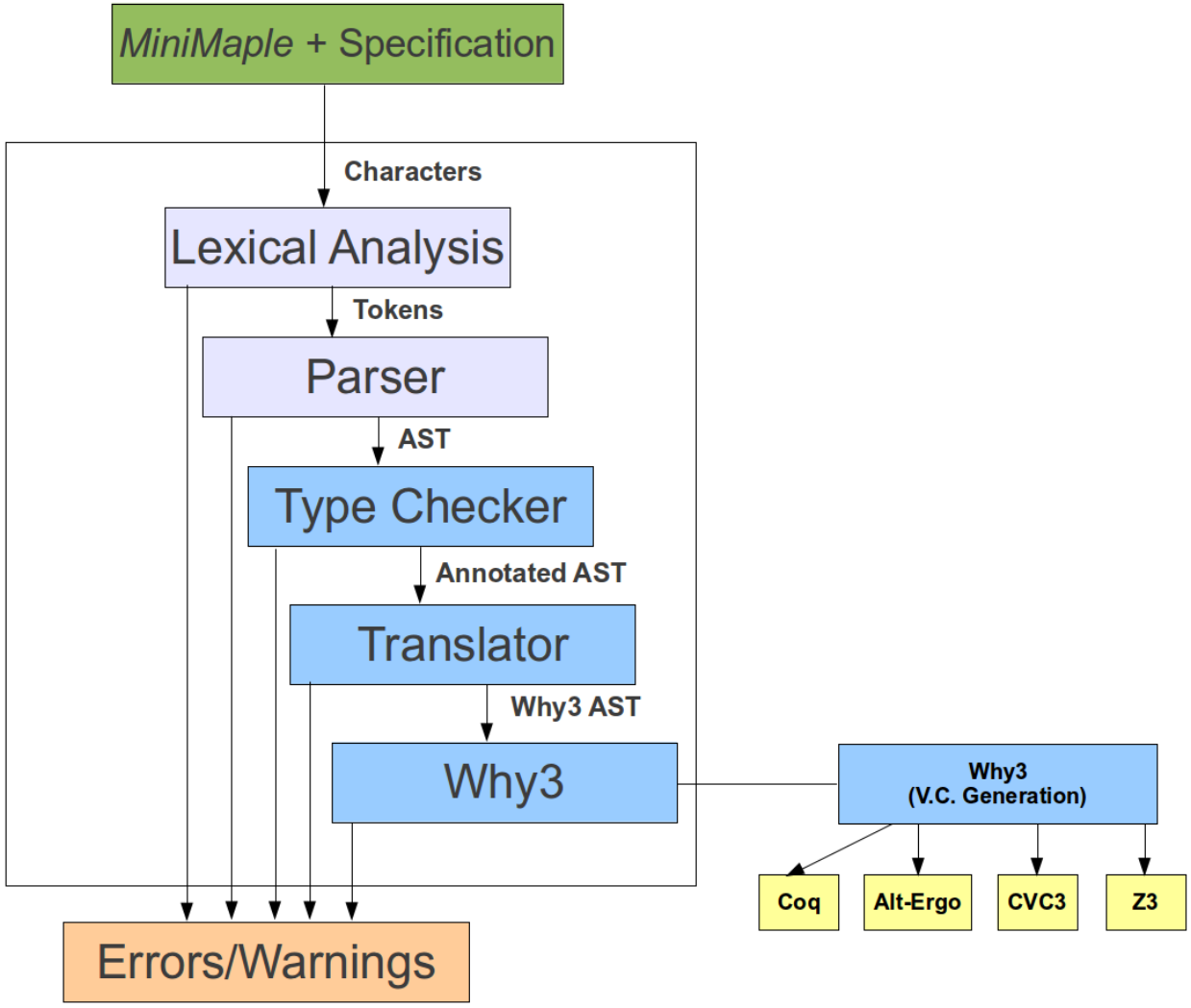
Motiváció

- Muhammad T. Khan: Formal Specification of Computer Algebra Software, (2014). PhD Disszertáció, JKU Linz, Konzulens: W. Schreiner.
  - Maple programok formális specifikációja és verifikációja
  - Maple → MiniMaple
  - MiniMaple típusellenőrző
  - MiniMaple formális specifikációja
  - MiniMaple formális szemantikája
  - „DifferenceDifferential” csomag formális verifikálása: Why3 (+ Coq, Alt-Ergo, Z3)

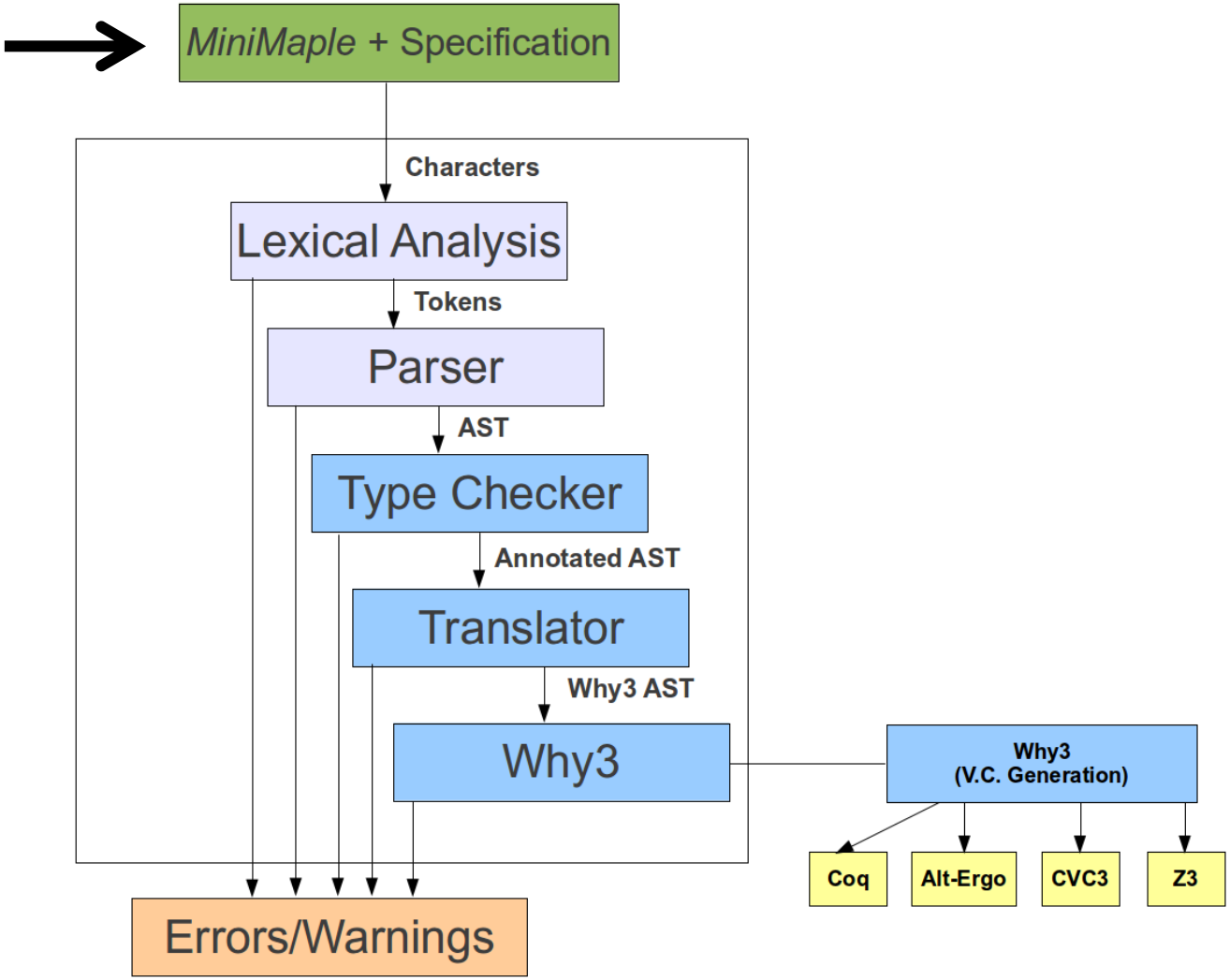
Maple

- Imperatív programnyelv + könyvtárak + IDE
- Pascal nyomán
- Kezdetek: 1980 – University of Waterloo
- 1988-tól kommerciális
- Több mint 5000 beépített függvény:
  - szimbolikus algebra
  - numerikus számítások, optimalizálás
  - absztrakt algebra (csoportelmélet, kommutatív algebra)
  - vizualizáció
- Interfész számos más nyelvhez: MATLAB, Fortran, Excel, VB, C, C#, Java, Python, Haskell, …

Maple V&V - Áttekintés



MiniMaple


MiniMaple + Specification
Characters
Lexical Analysis
Tokens
Parser
AST
Type Checker
Annotated AST
Translator
Why3 AST
Why3
Why3
(V.C. Generation)
Coq
Alt-Ergo
CVC3
Z3
Errors/Warnings

MiniMaple

- Khan munkája
- A Maple nyelv egy leszűkített részhalmaza
- *DifferenceDifferential* [Dönch, 2009] csomaghoz igazodva
- Nyelvtan formálisan specifikálva (BNF)
- Parser automatikusan generálva *ANTLR* segítségével

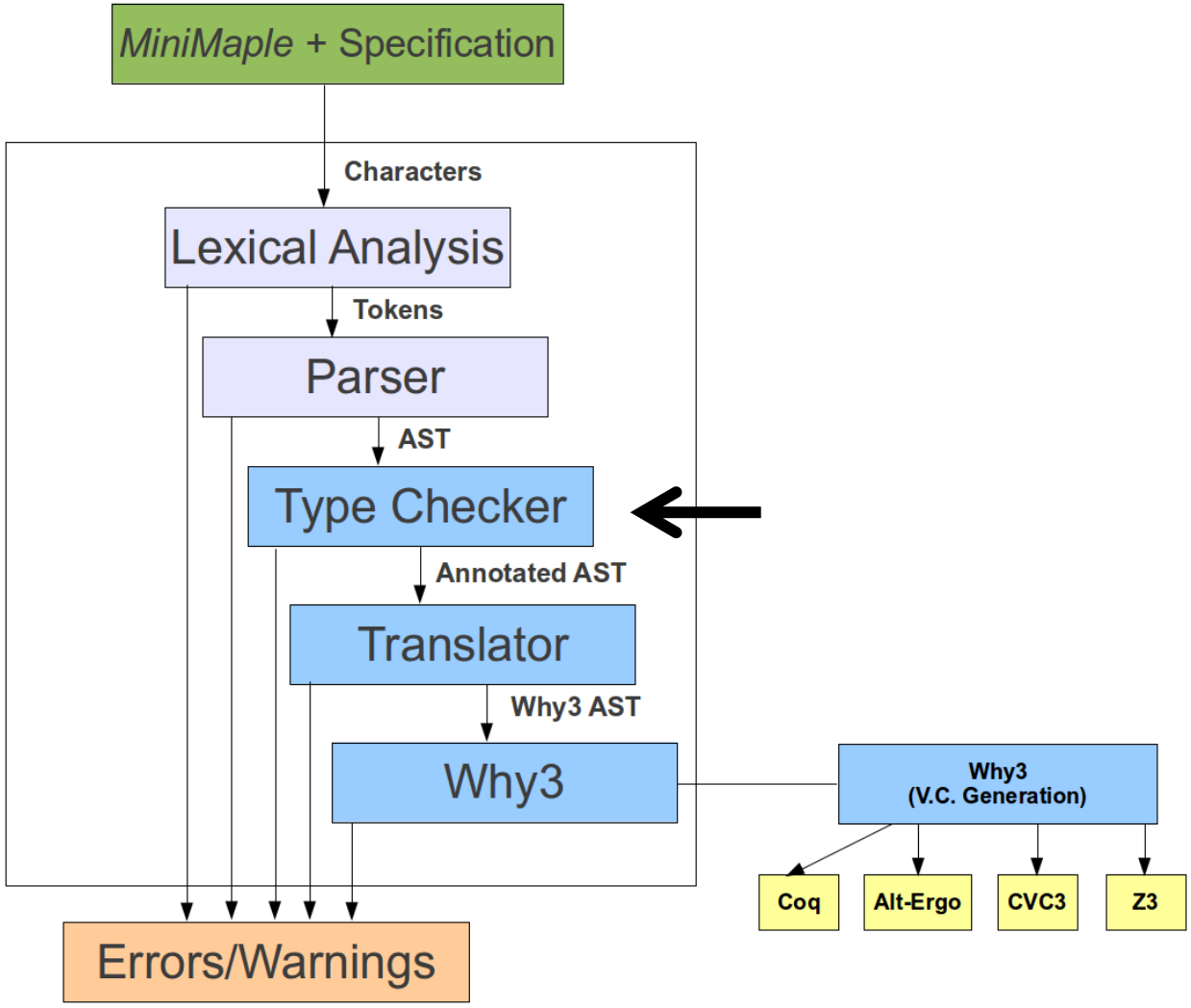
MiniMaple - BNF

Prog ::= Cseq
Cseq ::= EMPTY | C;Cseq
C ::= **if** E **then** Cseq Elif **end if** | **if** E **then** Cseq Elif **else** Cseq **end if**
| **while** E **do** Cseq **end do**
| **for** I **in** E **do** Cseq **end do**
| **for** I **in** E **while** E **do** Cseq **end do**
| **for** I **from** E **by** E **to** E **do** Cseq **end do**
| **for** I **from** E **by** E **to** E **while** E **do** Cseq **end do**
| **return** E; | **return**; | **error** | **error** I,Eseq
| **try** Cseq Catch **end** | **try** Cseq Catch **finally** Cseq **end**
| I,Iseq := E,Eseq | E(Eseq) | **'type/I'** := T
...
Eseq ::= EMPTY | E,Eseq
E ::= I | N | **module()** S;R **end module**;
| **proc**(Pseq) S;R **end proc**;| **proc**(Pseq)::T; S;R **end proc**;
| E1 Bop E2 | Uop E | Esop | E1 **and** E2 | E1 **or** E2 | E(Eseq)
| I1:-I2 | E,E,Eseq | **type**(I,T) | E1 = E2 | E1 <> E2
S ::= EMPTY | **local** It,Itseq;S | **global** I,Iseq;S | **uses** I,Iseq;S
| **export** It,Itseq;S
R ::= Cseq | Cseq;E
...

MiniMaple - Példaprogram

```
1.  status:=0;
2.  sum := proc(l::list(Or(integer,float)))::[integer,float];
3.          global status;
4.          local i, x::Or(integer,float), si::integer:=0, sf::float:=0.0;
5.          for i from 1 by 1 to nops(l) do
6.              x:=l[i];
7.              status:=i;
8.              if type(x,integer) then
9.                      if (x = 0) then
10.                         return [si,sf];
11.                     end if;
12.                     si:=si+x;
13.             elif type(x,float) then
14.                      if (x < 0.5) then
15.                         return [si,sf];
16.                     end if;
17.                     sf:=sf+x;
18.             end if;
19.         end do;
20.         status:=-1;
21.         return [si,sf];
22.         end proc;
```

MiniMaple statikus típusellenőrzése



MiniMaple statikus típusellenőrzése

- Dinamikusan típusos nyelv – statikus típusellenőrzés??
  - [Fritzon, 1997]: Statikus típusellenőzés Mathematica-hoz
  - [Monagan, 1993]: *Gauss* csomag – paraméterezett típusok, absztrakt típusok (futásidőben) → *domains* csomag
  - [Carette-Kucera, 2007]: Részleges kiértékelés

MiniMaple statikus típusellenőrzése

- Kihívások statikus analízis szempontjából:
  - Dinamikusan típusos nyelv - nincs statikus típusrendszer!
  - Nem-standard típusok: szimbólumok, kiértékeletlen kifejezések, stb.!
  - Deklaráció ~ értékadás!
  - Típusinformáció felhasználható elágazáshoz!
  - Polimorf típusrendszer – típushierarchia!

MiniMaple típusrendszere

- A Maple típusannotációit használjuk statikus típusellenőrzésre

$$
\begin{aligned}
T ::= \ & \textbf{integer} \mid \textbf{boolean} \mid \textbf{string} \mid \textbf{float} \mid \textbf{rational} \mid \textbf{anything} \\
& \mid \{\ T\ \} \mid \textbf{list}(\ T\ ) \mid [\ Tseq\ ] \mid \textbf{procedure}[\ T\ ](\ Tseq\ ) \\
& \mid I(\ Tseq\ ) \mid \textbf{Or}(\ Tseq\ ) \mid \textbf{symbol} \mid \textbf{void} \mid \textbf{uneval} \mid I
\end{aligned}
$$

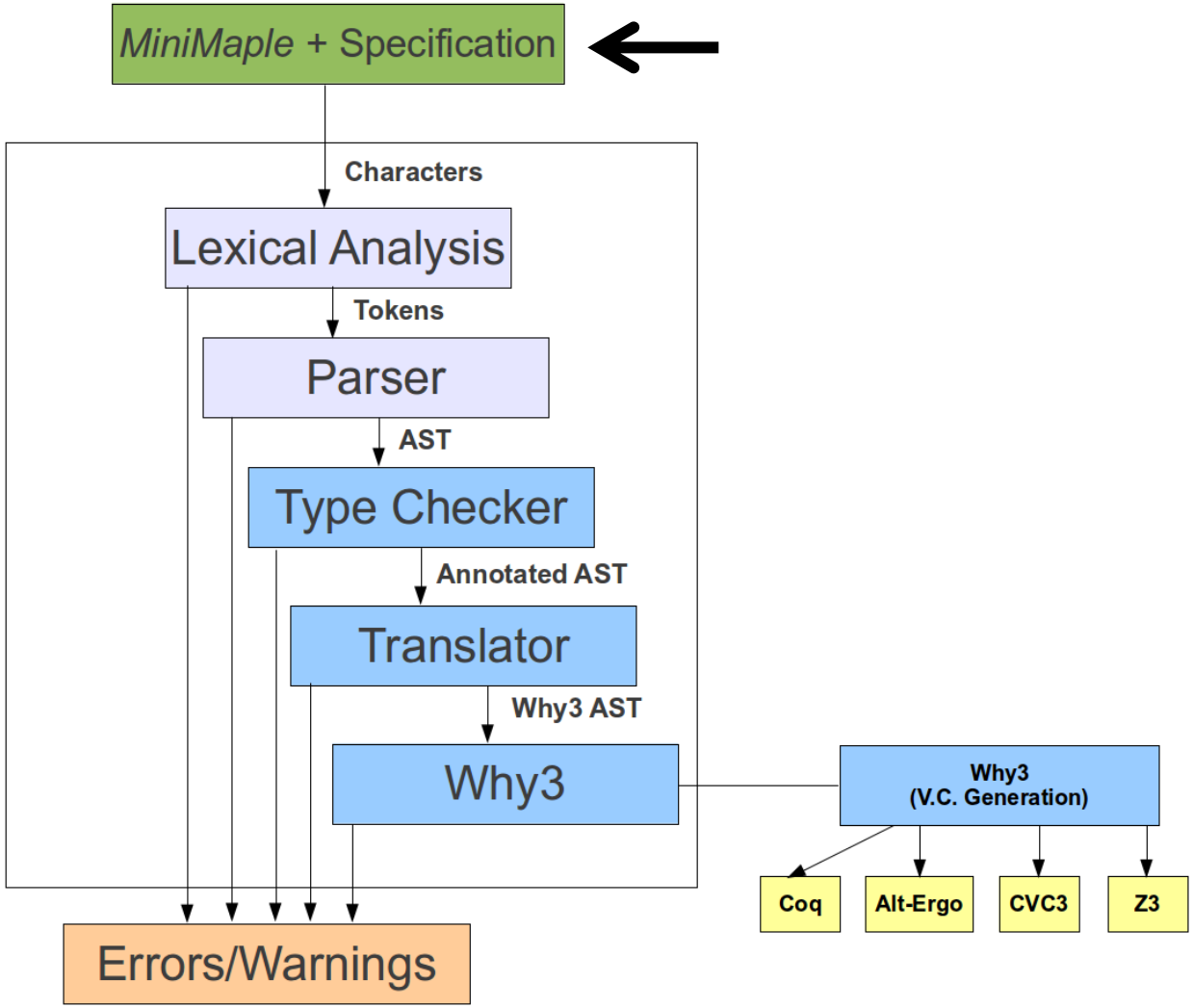
- Típus hierarchia: pl. integer < rational < … < anything
- Típusteszt: pl. `if type(x, integer) then …`
  - Típusannotációnál figyelembe kell venni (ld. a disszertációt)
- Külön `global` és `local` kontextus:
  - `global`: új változót értékadással hozzuk létre, típusa futásidőben változhat
  - `local`: új változót csak deklaráljuk, típusát csak specializálni lehet a típushierarchia szerint

MiniMaple típusellenőrzése

- Parser által generált AST alapján
- Khan által írt Java program: 100 osztály, 10K LOC

```
/home/taimoor/antlr3/Test6.m parsed with no errors.
Generating Annotated AST...
...
**********COMMAND-SEQUENCE-ANNOTATION START**********
PI -> [
sum:procedure[[integer,float]](list(Or(integer,float)))
status:integer
result:[integer,float]
]
RetTypeSet -> {}
ThrownExceptionSet -> {}
RetFlag -> not_aret
**********COMMAND-SEQUENCE-ANNOTATION END**********
Annotated AST generated.
The program type-checked correctly.
```

MiniMaple formális specifikációja



MiniMaple formális specifikációja

- Formális specifikációs nyelv Maple-hez
  - Elsőrendű logika
  - Maple szintaxis kiegészítése
- Eljárás specifikálása:

$$\begin{aligned} \textit{proc-spec} ::= \ & \textbf{requires } \textit{spec-expr}; \\ & \textbf{global } \textit{Iseq}; \\ & \textbf{ensures } \textit{spec-expr}; \ \textit{excep-clause} \end{aligned}$$

- Ciklus specifikálása:

$$\textit{loop-spec} := \textbf{invariant } \textit{spec-expr}; \ \textbf{decreases } \textit{spec-expr};$$

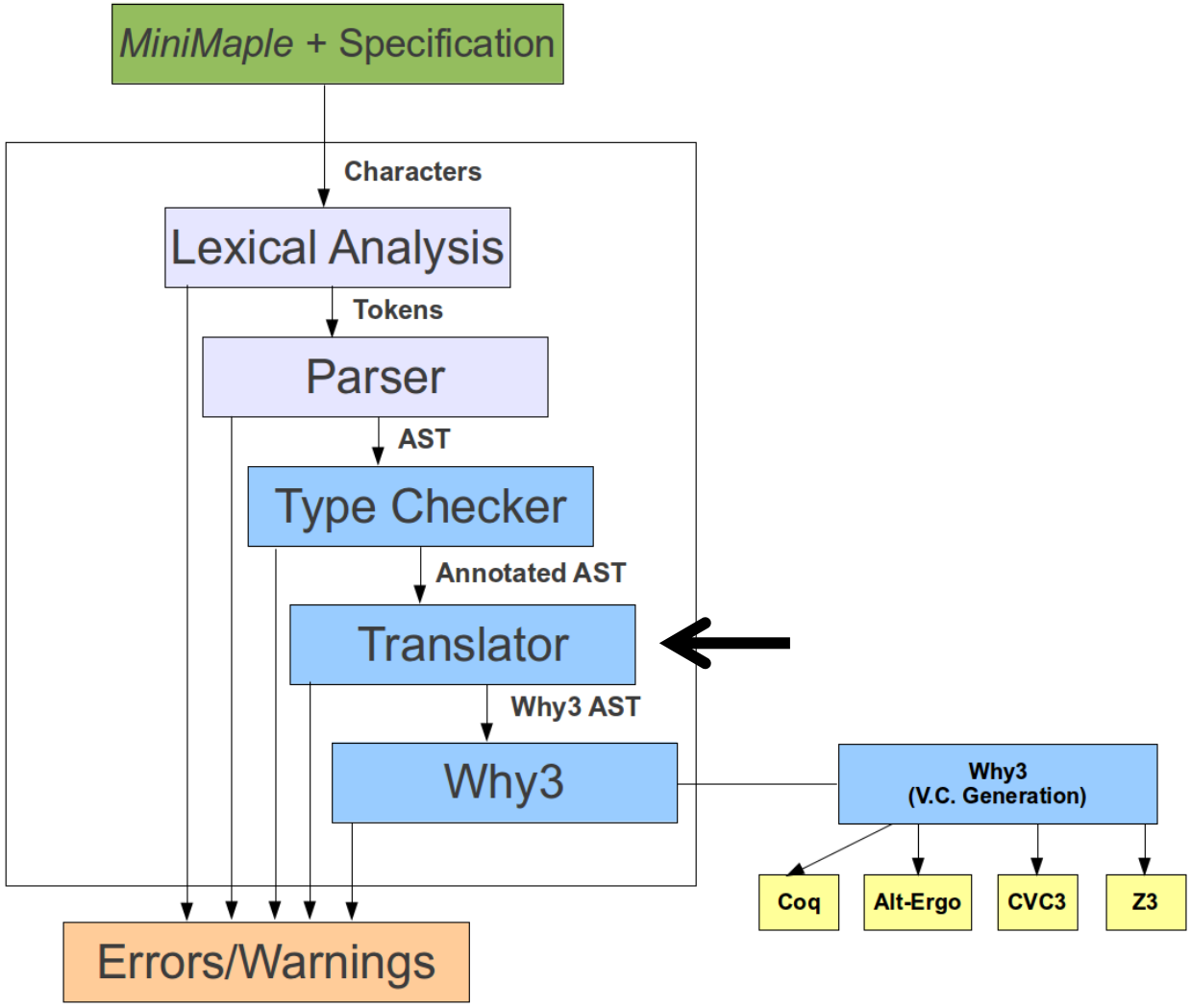
- Típus annotált specifikált program típushelyessége ellenőrizhető, mint előbb!

MiniMaple formális specifikációja

- Példaprogram:

```
status:=0;
sum := proc(l::list(Or(integer,float)))::[integer,float];
(*@
requires true;
global status;
ensures
(status = -1 and RESULT[1] = add(e, e in l, type(e,integer))
and RESULT[2] = add(e, e in l, type(e,float))
and forall(i::integer, 1<=i and i<=nops(l) and type(l[i],integer) implies l[i]<>0)
and forall(i::integer, 1<=i and i<=nops(l) and type(l[i],float) implies l[i]>=0.5))
or
(1<=status and status<=nops(l)
and RESULT[1] = add(l[i], i=1..status-1, type(l[i],integer))
and RESULT[2] = add(l[i], i=1..status-1, type(l[i],float))
and ((type(l[status],integer) and l[status]=0)
or (type(l[status],float) and l[status]<0.5))
and forall(i::integer, 1<=i and i<status and type(l[i],integer) implies l[i]<>0)
and forall(i::integer, 1<=i and i<status and type(l[i],float) implies l[i]>=0.5));
@*)
global status;
local i,x::Or(integer,float), si::integer:=0, sf::float:=0.0;
for i from 1 by 1 to nops(l) do
(*@
invariant (status <= i and
        (si = add(l[j], j=1..status, type(l[j],integer)) and
        sf = add(l[j], j=1..status, type(l[j],float)) and
          forall(i0::integer, 0 <= i0 and i0 <= status and type(l[i0],integer)
                                          implies l[i0]<>0) and
        forall(i0::integer, 0 <= i0 and i0 <= status and type(l[i0],float)
                                          implies l[i0]>=0.5)
        )
        or
        ( si = add(l[j], j=1..status-1, type(l[j],integer)) and
        sf = add(l[j], j=1..status-1, type(l[j],float)) and
        ((type(l[status],integer) and l[status]=0)
                or (type(l[status],float) and l[status]<0.5)) and
          forall(i0::integer, 0 <= i0 and i0 <= status and type(l[i0],integer)
                                          implies l[i0]<>0) and
        forall(i0::integer, 0 <= i0 and i0 <= status and type(l[i0],float)
                                          implies l[i0]>=0.5)
        );
decreases (nops(l) + 1 - i);
@*)
        x:=l[i];
        status:=i;
        if type(x,integer) then
                if (x = 0) then
                        return [si,sf];
                end if;
                si:=si+x;
        elif type(x,float) then
                if (x < 0.5) then
                        return [si,sf];
                end if;
                sf:=sf+x;
        end if;
end do;
status:=-1;
return [si,sf];
end proc;
```

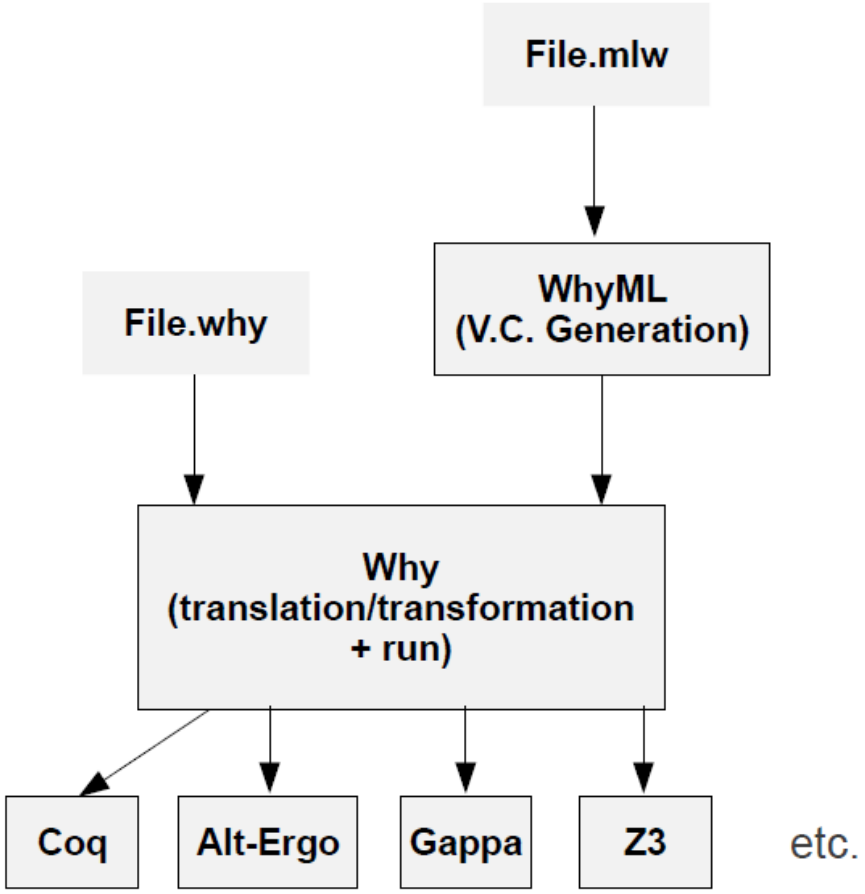
Why3


MiniMaple + Specification
Characters
Lexical Analysis
Tokens
Parser
AST
Type Checker
Annotated AST
Translator
Why3 AST
Why3
Why3
(V.C. Generation)
Coq
Alt-Ergo
CVC3
Z3
Errors/Warnings

Why3

- Verifikációs eszköz a Why3ML nyelvhez
- INRIA/LRI/CRNS (Franciao.)
- Why3ML: funkcionális, ML-alapú nyelv
- Why3ML kód → verifikációs feltételek
- Verifikációs feltételek → Why-kód: logikai specifikációs nyelv
- Why-kód → Tételbizonyítók/ SMT megoldók
  - Coq
  - Alt-Ergo
  - Z3
  - Stb.

Why3


File.mlw
WhyML
(V.C. Generation)
File.why
Why
(translation/transformation
+ run)
Coq
Alt-Ergo
Gappa
Z3
etc.

MiniMaple formális szemantikája

- Why3ML-re való fordításhoz definiálni kell a szemantikát!
- Nincs formálisan definiált szemantika Maple-hez – az implementációt kell alapul venni.
- Jellegzetességek:
  - Vannak mellékhatások (nem funkcionális jellegű nyelv!)
  - Denotációs szemantika
  - Statikus scoping
- Részleteket ld. a disszertációban!

MiniMaple → Why3ML

- MiniMaple fordítása Why3ML-re – kihívások:
  - MiniMaple-ben van return, Why3ML-ben nincs!
    - Megoldható kivételkezeléssel
  - Why3ML csak néhány típust (integer, real, string, tuple, list) támogat
    - Minden MiniMaple típus axiomatizálható
  - MiniMaple támogat egy Unió-típust (Or)
    - Pattern matchinggel implementálható
- Fordítás helyessége rigorózusan bizonyított – ld. A disszertációt!
- Khan implementációja: Java-ban – 80+ osztály, 5K+ LoC

MiniMaple fordítása Why3ML-re

```
theory SumList

  use export int.Int
  use export real.RealInfix
  use export list.List
  use export list.Length
  use export list.Nth

  type or_integer_float = Integer int | Real real

  (* sum integers among the first j elements of e *)
  function add_int (e: list or_integer_float) (j: int) : int =
    if j <= 0 then 0 else
    match e with
    | Nil -> 0
    | Cons (Integer n) t -> n + add_int t (j-1)
    | Cons _ t -> add_int t (j-1)
    end

  (* sum reals among the first j elements of e *)
  function add_real (e: list or_integer_float) (j: int) : real =
    if j <= 0 then 0.0 else
    match e with
    | Nil -> 0.0
    | Cons (Real x) t -> x +. add_real t (j-1)
    | Cons _ t -> add_real t (j-1)
    end

end

module SumListImpl

  use import SumList
  use import module ref.Ref

  val status: ref int

  exception Break

  val get (n: int) (l: list 'a) :
    { 0 <= n < length l } 'a { nth n l = Some result }

  let sum (l: list or_integer_float) : (int, real) =
    { true }
    status := 0;
    let si = ref 0 in
    let sf = ref 0.0 in
    try
      for i = 0 to length l - 1 do
        invariant { ( i = 0 /\ !status = 0 /\ !si = 0 /\ !sf = 0.0   )
                    \/
                  ( i > 0 /\ !status = i-1 /\
                         forall j: int. 0 <= j <= !status ->
                         match nth j l with
                         | None -> false
                         | Some y -> match y with
                                 | Integer n -> n <> 0
                                 | Real r -> r >=. 0.5
                             end
                         end /\
                    !si = add_int l (!status + 1) /\
                    !sf = add_real l (!status + 1))
        }
        status := i;
        match get i l with
        | Integer n -> if n = 0 then raise Break; si := !si + n
        | Real r -> if r <. 0.5 then raise Break; sf := !sf +. r
        end
      done;
      status := -1;
      (!si, !sf)
    with Break ->
      (!si, !sf)
    end
    { let (si, sf) = result in
        ( !status = -1 /\
            forall j: int. 0 <= j < (length l) ->
            match nth j l with
            | None -> false
            | Some y -> match y with
                    | Integer n -> n <> 0
                    | Real r -> r >=. 0.5
                    end
            end /\
            si = add_int l (length l) /\ sf = add_real l (length l) )
        \/
        ( 0 <= !status < length l /\
                  match nth !status l with
                  | None -> false
                  | Some y -> match y with
                          | Integer n -> n = 0
                          | Real r -> r <. 0.5
                      end
                  end /\
            forall j: int. 0 <= j < !status ->
            match nth j l with
            | None -> false
            | Some y -> match y with
                    | Integer n -> n <> 0
                    | Real r -> r >=. 0.5
                    end
            end /\
          si = add_int l !status /\ sf = add_real l !status )
    }

end
```

Verifikáció Why3-vel

Verifikáció Why3-vel - Példaprogram

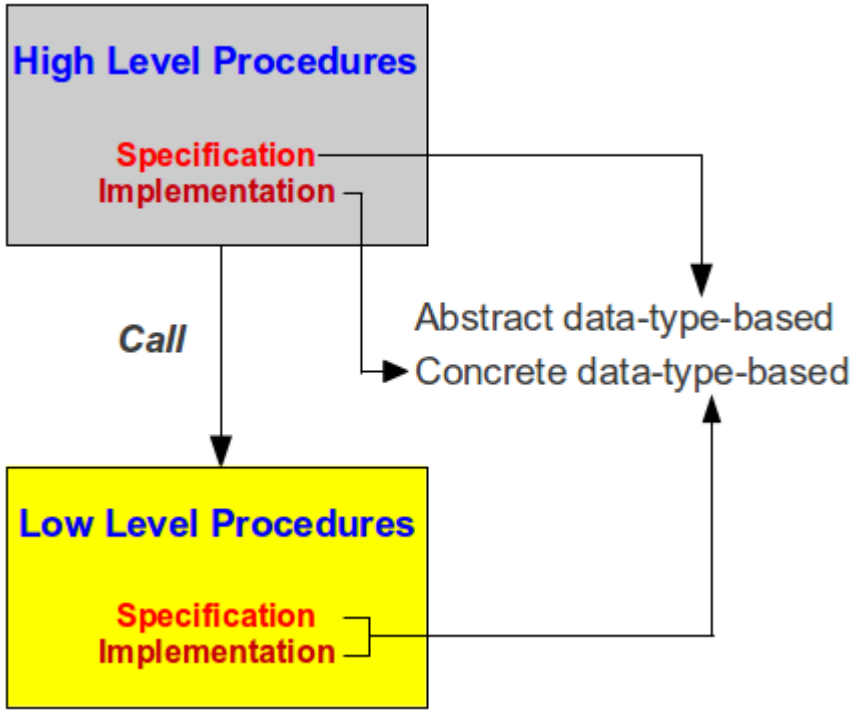
- Hiányzó lemmák (4 db)
  - Manuálisan kell hozzáadni
  - Interaktív bizonyítás Coq-kal
- Verifikáció automatikus tételbizonyítókkal
  - Alt-Ergo
  - Z3

DifferenceDifferential csomag verifikációja

- *DifferenceDifferential* Maple csomag – „Hilbert-polinom” számítása differenciális ideálokra (relatív Gröbner bázis)
- Alacsony és magas szintű eljárások
- Típusellenőrzés:
  - Annotációk hozzáadása manuálisan
  - Ellenőrző nem talált súlyos hibát, csak azonosított potenciálisan „veszélyes” kódrészeket
    - Deklarált, de nem használt változók
    - Globális és lokális scope-ban duplán deklarált változók

DifferenceDifferential csomag verifikációja

- Formális specifikáció:

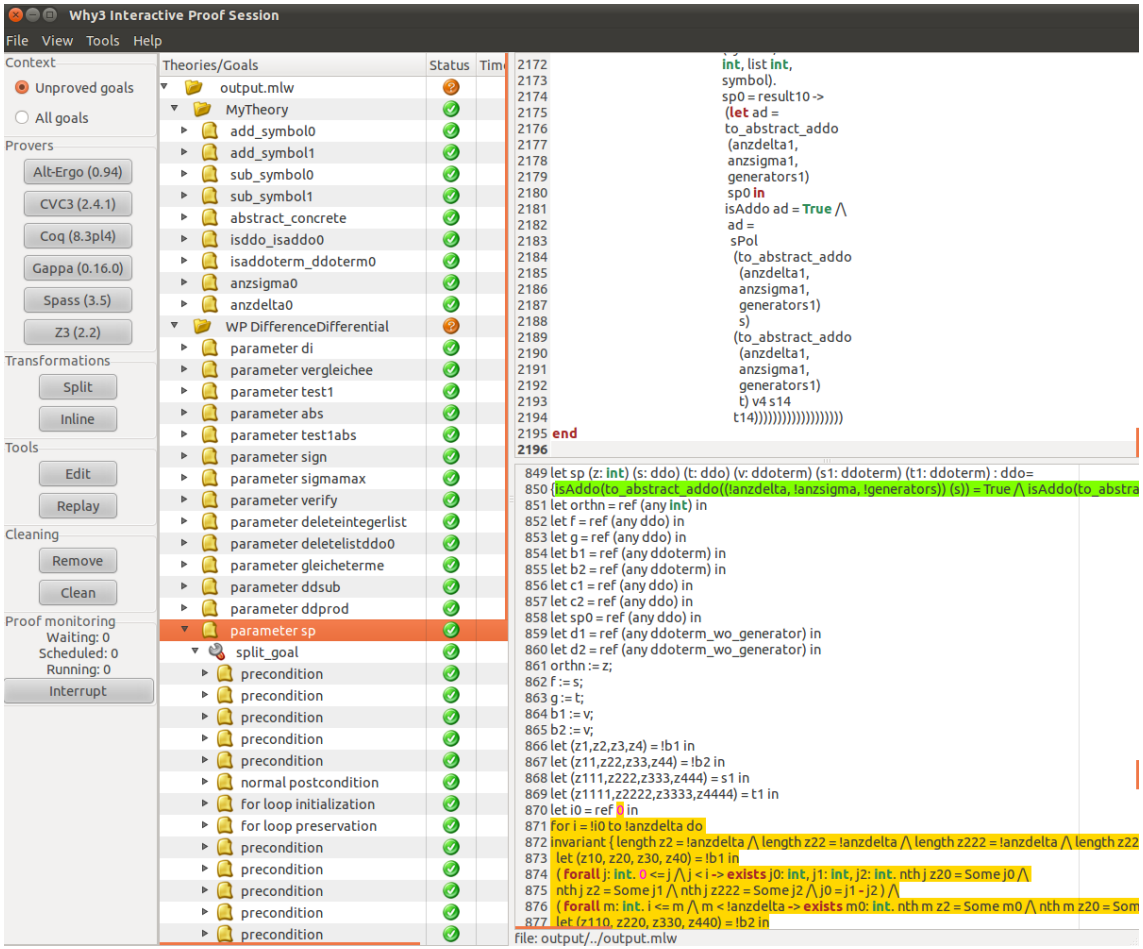


DifferenceDifferential csomag verifikációja

- Verifikáció Why3-vel:
  - 80%-ban automatikus
    - Z3
    - CVC3
    - Alt-Ergo
  - 20%-ban interaktív (alacsony szint: hiányzó lemmák, magas szint: minden)
    - Coq

DifferenceDifferential csomag verifikációja

- Why3



Összefoglalás

- Komputer-algebra programok verifikációja – nyitott probléma
- Khan: az első eredmények Maple verifikációjára
- Maple+Specifikáció → Típusellenőrzés → Why3ML kód → Verifikáció Why3-vel
- Nyitott kérdések, problémák:
  - Automatikus verifikáció – hiányzó lemmák automatikus generálása
  - Kiterjesztés Mathematica-ra – szabály alapú nyelv! Konverzió procedurális nyelvre?

  http://www.risc.jku.at/people/mtkhan/dk10/software.html

Köszönöm a figyelmet!