Good Quality of Information Streams in Modula/Oberon Programs

Ivan S. Zapreev, 2003
Institute of Informatics Systems
E-mail: cerndan@mail.ru

Project participants

Leader: Professor I.V. Pottosin

Research assistants:

- Dr. V.I. Shelekhov

Students:

- I.S. Zapreev

Institute of Informatics Systems,
Laboratory of System Programming,
Lavrentieva 6, Novosibirsk, Russia, 630090

Criteria of a program quality

- safety,
- stability,
- understandability,
- testability,
- good organization of information streams and flow of control [1],
- etc;

The aim of this work

Create algorithms that check quality of information streams in Modula-2/Oberon programs depending on [2]:

- A criteria of information streams regularity (ISR)
- A criteria of information streams confirmation (ISC)
- A stream analysis.

What is stream analysis?

The stream analysis is:

An inter-procedural, context sensitive analysis with approximation of mandatory and eventual informational relations [3] that implements an abstract interpretation of a program in the sense of **Cousot** [4]. A Single Static Assignment (SSA) form is used.

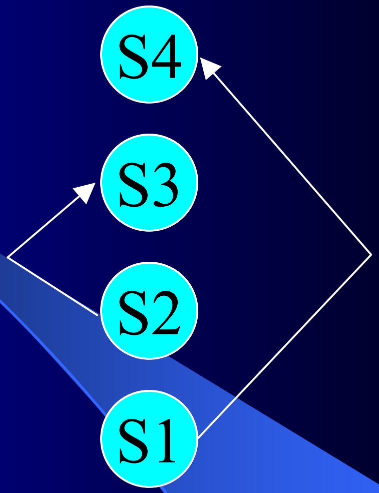
The criterion of regularity

- The ISR basing on the information stream formal definition determines what is required to avoid abnormal information streams intersection [2].
- This criterion is checked depending on information graphs of a program's linear parts.

ISR example

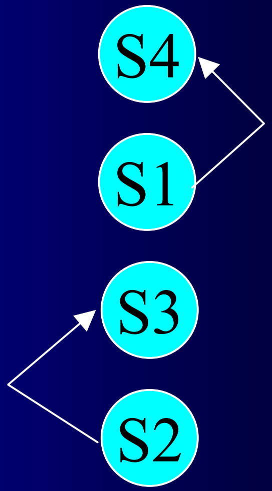
WRONG:

```
int a = 2, b = 1, c , d, e, f;
c = a + 2;          //S1
e = 4;              //S2
f = e * 3;          //S3
d = c + b;          //S4
```



CORRECT:

```
int a = 2, b = 1, c , d, e, f;
c = a + 2;          //S1
d = c + b;          //S4
e = 4;              //S2
f = e * 3;          //S3
```



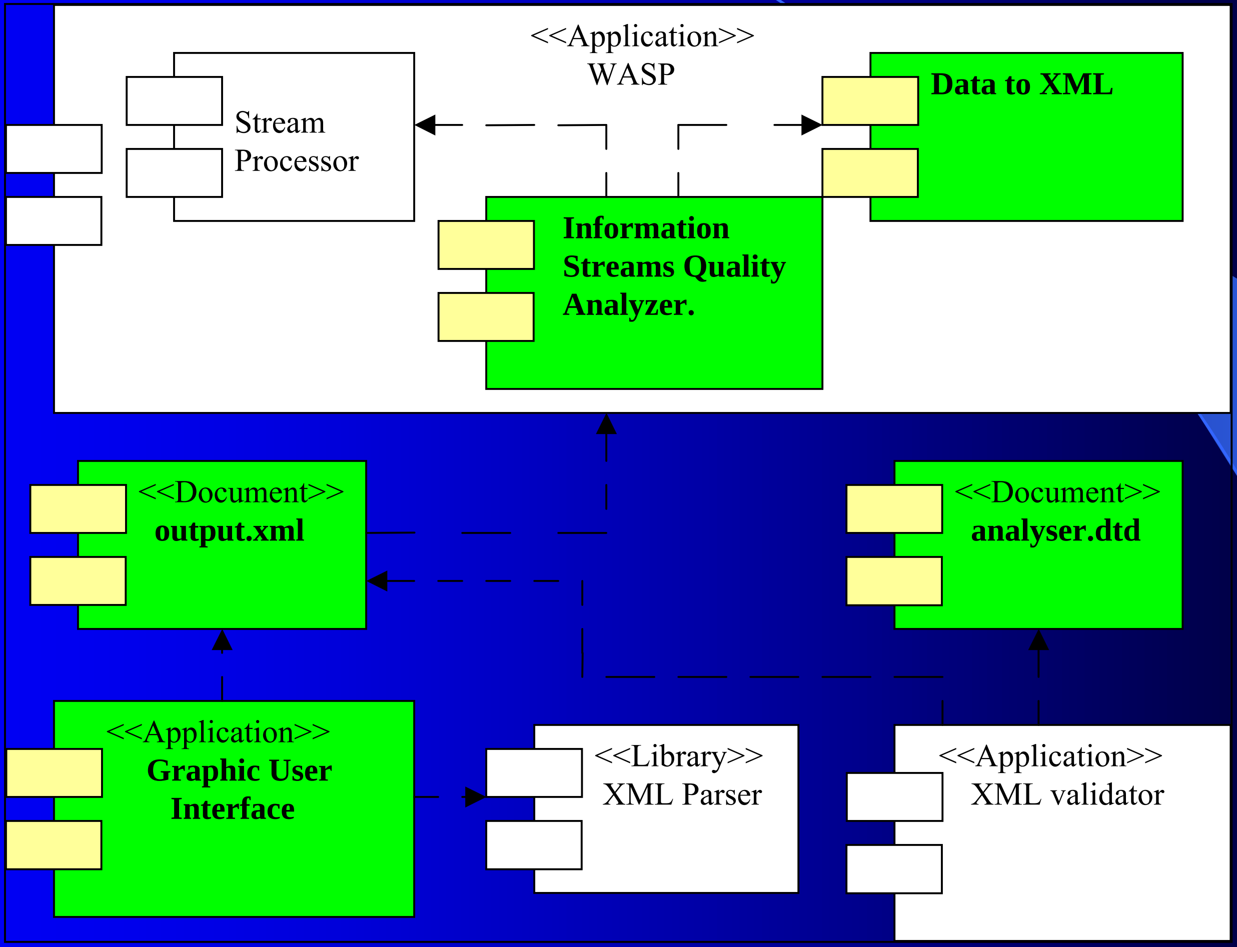
The criterion of confirmation

- The ISC basing on the operators' arguments, results and compulsory results sets determines whether there is a usage of uninitialized variable. This criterion separately examines initialization in cycles and conditional operators [2].
- This criterion is checked depending of the stream analysis results.

ISC example

```
procedure func(var smth : integer):integer;
var
    a,b:integer;
begin
    if(smth >= 0)then
        a = 1;
    else
        b = 2;
    end;
    return b;      ←Error
end func;
……
func(3);
```

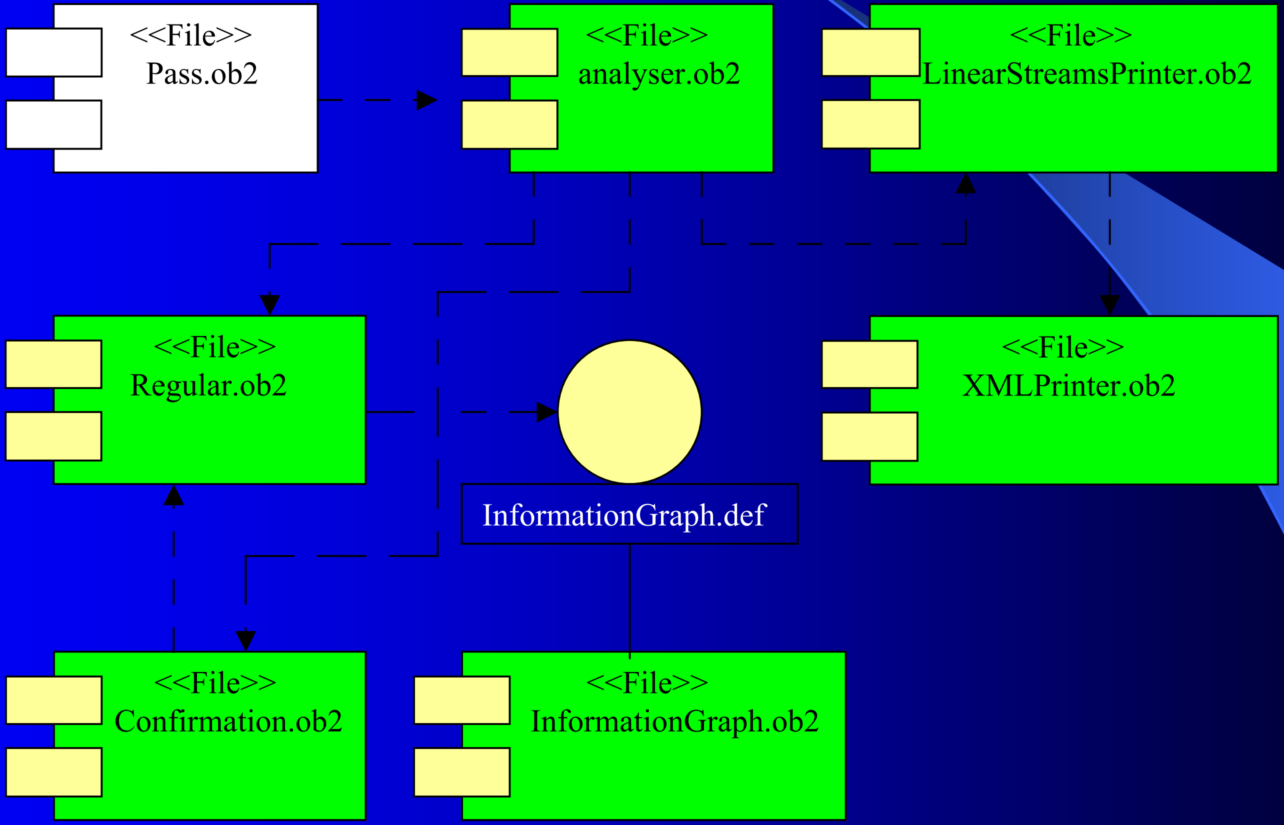
Entire system components



Implementation

- Quality analyser and storing data to XML – Modula-2/Oberon.
- *output.xml*, *analyser.dtd* – XML 1.0.
- Graphic User Interface – Java, Swing, Apache Xerces version 1.2.0

Internal architecture



The work results

- An algorithm for checking ISR and ISC on the basis of the static analysis results has been developed.
- A multifunctional Graphic User Interface has been developed.
- A mechanism for transferring of data from analyzer to visualizer has been developed on the basis on an XML standard.

References

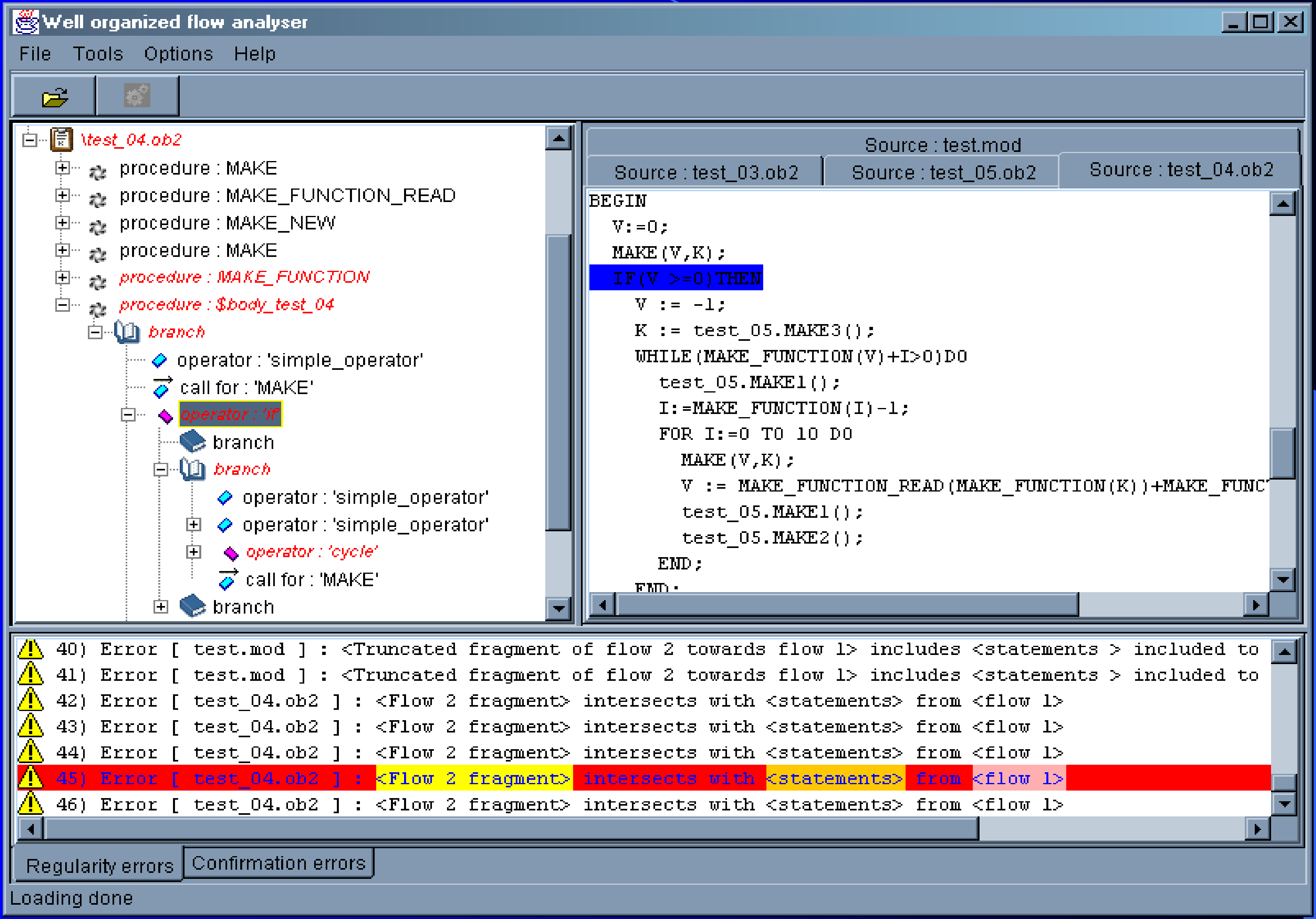
[1] Igor V. Pottosin, A "Good Program": An Attempt at Exact Definition of the Term // Programming and Computer Software, v.23, № 2 1997, p. 59-69.

[2] Igor V. Pottosin, Good quality of Programs and Information Streams // Open Systems, № 6 1998, p. 41-45.

[3] Vladimir I. Shelechov, A program structure in the language-oriented stream analysis // Programming, № 3 1996, p. 47-59.

[4] Cousot P. and Cousot R. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoint // Rec. of the 18th ACM Symposium on Principles of Programming Languages ACM Press, 1977, p.55-56.

Main Window

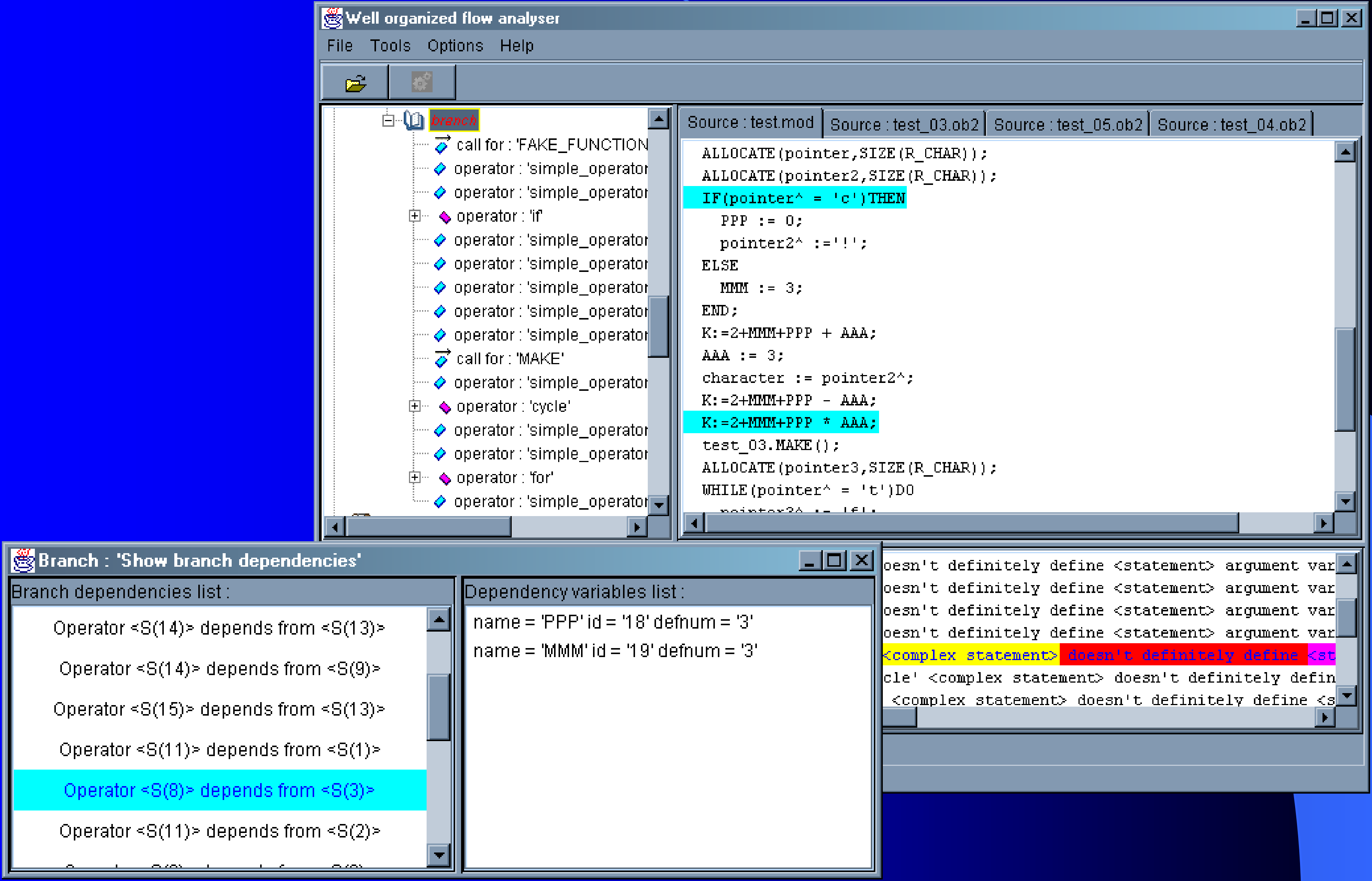


```
\test_04.ob2
    procedure : MAKE
    procedure : MAKE_FUNCTION_READ
    procedure : MAKE_NEW
    procedure : MAKE
    procedure : MAKE_FUNCTION
    procedure : $body_test_04
        branch
            operator : 'simple_operator'
            call for : 'MAKE'
            operator : 'if'
                branch
                branch
                    operator : 'simple_operator'
                    operator : 'simple_operator'
                    operator : 'cycle'
                    call for : 'MAKE'
                branch
```

```
BEGIN
  V:=0;
  MAKE(V,K);
  IF(V >=0)THEN
    V := -1;
    K := test_05.MAKE3();
    WHILE(MAKE_FUNCTION(V)+I>0)DO
      test_05.MAKE1();
      I:=MAKE_FUNCTION(I)-1;
      FOR I:=0 TO 10 DO
        MAKE(V,K);
        V := MAKE_FUNCTION_READ(MAKE_FUNCTION(K))+MAKE_FUNC
        test_05.MAKE1();
        test_05.MAKE2();
      END;
    END;
```

```
40) Error [ test.mod ] : <Truncated fragment of flow 2 towards flow 1> includes <statements > included to
41) Error [ test.mod ] : <Truncated fragment of flow 2 towards flow 1> includes <statements > included to
42) Error [ test_04.ob2 ] : <Flow 2 fragment> intersects with <statements> from <flow 1>
43) Error [ test_04.ob2 ] : <Flow 2 fragment> intersects with <statements> from <flow 1>
44) Error [ test_04.ob2 ] : <Flow 2 fragment> intersects with <statements> from <flow 1>
45) Error [ test_04.ob2 ] : <Flow 2 fragment> intersects with <statements> from <flow 1>
46) Error [ test_04.ob2 ] : <Flow 2 fragment> intersects with <statements> from <flow 1>
```

Branch dependencies



Well organized flow analyser
File
Tools
Options
Help
Branch : 'Show branch dependencies'

- branch
  - call for : 'FAKE_FUNCTION
  - operator : 'simple_operator
  - operator : 'simple_operator
  - operator : 'if'
  - operator : 'simple_operator
  - operator : 'simple_operator
  - operator : 'simple_operator
  - operator : 'simple_operator
  - operator : 'simple_operator
  - call for : 'MAKE'
  - operator : 'simple_operator
  - operator : 'cycle'
  - operator : 'simple_operator
  - operator : 'simple_operator
  - operator : 'for'
  - operator : 'simple_operator

Source : test.mod | Source : test_03.ob2 | Source : test_05.ob2 | Source : test_04.ob2

```
ALLOCATE(pointer,SIZE(R_CHAR));
ALLOCATE(pointer2,SIZE(R_CHAR));
IF(pointer^ = 'c')THEN
  PPP := 0;
  pointer2^ :='!';
ELSE
  MMM := 3;
END;
K:=2+MMM+PPP + AAA;
AAA := 3;
character := pointer2^;
K:=2+MMM+PPP - AAA;
K:=2+MMM+PPP * AAA;
test_03.MAKE();
ALLOCATE(pointer3,SIZE(R_CHAR));
WHILE(pointer^ = 't')DO
```

```
oesn't definitely define <statement> argument var
oesn't definitely define <statement> argument var
oesn't definitely define <statement> argument var
oesn't definitely define <statement> argument var
<complex statement> doesn't definitely define <st
cle' <complex statement> doesn't definitely defin
 <complex statement> doesn't definitely define <s
```

Branch dependencies list :

Operator <S(14)> depends from <S(13)>

Operator <S(14)> depends from <S(9)>

Operator <S(15)> depends from <S(13)>

Operator <S(11)> depends from <S(1)>

Operator <S(8)> depends from <S(3)>

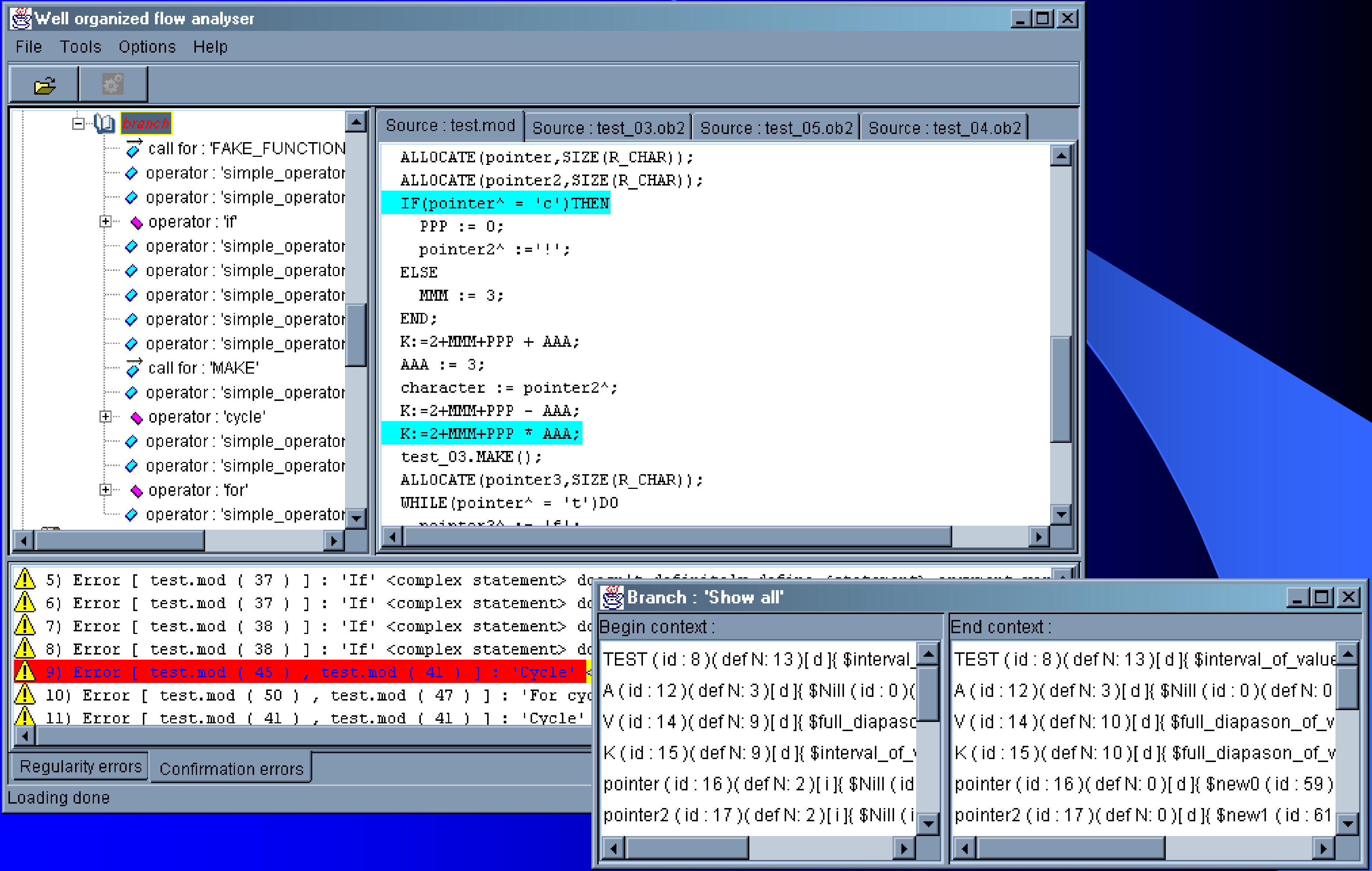
Operator <S(11)> depends from <S(2)>

Dependency variables list :

name = 'PPP' id = '18' defnum = '3'

name = 'MMM' id = '19' defnum = '3'

Branch context



```
call for : 'FAKE_FUNCTION
operator : 'simple_operator
operator : 'simple_operator
operator : 'if'
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
call for : 'MAKE'
operator : 'simple_operator
operator : 'cycle'
operator : 'simple_operator
operator : 'simple_operator
operator : 'for'
operator : 'simple_operator
```

```
ALLOCATE(pointer,SIZE(R_CHAR));
ALLOCATE(pointer2,SIZE(R_CHAR));
IF(pointer^ = 'c')THEN
  PPP := 0;
  pointer2^ :='!';
ELSE
  MMM := 3;
END;
K:=2+MMM+PPP + AAA;
AAA := 3;
character := pointer2^;
K:=2+MMM+PPP - AAA;
K:=2+MMM+PPP * AAA;
test_03.MAKE();
ALLOCATE(pointer3,SIZE(R_CHAR));
WHILE(pointer^ = 't')DO
```

```
5) Error [ test.mod ( 37 ) ] : 'If' <complex statement> do
6) Error [ test.mod ( 37 ) ] : 'If' <complex statement> do
7) Error [ test.mod ( 38 ) ] : 'If' <complex statement> do
8) Error [ test.mod ( 38 ) ] : 'If' <complex statement> do
9) Error [ test.mod ( 45 ) , test.mod ( 41 ) ] : 'Cycle'
10) Error [ test.mod ( 50 ) , test.mod ( 47 ) ] : 'For cyc
11) Error [ test.mod ( 41 ) , test.mod ( 41 ) ] : 'Cycle'
```

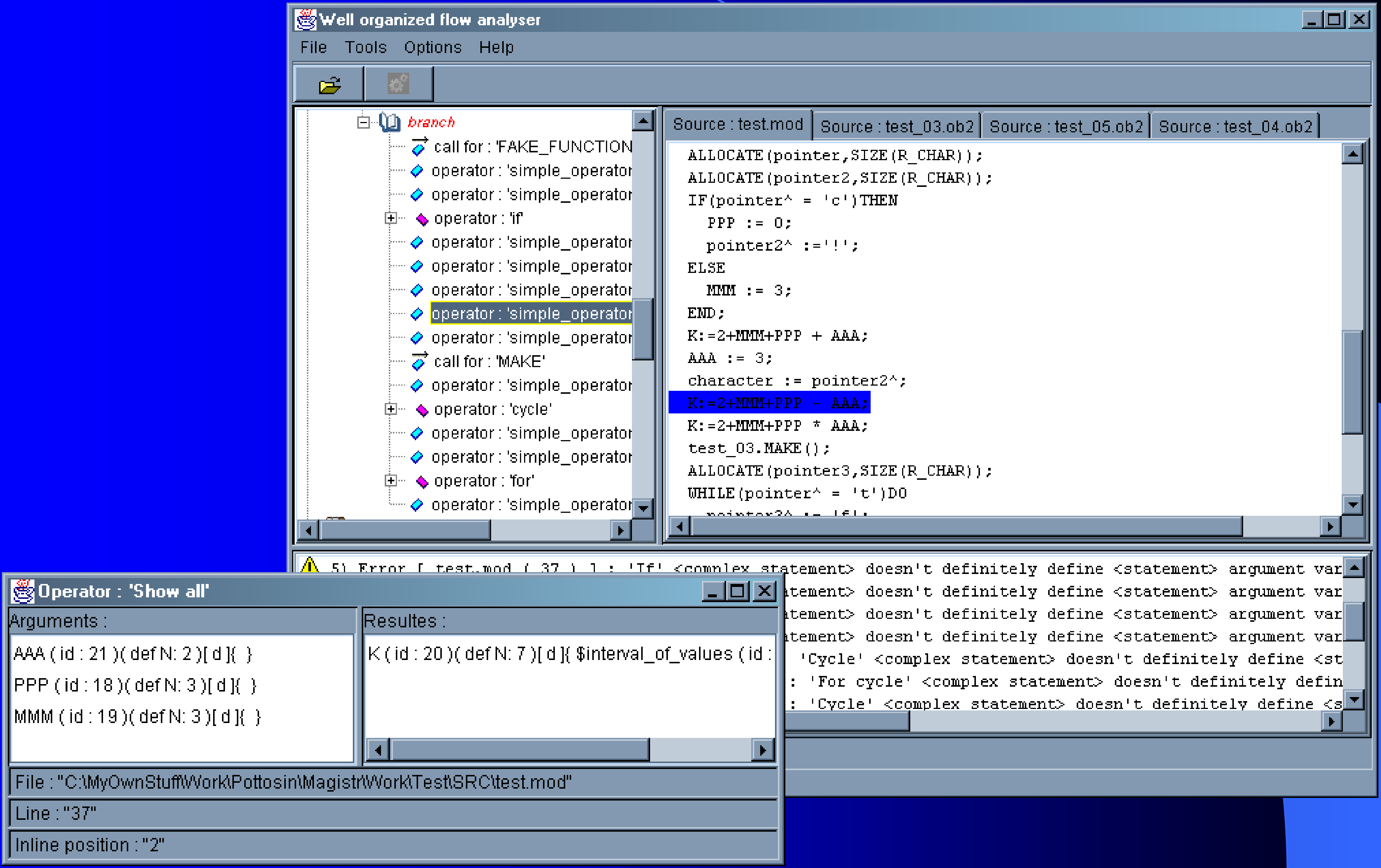
Begin context :

```
TEST ( id : 8 )( def N: 13 )[ d ]{ $interval_
A ( id : 12 )( def N: 3 )[ d ]{ $Nill ( id : 0 )(
V ( id : 14 )( def N: 9 )[ d ]{ $full_diapasc
K ( id : 15 )( def N: 9 )[ d ]{ $interval_of_
pointer ( id : 16 )( def N: 2 )[ i ]{ $Nill ( id
pointer2 ( id : 17 )( def N: 2 )[ i ]{ $Nill ( i
```

End context :

```
TEST ( id : 8 )( def N: 13 )[ d ]{ $interval_of_value
A ( id : 12 )( def N: 3 )[ d ]{ $Nill ( id : 0 )( def N: 0
V ( id : 14 )( def N: 10 )[ d ]{ $full_diapason_of_v
K ( id : 15 )( def N: 10 )[ d ]{ $full_diapason_of_v
pointer ( id : 16 )( def N: 0 )[ d ]{ $new0 ( id : 59 )
pointer2 ( id : 17 )( def N: 0 )[ d ]{ $new1 ( id : 61
```

Operator attributes



Well organized flow analyser
File
Tools
Options
Help
branch
call for : 'FAKE_FUNCTION
operator : 'simple_operator
operator : 'simple_operator
operator : 'if'
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
operator : 'simple_operator
call for : 'MAKE'
operator : 'simple_operator
operator : 'cycle'
operator : 'simple_operator
operator : 'simple_operator
operator : 'for'
operator : 'simple_operator
Source : test.mod
Source : test_03.ob2
Source : test_05.ob2
Source : test_04.ob2
Operator : 'Show all'
Arguments :
Resultes :

```
ALLOCATE(pointer,SIZE(R_CHAR));
ALLOCATE(pointer2,SIZE(R_CHAR));
IF(pointer^ = 'c')THEN
  PPP := 0;
  pointer2^ :='!';
ELSE
  MMM := 3;
END;
K:=2+MMM+PPP + AAA;
AAA := 3;
character := pointer2^;
K:=2+MMM+PPP - AAA;
K:=2+MMM+PPP * AAA;
test_03.MAKE();
ALLOCATE(pointer3,SIZE(R_CHAR));
WHILE(pointer^ = 't')DO
  pointer3^ := 'f';
```

```
5) Error [ test.mod ( 37 ) ] : 'If' <complex statement> doesn't definitely define <statement> argument var
atement> doesn't definitely define <statement> argument var
atement> doesn't definitely define <statement> argument var
atement> doesn't definitely define <statement> argument var
 'Cycle' <complex statement> doesn't definitely define <st
: 'For cycle' <complex statement> doesn't definitely defin
: 'Cycle' <complex statement> doesn't definitely define <s
```

Arguments :

AAA (id : 21)(def N: 2)[d]{ }

PPP (id : 18)(def N: 3)[d]{ }

MMM (id : 19)(def N: 3)[d]{ }

Resultes :

K (id : 20)(def N: 7)[d]{ $interval_of_values (id :

File : "C:\MyOwnStuff\Work\Pottosin\Magistr\Work\Test\SRC\test.mod"

Line : "37"

Inline position : "2"