Tool demonstration: Euclide

UMR IRISA

Benjamin Cama, A. Gotlieb,
G. Andrade-Barroso

TAP 2009, Zürich

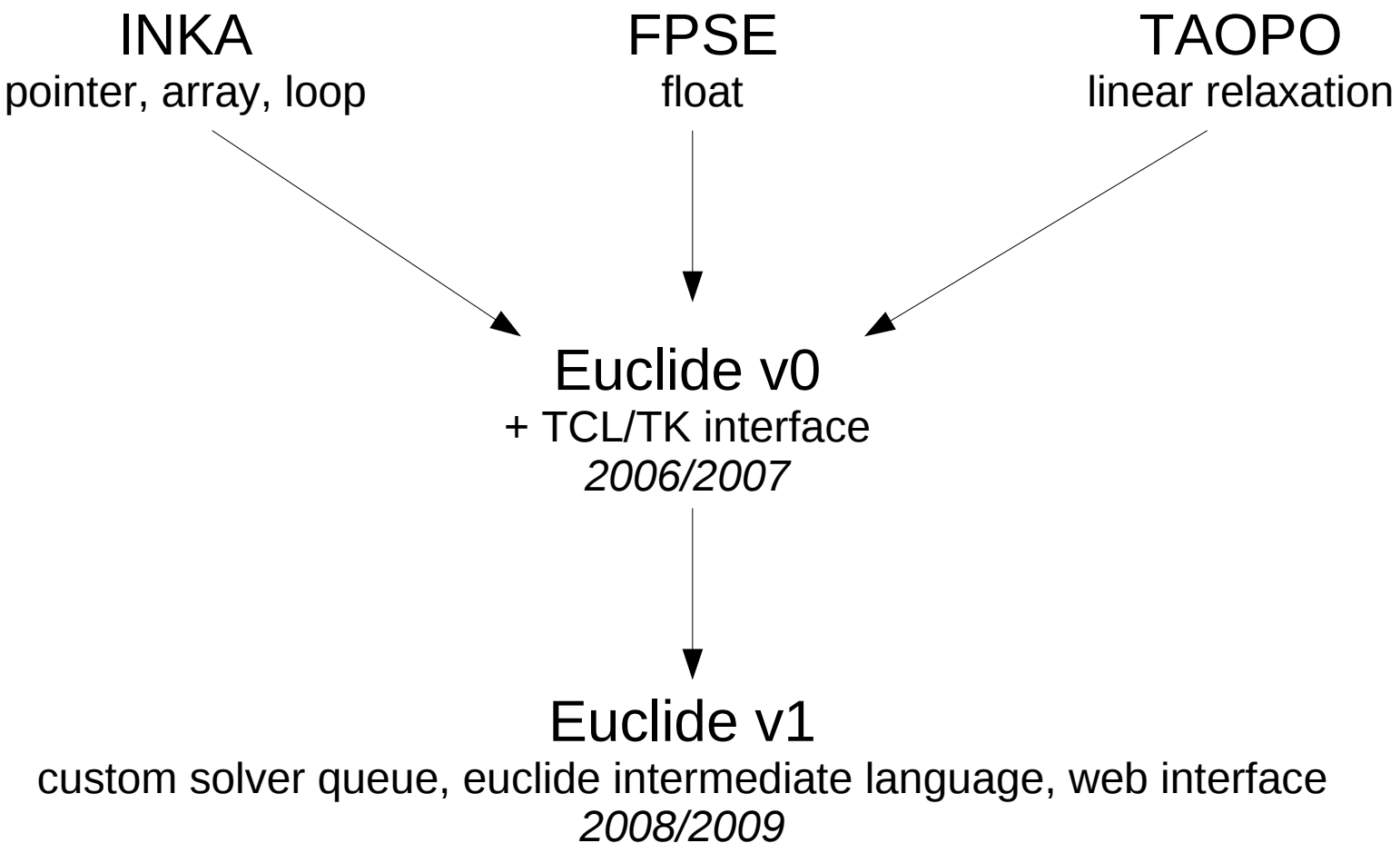
CAVERN project
ANR-07-SESUR-003

2009/07/02

Context

- **What**
  - Euclide is a constraint-based testing tool for testing and verifying critical C programs
- **How**
  - Model C programs as relations and reason on them using constraint solving techniques
- **Applications**
  - Automatic structural test data generation
  - Counter-example generation
  - Assertion checking

Project's history



Details

Techniques

- Static Single Assignement
- Constraint propagation (using CLPFD)
- Integer linear relaxation
- Goal-oriented test data generation

Implementation

- A web interface to easily demonstrate Euclide's features to interested people
- Coded in Prolog
- Licensed under CeCILL-C (LGPL-like)

Typical session

- **Small program demonstration**
  - Input your C source code

Example

```
1   int f(int i)
2   {
3     int j;
4     j = 2;
5
6     if (i <= 16)
7       j = j * i ;
8
9     if ( j > 8 )
10      {
11        j = (j - 10) / 5  ;
12        j = j * j  ;
13      }
14
15    return j;
16  }
…
42  int main(...) {...}
```

Typical session

- **Small program demonstration**
  - Input your C source code
  - Select a function to analyse

Example

```
1   int f(int i)
2   {
3     int j;
4     j = 2;
5
6     if (i <= 16)
7       j = j * i ;
8
9     if ( j > 8 )
10      {
11        j = (j - 10) / 5  ;
12        j = j * j  ;
13      }
14
15    return j;
16  }
...
42  int main(...) {...}
```

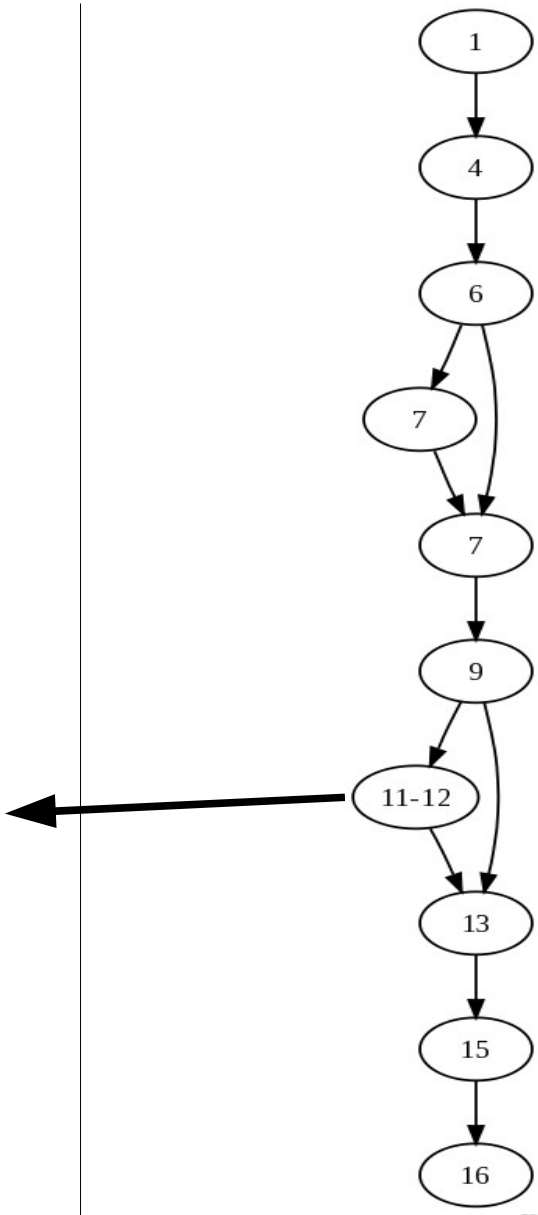
```
rel f(i_0,return_1) iff {
  j_1 = 2,
  tmp1_1 = $(i_0 =< 16),
  ite(tmp1_1,
    {
      j_2 = j_1 * i_0,
      j_3 = j_2
    },
    {
      j_3 = j_1
    })
  ,
  tmp2_1 = $(j_3 > 8),
  ite(tmp2_1,
    {
      j_4 = j_3 - 10,
      j_5 = j_4 / 5,
      j_6 = j_5 * j_5,
      j_7 = j_6
    },
    {
      j_7 = j_3
    })
  ,
  return_1 = j_7
}
```

Typical session

- **Small program demonstration**
  - Input your C source code
  - Select a function to analyse
  - Choose a goal: statement/decision reachability

Example

```
1   int f(int i)
2   {
3     int j;
4     j = 2;
5
6     if (i <= 16)
7       j = j * i ;
8
9     if ( j > 8 )
10      {
11        j = (j - 10) / 5  ;
12        j = j * j  ;
13      }
14
15    return j;
16  }
…
42  int main(...) {...}
```



Example

```
1   int f(int i)
2   {
3     int j;
4     j = 2;
5
6     if (i <= 16)
7       j = j * i ;
8
9     if ( j > 8 )
10      {
11        j = (j - 10) / 5  ;
12        j = j * j  ;
13      }
14
15    return j;
16  }
…
42  int main(...) {...}
```

```
rel f(i_0,return_1) iff {
  j_1 = 2,
  tmp1_1 = $(i_0 =< 16),
  ite(tmp1_1,
    {
      j_2 = j_1 * i_0,
      j_3 = j_2
    },
    {
      j_3 = j_1
    })
  ,
  tmp2_1 = $(j_3 > 8),
  ite(tmp2_1,
    {
      j_4 = j_3 - 10,
      j_5 = j_4 / 5,
      j_6 = j_5 * j_5,
      j_7 = j_6,
      reach
    },
    {
      j_7 = j_3
    })
  ,
  return_1 = j_7
}
```

Typical session

- **Small program demonstration**
  - Input your C source code
  - Select a function to analyse
  - Choose a goal: statement/decision reachability
  - Launch test data generation !

Example

```
1   int f(int i)
2   {
3     int j;
4     j = 2;
5
6     if (i <= 16)
7       j = j * i ;
8
9     if ( j > 8 )
10      {
11        j = (j - 10) / 5  ;
12        j = j * j  ;
13      }
14
15    return j;
16  }
…
42  int main(...) {...}
```

Line 12 is reachable

Input values (*not trivial to find*):
i = 10

You can even get the output values of the constraint model for this variables instanciation (here when i=10), as a hint:
j = 4

Typical session

- **Small program demonstration**
  - Input your C source code
  - Select a function to analyse
  - Choose a goal: statement/decision reachability
  - Launch test data generation !
- **Other possibilities**
  - Optionnaly annotate sources (assertions, ...)
  - Tweak the generation parameters (heuristic, timeout)
  - Look at variables domain range

Live demo

→ Example:

- TCAS, the Traffic Collision Avoidance System
  - Used in planes for automatic and deterministic decision taking, when some other aircraft may present a threat of mid-air collision
  - Academic implementation

Conclusion

➔ Not shown today

- Loops efficiently handled
  - A. Gotlieb, B. Botella, and M. Rueher. **A clp framework for computing structural test data**. In *Proceedings of Computational Logic* (CL'2000), LNAI 1891, pages 399–413, London, UK, July 2000.
- Interprocedural calls
  - Arnaud Gotlieb. **EUCLIDE: A Constraint-Based Testing platform for critical C programs**. In *proceedings of 2th International Conference on Software Testing, Validation and Verification* (ICST'09), April 2009.

Conclusion

➔ Limitations

- Only handle sequentialy structured programs (no goto), correctly typed, and without recursion
- No dynamic memory allocation
- No library/external calls
- Too hard problems: function pointers, pointer arithmetic

Conclusion

➔ Scheduled enhancements

- Pointer and array management
  - October 2009
  - A. Gotlieb, T. Denmat, and B. Botella. **Goal-oriented test data generation for pointer programs**. *Information and Software Technology*, 49(9-10):1030–1044, Sep. 2007.
- Floating point calculations
  - End of 2009
  - B. Botella, A. Gotlieb, and C. Michel. **Symbolic execution of floating-point computations**. *The Software Testing, Verification and Reliability journal*, 16(2):pp 97–121, June 2006.
- C structures

Contact

- **Project page:**
  - http://euclide.gforge.inria.fr/
- **Web interface demo:**
  - http://alexandrie.irisa.fr/
- **Email:**
  - benjamin.cama@irisa.fr
  - arnaud.gotlieb@irisa.fr
  - guillermo.andrade-barroso@irisa.fr

The End

➔**Thank you !**