Automated Testing&Verification

Lab Session #1 : Verification project

Juan Pablo Galeotti, Alessandra Gorla, Andreas Rau
Saarland University

Project #1 – A Small Static Verifier

- Handout date: 22.11.2012
- 3 Exercises
  - Nr. 1: Prove 3 properties using CVC3 automatic theorem prover
  - Nr. 2: Write your own translator from Pest to CVC3
  - Nr. 3: Use your translator+CVC3 to fix program specifications

CVC3

- CVC3 is an automated validity checker for a many-typed FOL with built-in theories:
  - equality over un-interpreted function symbols
  - real and integer linear arithmetic (limited non-linear)
  - bit vectors
  - arrays
  - tuples
  - records
  - user-defined inductive datatypes

CVC3: Languages

- Native CVC3 input language: Presentation
- Standardized language: SMTLIB

CVC3: Axioms & Conjectures

- Axioms

```
ASSERT F1;
ASSERT F2;
```

…

- Conjectures

```
QUERY F3;
```

- Counterexample

```
COUNTERMODEL;
```

CVC3: Arithmetics

- Constant declaration

```
i:  INT;
j:  INT;
k:  INT;
```

- Arithmetic operators:
  - +, -, *, /
  - <,>,<=,>=,/=(inequality)
  - Integer literals: 0,1,2,3,… etc.

CVC3: logical symbols

- Boolean constants
  - TRUE
  - FALSE
- Boolean operators
  - NOT, AND, OR, XOR, =>, <=>

CVC3: Quantifiers

- Existential:

```
EXISTS (z: INT): x/=z OR z/=y
```

- Universal

```
FORALL (x,y: A, i,j,k:B): i=j AND i/=k
```

CVC3: Example

```
% constants
i: INT;
j: INT;
% axioms
ASSERT FORALL (k: INT): k>i => k>j;

% conjectures
QUERY EXISTS (w:INT): w<i AND w<j;
COUNTERMODEL;
```

CVC3: Outcomes

- Valid: cvc3 was able to prove that the conjecture is always true.
- Invalid: cvc3 found a model under which the conjecture is false.
- Unknown: cvc3 was unable to prove nor disprove the conjecture

CVC3: contexts

- PUSH/POP allow us to store a stack of contexts:

```
i: INT;
PUSH;
ASSERT i>0;
...
POP;
PUSH;
ASSERT i<0;
...
POP;
```

Uninterpreted functions

```
% declaration
plusOne: INT -> INT;

% axioms
ASSERT FORALL (i: INT): plusOne(i)=i+1;
```

CVC3: Running the theorem prover

- Command line execution

```
$ cvc3 inputfile.cvc
```

CVC3: example #1

```
% constants
i: INT;
j: INT;
% assertions
ASSERT FORALL (k: INT): k>i => k>j;
% queries
QUERY EXISTS (w:INT): w<i AND w<j;
```

CVC3: example #2

```
% constants
i: INT;
j: INT;
% assertions
ASSERT i=15;
ASSERT j=10;
% queries
QUERY i=j;
```

CVC3: Example #3

```
% constants
i: INT;
j: INT;
% functions
F1: (INT,INT) -> BOOLEAN;
% assertions
ASSERT FORALL (k: INT): F1(k,i) => F1(k,j);
% queries
QUERY NOT F1(i,j) OR i=j ;
```

CVC3: Example #4

```
% constants
i: INT;
j: INT;
w: INT;
% functions
F1 : (INT, INT) -> BOOLEAN;
% assertions
ASSERT FORALL (k: INT) : F1(k,i) => F1(k,j);
% queries
QUERY F1(w,i) => F1(w,j) ;
```

Pest

- Very basic while-style, recursion-free, multi-procedural, monomorphic language
- Only integers, Booleans and integer arrays
- Integer values are unbounded.
- Expressive specification mechanism

Pest: Maximum number

```
Max(a,b,r)
:? true
:! (a>=b => r=a) && (a<b=>r=b)
:* r
{
    if a>=b then
        r<-b
    else
        r<-b
}
```

Pest: Integer division

```
Divide(a,b,r)
:? b>0
:! r=a/b
:* r
{
    r<-a/b
}
```

Pest: Increment number

```
Increment(i)
:? true
:! i=i@pre+1
:* i
{
    i<-i+1
}
```

Pest: Integer arrays

```
ArrayGetValue(A,i,r)
:? |A|>0 && i>=0 && i<|A|
:! r=A[i]
{
    r<-A[i]
}
```

Loops & Quantifiers

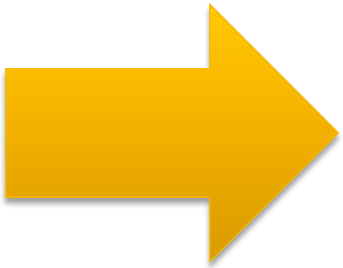
```
MaxArrayValue(A,max)
:? |A|>0
:! forall k from 0 to |A|-1:max>=A[k]
:* max
{
      max<-A[0]
      local i<-1
      while i<|A|
            :?! 1<=i && i<=|A|
            && forall k from 0 to i-1: max>=A[k]
            :# |A|-i
      do {
            if max<A[i] then
                  max<-A[i]
            else
                  skip
            i<-i+1
      }
}
```

From Pest to CVC3

java -jar tp1.jar examples/test1.pest

INPUT:

```
 test1(a)
:? a = 10
:! a = 11
:* a
{
   a <- a + 1
}
```



OUTPUT:

```
 % Var declare
a$0:INT;
a$1:INT;
% pre
ASSERT (a$0=10);
% body
ASSERT (a$1 = (a$0 + 1));
% post
QUERY (a$1=11);
```

From Pest to CVC3

- Available: Parser from Pest program files to Pest Abstract Syntax Tree (AST)

- **Hint:** Extend the visitor class (see "visitor design pattern") to output the verification conditions you need.