SyGuS Syntax for SyGuS-COMP'15

Rajeev Alur, Dana Fisman, P. Madhusudan, Rishabh Singh, Armando Solar-Lezama

Abstract. We describe here the new syntax for SyGuS, to accommodate for the new tracks of SyGuS-COMP'15: (1) conditional linear integer arithmetic and (2) invariant synthesis. The enhanced syntax can be easily translated to the syntax of SyGuS-COMP'14 [2], which will be the syntax for the general track of SyGuS-COMP'15.

1 Conditional Linear Integer Arithmetic track

This track is aimed at synthesizing function in conditional linear integer arithmetic. That is, the grammar of the function to be synthesized is determined a priory to be in LIA, and no further syntactic restrictions are imposed. To allow this we relax the syntax of `synth-func` by making the last argument, which describes the grammar for the function to be synthesized, optional. When it is absent the the only syntactic restriction for the formula to be synthesized is to be in the defined logic.

command `synth-func`

The new syntax of `synth-func` is:

$$\begin{array}{rl} \langle \mathit{SynthFunCond} \rangle & ::= \ (\texttt{synth-func}\ \langle \mathit{Symbol} \rangle\ ((\langle \mathit{Symbol} \rangle\ \langle \mathit{SortExpr} \rangle)^*)\ \langle \mathit{SortExpr} \rangle\ (\langle \mathit{NTDef} \rangle^+))\ | \\ & ::= \ (\texttt{synth-func}\ \langle \mathit{Symbol} \rangle\ ((\langle \mathit{Symbol} \rangle\ \langle \mathit{SortExpr} \rangle)^*)\ \langle \mathit{SortExpr} \rangle) \end{array}$$

Example

The following is a legal SyGuS problem that can be used in the Conditional Linear Integer Arithmetic track of SyGuS-COMP'15:

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int)
(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (or (= y (max2 x y)))))
(check-synth)
```

Since `synth-fun` does not provide a grammar argument, the function `max2` is required to be in LIA, and no other syntactic restrictions on it are imposed. That is, it is equivalent to the following example which is given in SyGuS syntax for the general track.

```
(set-logic LIA)

(synth-fun max2 ((x Int)(y Int)) Int
    ((Start Int (StartInt))

     (StartInt Int (x   y  ConstantInt
            (+ StartInt StartInt)
            (- StartInt StartInt)
            (* StartInt ConstantInt)
            (* ConstantInt StartInt)
            (div StartInt ConstantInt)
            (mod StartInt ConstantInt)
            (ite StartBool StartInt StartInt)))

 (ConstantInt (Constant Int))

     (StartBool Bool (true   false
            (and  StartBool StartBool)
            (or   StartBool StartBool)
            (=>   StartBool StartBool)
            (xor  StartBool StartBool)
            (xnor StartBool StartBool)
            (nand StartBool StartBool)
            (nor  StartBool StartBool)
            (iff  StartBool StartBool)
            (not  StartBool)
            (=    StartBool StartBool)
            (<= StartInt StartInt)
            (=  StartInt StartInt)
            (>= StartInt StartInt)
            (>  StartInt StartInt)
            (<  StartInt StartInt)))))

(declare-var x Int)
(declare-var y Int)

(constraint (>= (max2 x y) x))
(constraint (>= (max2 x y) y))
(constraint (or (= x (max2 x y)) (= y (max2 x y))))

(check-synth)
```

2 Invariant Synthesis Track

The `synth-fun` command declares a function to be synthesizes. For the *invariant synthesis track* of SyGuS-COMP'15, we want to synthesize predicates. In addition we want to have a mechanism for relating the constraint to the pre-condition, post-condition and transition relation, which rely on primed and unprimed versions of variables. We suggest to add the following keyword

`synth-inv`, `declare-primed-var`, and `inv-constratint`.

with the syntax and semantics defined below.

Command `synth-inv`

The syntax of `synth-inv` is similar to that of `synth-fun` but does not require the return sort which is always `Bool`.

$$\begin{aligned} \langle \mathit{SynthInvCmd} \rangle &::= (\texttt{synth-inv}\ \langle \mathit{Symbol} \rangle\ ((\langle \mathit{Symbol} \rangle\ \langle \mathit{SortExpr} \rangle)^*)\ \langle\ ((\langle \mathit{NTDef} \rangle^+))\rangle)\ | \\ &::= (\texttt{synth-inv}\ \langle \mathit{Symbol} \rangle\ ((\langle \mathit{Symbol} \rangle\ \langle \mathit{SortExpr} \rangle)^*)\ ) \end{aligned}$$

Command `declare-primed-var`

The `declare-primed-var` command is equivalent to two `declare-var` commands, one with the given variable, and one with the given variable suffixed by `!`, which stands for the primed version of that variable.

$$\langle \textit{DeclarePrimedVar} \rangle \ ::= \ (\texttt{declare-primed-var} \ \langle \textit{Symbol} \rangle \ \langle \textit{SortExpr} \rangle)$$

For example,

```
(declare-primed-var x Int)
```

is equivalent to

```
(declare-var x  Int)
(declare-var x! Int)
```

Command `inv-constratint`

The `inv-constratint` command has 4 arguments, all are strings corresponding to already defined names: The first is the name of the invariant to synthesize, the second is the name of a user defined function providing the pre-condition, the third is the name of a user defined function providing the transition, and the forth is the name of a user defined function providing the post-condition.

$$\langle \textit{InvConstraintCmd} \rangle \ ::= \ (\texttt{inv-constraint} \ \langle \textit{Symbol} \rangle \ \langle \textit{Symbol} \rangle \ \langle \textit{Symbol} \rangle \ \langle \textit{Symbol} \rangle)$$

The number and sorts of the arguments of the pre-condition function and the post-condition function should be the same as that of the invariant to synthesize (not all arguments must be referred to in the function definition). Since the function for the transitions needs to refer to both the primed and unprimed versions of the variables to be synthesized, its list of arguments should be a concatenation of two list of arguments for the given invariant.

An `inv-constraint` command is compiled into three constraints:
- A constraint stipulating that the pre-condition implies the invariant (with unprimed variables).
- A constraint stipulating that the conjunction of the invariant with unprimed variables and the transitions implies the invariant with primed variables.
- A constraint stipulating that the invariant (with unprimed variables) implies the pos-condition.

Example

Consider the following program

```
f(bool b) {
   int x, y;
   x = random(5,9);
   y = random(1,3);
   while (nondet) {
     y = x;
     if (b)
        x = x+10;
     else
        x = x+12;
   }
   assert(y<x);
}
```

Encoding in SyGuS-COMP'15

The problem of finding an invariant for this program can be encoded in SyGuS-COMP'15 syntax as follows:

```
(set-logic LIA)

(synth-inv inv-f ((x Int) (y Int) (b Bool)))

(declare-primed-var b Bool)
(declare-primed-var x Int)
(declare-primed-var y Int)

(define-fun pre-f ((x Int) (y Int) (b Bool)) Bool
                  (and (and (>= x 5) (<= x 9))
                       (and (>= y 1) (<= y 3))))

(define-fun trans-f ((x Int) (y Int) (b Bool) (x! Int) (y! Int) (b! Bool)) Bool
                    (and (and (= b! b) (= y! x))
                         (ite b (= x! (+ x 10)) (= x! (+ x 12)))))

(define-fun post-f ((x Int) (y Int) (b Bool)) Bool
                   (< y x))

(inv-constraint inv-f pre-f trans-f post-f)

(check-synth)
```

Compilation to general track

This is equivalent to the following problem encoded in SyGuS under first tack.

```
(set-logic LIA)

(synth-fun inv-f ((x Int)(b Bool)) Bool)

(declare-var b  Bool)
(declare-var b! Bool)
(declare-var x  Int)
(declare-var x! Int)
(declare-var y  Int)
(declare-var y! Int)

(define-fun pre-f ((x Int) (y Int) (b Bool)) Bool
                  (and (and (>= x 5) (<= x 9))
                       (and (>= y 1) (<= y 3))))

(define-fun trans-f ((x Int) (y Int) (b Bool) (x! Int) (y! Int) (b! Bool)) Bool
                    (and (and (= b! b) (= y! x))
                         (ite b (= x! (+ x 10)) (= x! (+ x 12)))))

(define-fun post-f ((x Int) (y Int) (b Bool)) Bool
                   (< y x))

(constraint (=> (pre-f x y b)
                (inv-f x y b)))

(constraint (=> (and (inv-f x y b) (trans-f x y b x! y! b!))
                (inv-f x! y! b!)))
```

```
(constraint (=> (inv-f x y b)
                (post-f x y b)))

(check-synth)
```

References

1. Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak and Abhishek Udupa. Syntax-Guided Synthesis. In *FMCAD*, pages 1–17. IEEE, 2013.
2. Mukund Raghothaman and Abhishek Udupa. Language to Specify Syntax-Guided Synthesis Problems. In *http://sygus.seas.upenn.edu/files/SyGuS-IF.pdf*, May, 2014.