Formal Verification of Synchronization Issue of System-Level Design with Automatic Abstraction

Thanyapat Sakunkonchak and Masahiro Fujita
Department of Electronic Engineering, The University of Tokyo

Introduction

Our target verification for synchronization is based on "***SpecC Language***", since SpecC
- is designed to support the SoC design
- can describe both HW and SW
- is promising the C-based specification/design/implementation
- [http://www.SpecC.org]

Objectives

- Construct a framework that can verify for the event synchronizations
- If the verification fails, the counter-example (the error path) must be automatically generated

This is based on the following concepts and tools:
- Model checking technique
- Boolean Program (Microsoft Research)
- Difference Decision Diagrams (DDDs) (IT University of Copenhagen)
- Cooperating Validity Checker (CVC) (Standford Univeristy)

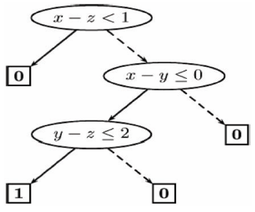
Boolean Program

- Proposed by Ball and Rajamani under SLAM project at Microsoft Research
- Boolean program abstracts irrelevant statements such that the transformed program contains only boolean variables
- Since boolean program abstracts the original one: some properties that hold in boolean program must eventually hold in the original program
- The verification process is completely automatic, with counter-example provided when the properties to be checked are failed

Difference Decision Diagrams

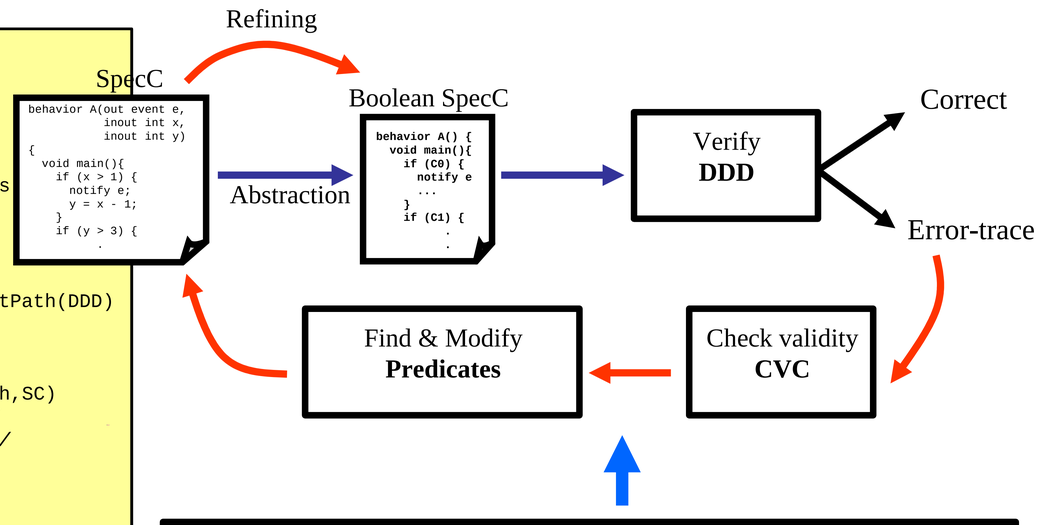
- Introduced by Mϕller, et.al. (IT University of Copenhagen)
- Symbolic representation of 'non-boolean', e.g. inequalities: less efficient if using BDDs

Represents graph for
¬(x-z<1)Λ(x-y<=0)Λ(y-z<=2)




```
procedure SpecCVerification
declare
   SC: a SpecC source code
   BS: a Boolean SpecC code
   Pre: set of predicates abstracted from SC
   DDD: Difference Decision Diagram
   Simpath: a simulation path obtain from DDDs
begin
   <BS,Pre> := Abstraction(SC)
   while not timeout or Pre != ∅ do
      DDD := ConstructDDD(BS)
      <result1,Simpath> := Verify&FindShortestPath(DDD)
      if (result1)
         Terminate /*Sync. Satisfied*/
      else
         <result2,P> = ChoosePredicate(Simpath,SC)
         if (result2) /*Sync. NOT Satisfied*/
            Terminate /*with COUNTER-EXAMPLE*/
         Pre := Pre \ P
         BS := ModifyPredicate(SC,BS)
      fi
   od
   Terminate /*Cannot conclude*/
end
```

```
procedure ChoosePredicate(Simpath,SC)
begin
   result := MatchSimpath2SC&PutInCVC(Simpath,SC)
   P := GroupSameBoolean&FindOneThatInvalid(Simpath)
return <result,P>
```



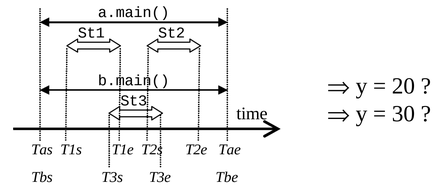
Verification Flows (Both graphical representation and Pseudo code)

Synchronization in SpecC Language

```
main() {
   par{ a.main();
        b.main(); }}

behavior a {
   main() { x=10;        /*st1*/
            y=x+10;      /*st2*/ }}

behavior b {
   main() { x=20;        /*st3*/ }}
```

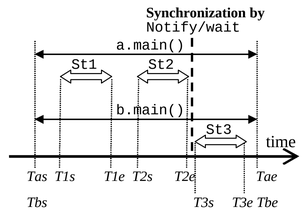


⇒ y = 20 ?
⇒ y = 30 ?

```
main() {
   par{ a.main();
        b.main(); }}

behavior a {
   main() { x=10;        /*st1*/
            y=x+10;      /*st2*/
            notify e;    /*New*/}}

behavior b {
   main() { wait e;      /*New*/
            x=20;       /*st3*/ }}
```



⇒ y = 20
Always

We use the timing constraints, e.g. Tas<=T1s<T1e<=T2s<T2e<=Tae, as our criteria for verification. And these series of inequalities can be represented using DDDs

Conclusion and Outlook

Current Implementation

- can handle the basic SpecC constructs
- can verify the SpecC code that contains synchronization semantics
- the properties can be checked by simply check for the assertion violation (reachability problem)
- when the properties are not satisfied, the counter-example can be generated

Future plan

- the entire processes, from abstraction to generation of counter-example, are still not completely automatic (need to assemble all parts together and make them automatically running)
- upon the completion of making the entire flows running automatic, we will try to verify with some real implementations

東京大學
THE UNIVERSITY OF TOKYO