Automated Test Program Generation for an Industrial Optimizing Compiler

Chen Zhao, **Yunzhi Xue**, Qiuming Tao,
Liang Guo, Zhaohui Wang

Institute of Software, Chinese Academy of Sciences

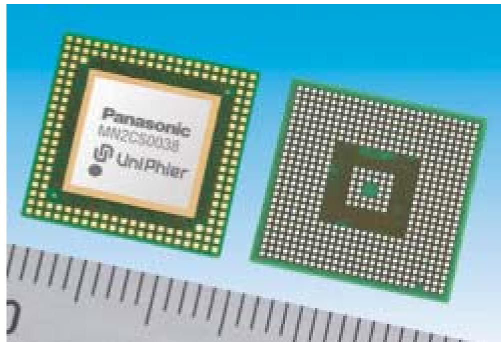
May 18, 2009

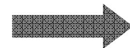
The 4th International Workshop on Automation of Software Test
Vancouver, Canada

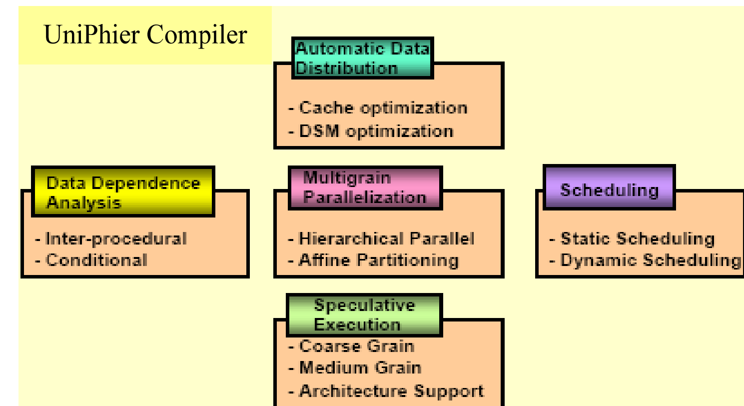
Outline

- Motivation
- Brief Introduction to Compiler Optimizations
- JTT: Automated Tool for Test Program Generation
- Formal Model-based Generation: From Formula to Program
- Evaluation in Testing UniPhier Compiler
- Contributions

Motivation







UniPhier: A popular, high-performance, embedded chip from Panasonic

A collection of compiler optimizations

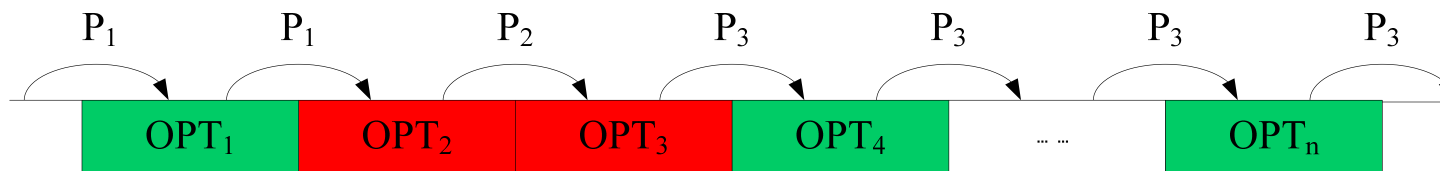
All optimizations must be TRUSTABLE!

ISCAS

Compiler Optimizations

What is compiler optimization?

"*Compiler optimization is the process of tuning the output of a compiler to minimize or maximize some attribute of an executable computer program.*" ——Wikipedia

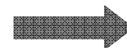


Well-Known Optimizations:

Dead Code Elimination, Common Subexpression Elimination, Loop Interchange, Loop Unrolling…

```
a = b * c + g;

d = b * c * d;
```



```
tmp = b * c;
a = tmp + g;

d = tmp * d;
```

ISCAS

Compiler Optimizations (Cont'd)

- General Form

$$I \rightarrow I' \ \textit{if} \ \{\textit{Side Conditions}\}$$

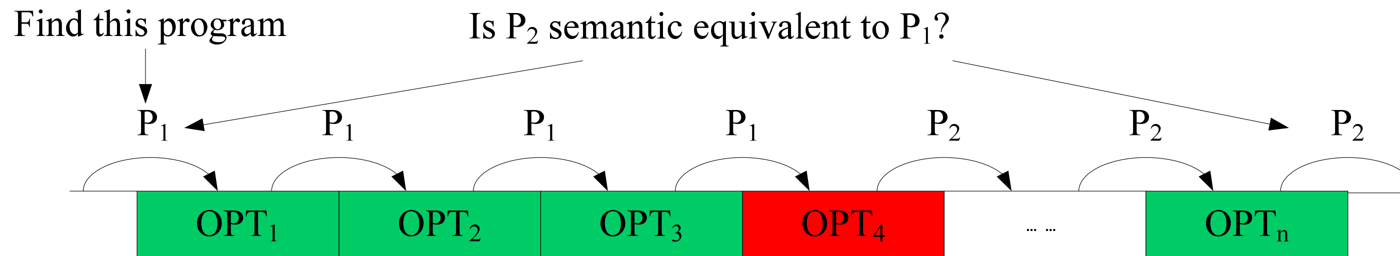
- Features
  - Semantic-preserving
  - Many possible prerequisites
  - Dependent to data types

ISCAS

How to Test An Optimization

- Basic approach

  Find a program to active the optimization



- Requirements for test programs
  - Definitely active specific optimization
  - Cover as many prerequisites as possible
  - Use various data types
  - Interleave optimizations and data types

ISCAS

Extra Difficulties in Testing UniPhier Compiler

Architecture-dependent optimizations
→ not common optimizations
→ special program to active such optimizations

Architecture-dependent data types
→ not common data types
→ special program to use such data types

Especially when such optimizations and data types are interleaving with each other!

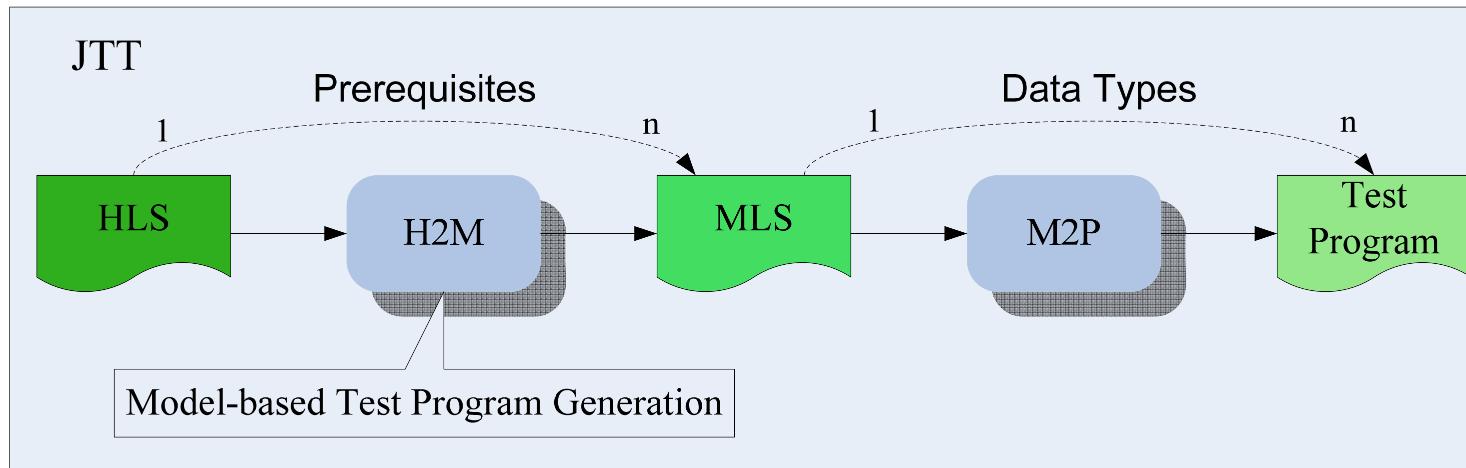
ISCAS

Possible Solutions

Solution	What's it?	Advantage	Disadvantage
Benchmark	SPEC Benchmarks, EEMBC, etc	Popular, public	Performance-oriented, not optimization-oriented; without UniPhier-related features
Public Test Suite	GCC Test Suite and so on	Public available	Language-oriented, not optimization-oriented; without UniPhier-related features
Internal Test Suite by Panasonic	Legacy of past testing in Panasonic	Optimization-oriented, Concerning both common and UniPhier-dependent optimizations	Not available for new architecture features, new optimizations and new data types
Manual Programming	Test program written by test engineers	Optimization-oriented, concerning all optimizations, be aware for evolution of UniPhier and its compiler	Labor-intensive, test efficiency and quality are difficult to achieve

ISCAS

JTT: An Automated Tool for Test Program Generation



HLS (High Level Script): specify test requirements, including optimizations, data types used, statement types, operators and so on. Readable for test engineers.

MLS (Middle Level Script): specify test program template, including program statement template, data types, operators and so on

H2M: transform HLS to MLS M2P: transform MLS to test program

H2M is based on formal model of compiler optimizations

ISCAS

HLS Script & MLS Script

```
Element {
   Var-type: char, pointer(int), struct(char*pointer(int))
   Statement: if, for, switch, goto
   Operator: +, *, <<, &
}
Structure {
   Intra-module: Type = if+loop, Nest = 3, Parallel = 2
   Inter-module: Depth = 2, Width = 3, Recursion = 0
}
Content {
   ScalarOpt: DCE, CSE
   LoopOpt: Skew, Fusion, Interchange
   Pragma: Prefetch, UnrollLength
}
```

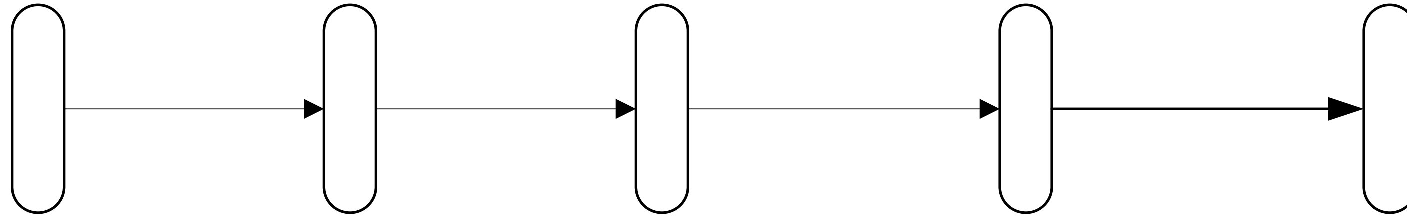
An Example HLS Script

```
Element { ... }
Structure {
   Func1( ){
      IF(||,2;&&,3){
         ASS
      }
      LOOP(Times=20){
         Func2
         ASS
      }
   }
   Func2( ){
      ASS
   }
}
Content { ... }
```

A Possible MLS Script

- Advantage of Two-Level Script
  - Separate test program requirements and test program specification
  - Hide details of test program and architecture in MLS
  - Prerequisites and data types can be interleaved easily
  - Thousands of test programs with respect to HLS can be generated automatically and easily

Model-based Generation: from Formula to Program



CTL Formulas for Optimizations | Node Control Graph | CFG with State and Path Quantifiers | CFG with First-Class Formulas | Program Template in MLS

CTL Formulas for Optimizations: Computation Tree Logic (CTL) Formulas (only name listed in HLS)

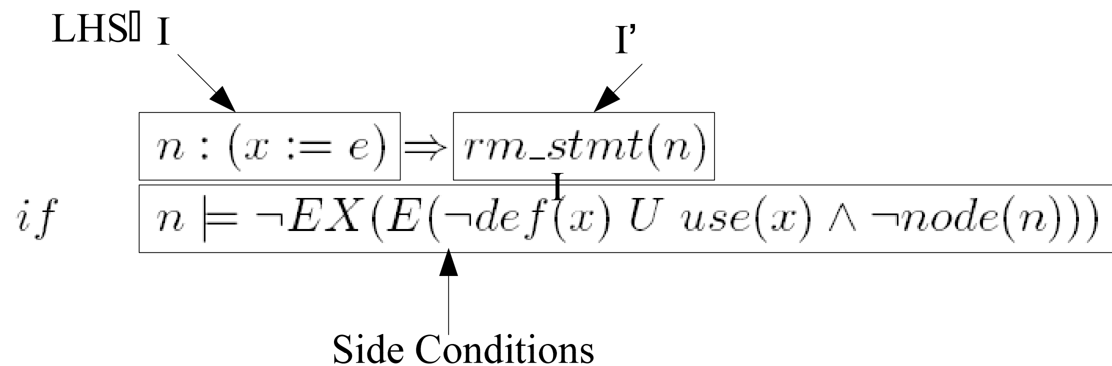
NCG: Node Control Graph, a special CFG

Construction of CFG: enrich complexity of resulting code

Expansion of Temporal Quantifiers: make all nodes in CFG only labeled with first-order formulas

CTL Formulas for Compiler Optimizations

A conditional rewriting rule containing CTL formulas for Dead Code Elimination

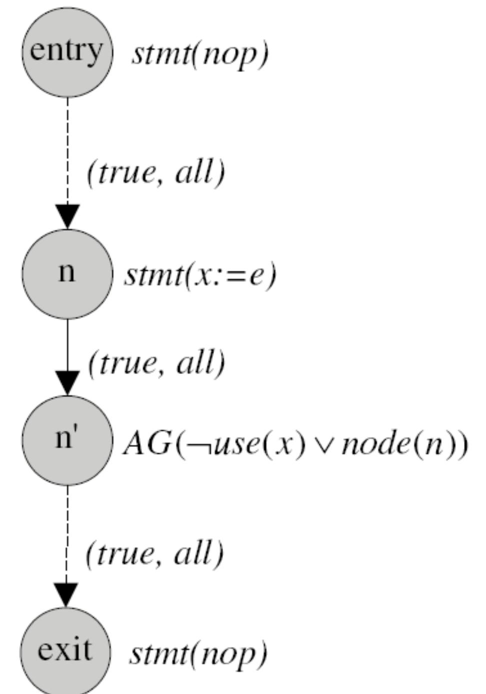


LHS specifies some statements in test program
Side conditions specify why an optimization is semantic-preserving – a relationship between LHS and other statements in test program

LHS + Side Conditions → Test Programs

Node Control Graph

- A special CFG
  - Constructed according to CTL Formula
    - X operator leads to a first-successor node and a solid edge to current node
    - U operator leads to a successor node and a dashed edge to current node
  - Each dashed edge can be replaced by a basic block
  - Each edge is labeled with a formula for all or some paths in the basic block
  - Each node is labeled with a formula for statement at the node

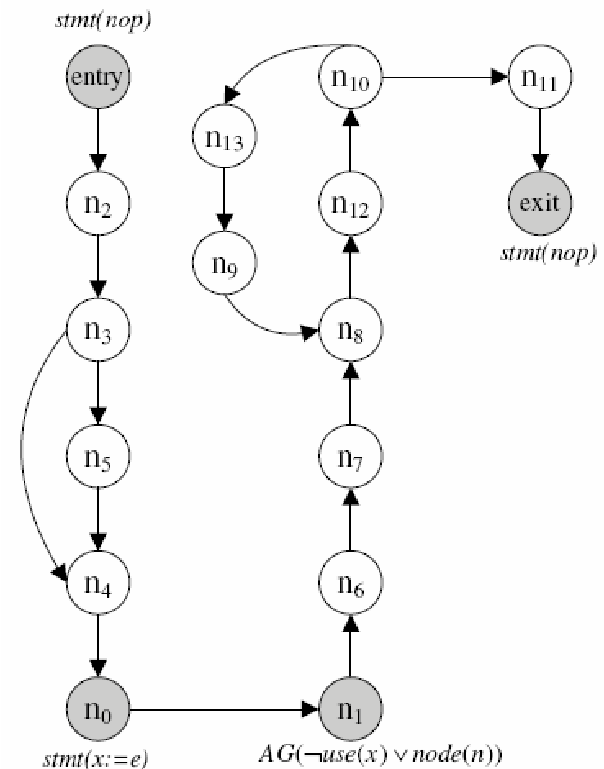


ISCAS

Construction of CFG with State & Path Quantifiers

- Replace each dashed edge with a basic block

The basic blocks are generated randomly



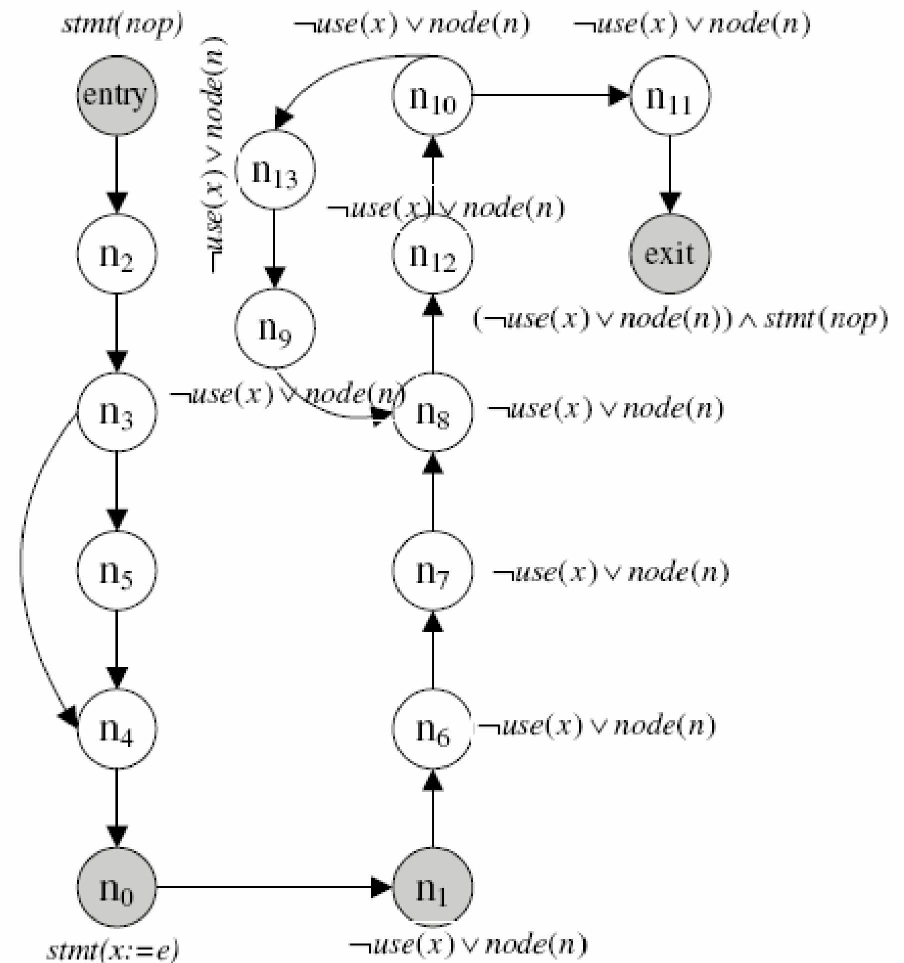
ISCAS

CFG with first-order Formulas

Expand temporal quantifiers to all involved nodes

Make all nodes only labeled with first-order formulas for convenience of generation

Different expansion for A/E path quantifiers and G/F state quantifiers



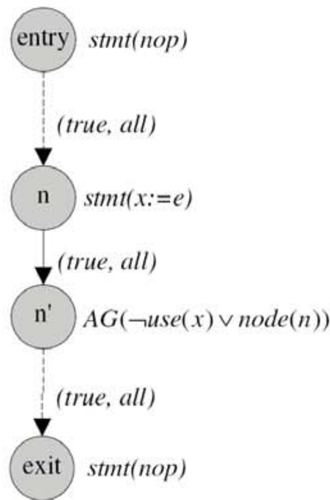
ISCAS

Example Generation

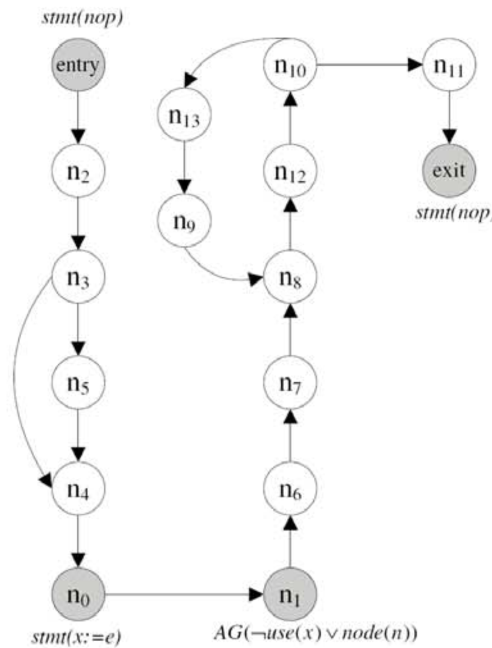
$n : (x := e) \Rightarrow remove\ statement\ n$

$if \quad n \models \neg EX(E(\neg def(x)\ U\ use(x) \wedge \neg node(n)))$

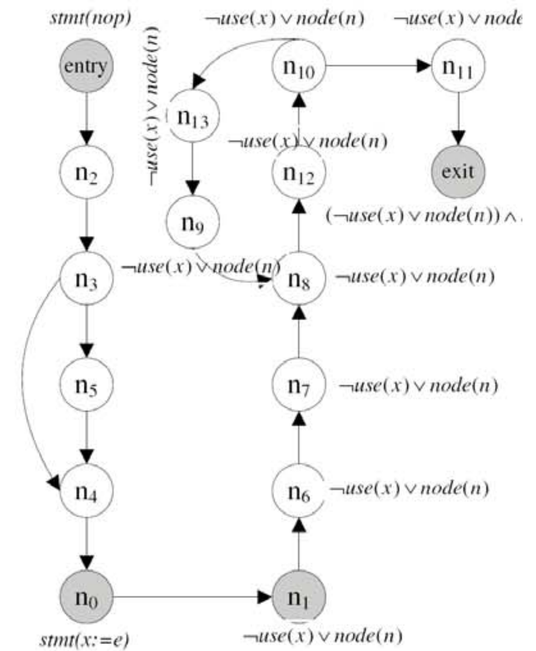
(a) DCE Formula



(b) NCG of A Possible DCE Instance



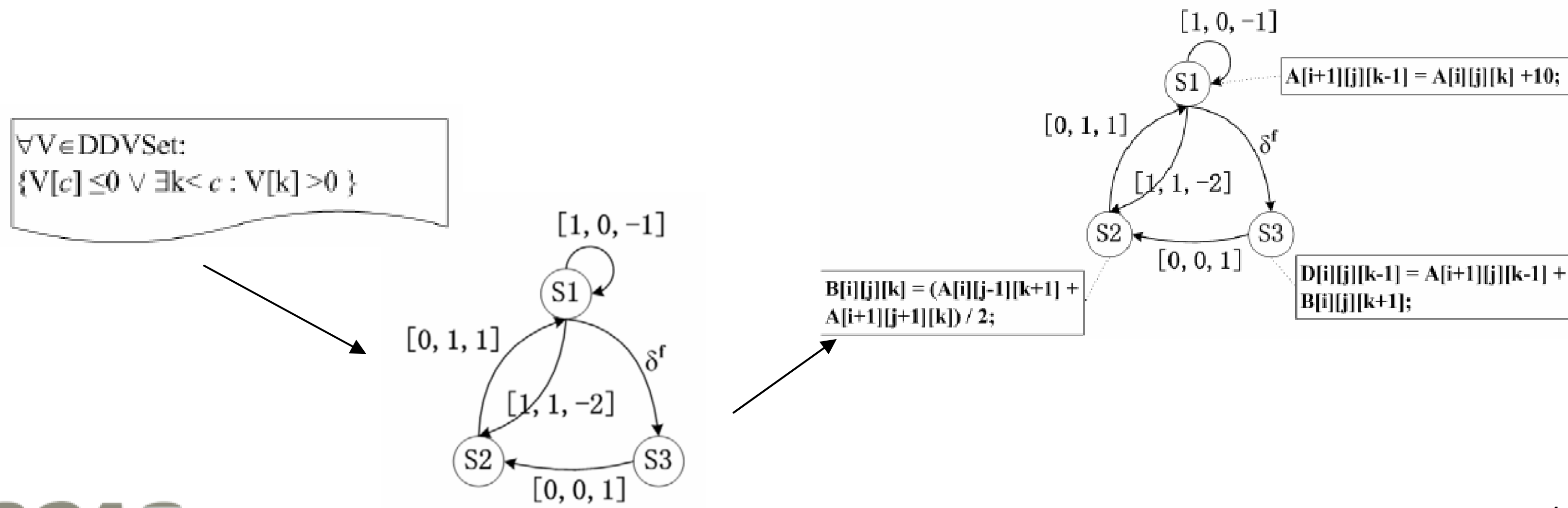
(c) CFG Labeled with Temporal-Logic Formula



(d) Program Template

Data Dependence Generation for Loop Optimizations

- Assign flow-, anti-, and output-dependence to statements within loop nests
  - Steps:
    1. Assign valid types of data dependence to statements
    2. generate proper subscription function according to data dependence (Linear Integer Programming)



Evaluation in UniPhier Compiler

- JTT v.s. Benchmark & Public Test Suite

	M1	M2	M3	M4	M5	M6	M7
Benchmark & Public Test Suite	85%	18%	79%	0%	0%	1%	1%
With JTT	88%	54%	80%	63%	49%	68%	56%

Statement Coverage of 7 Optimizing Modules in UniPhier Compiler

- JTT v.s. Manual Programming

6,000+ test programs v.s. 1,000+ test programs

- Bugs Found
  - 6 new serious bugs found by JTT
  - Found 21% bugs by JTT during testing

ISCAS

Contributions

- Model-based test program generation for optimizing modules
- Automated script-driven test case generation for a large system with complex inputs
- Application of formal methods in a large complex system

ISCAS

Thanks for Your Attention!
Any Questions?

Yunzhi Xue

yunzhi@itechs.iscas.ac.cn

Institute of Software, Chinese Academy of Sciences