Vdiff: A Program Differencing Algorithm for Verilog HDL

Adam Duley
ARM Inc

Christopher Spandikow
IBM Corporation

Miryung Kim
The University of Texas at Austin

Problem: Limitations of using *diff* on evolving hardware designs

- assumes sequential execution semantics
- relies on code elements having unique names
- does not leverage Boolean expression equivalence checking despite the availability of SAT solvers

Solution: Vdiff

- a position-independent differencing algorithm with intimate knowledge of Verilog semantics
- 96.8% precision with 97.3% recall compared to manually classified differences
- produces syntactic differencing results in terms of Verilog-specific change types

Outline

- **Motivation**
- Verilog Background
- Vdiff Algorithm
- Evaluation
- Conclusions

Motivation

- hardware designers collaboratively evolve large Verilog programs
- hard to use *diff*-like tools during code reviews
- develop a foundation for reasoning about evolving hardware design descriptions

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
output [fifo_width-1:0] data_out;
reg [fifo_counter_w-1:0] fifo;
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
  begin
    fifo[1] <=  0;
    fifo[0] <=  0;
  end
end
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
output [fifo_width-1:0] data_out;
reg [fifo_counter_w-1:0] fifo;
```

Modules are building blocks with
an explicit input and output port interface.

A module is similar to a Java class.

```
    fifo[1] <=  0;
    fifo[0] <=  0;
  end
end
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
output [fifo_width-1:0] data_out;
reg [fifo_counter_w-1:0] fifo;
wire [fifo_pointer_w-1:0] overrun;
```

Input and output ports are public interfaces that connect modules to external hierarchy.

They are similar to a constructor's parameter list in Java.

```
  end
end
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
output [fifo_width-1:0] data_out;
reg [fifo_counter_w-1:0] fifo;
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
```

Wires, registers, & integers are variable declarations.

```
  end
end
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
```

`always` blocks are similar to Java methods.
However, they execute concurrently
when the sensitivity list is true.

```
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
  begin
    fifo[1] <=  0;
    fifo[0] <=  0;
  end
end // always
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
output [fifo_width-1:0] data_out;
reg [fifo_counter_w-1:0] fifo;
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
  begin
    fifo[1] <=  0;
```

Assign statements model concurrent dataflow.

```
end // always
assign data_out = fifo[0];
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
```

Blocking statements are sequential assignments.

```
reg [fifo_counter_w-1:0] fifo;
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
  begin
    fifo[1] =  0;
    fifo[0] =  0;
  end
end
assign data_out = fifo;
endmodule
```

Verilog HDL Background

```
include "uart_defines.v"
module uart_rfifo (clk, reset, data_out);
input clk, reset;
wire [fifo_pointer_w-1:0] overrun;
always @(posedge clk or posedge reset)
begin
  if(reset)
  begin
    fifo[1] <= 0;
    fifo[0] <= 0;
  end
end
assign data_out = fifo;
endmodule
```

Non-blocking statements are concurrent assignments, and they are prevalent in Verilog designs.

Diff Results

```
always @(posedge clk)
begin
  if(reset)
  begin
    fifo[1] <=  0;
-   fifo[0] <=  0;
  end
end // always
-always @(posedge clk )
-begin
-  if (reset)
-    overrun <=  0;
-end // always
```

```
+ always @(posedge clk)
+ begin
+   if (reset)
+     overrun <= 0;
+ end
always @(posedge clk)
begin
  if(reset)
  begin
+   fifo[0] <=  0;
    fifo[1] <=  0;
+   fifo[2] <=  0;
  end
end // always
```

Verilog's non-unique identifiers and concurrent semantics cause *diff* to identify a large amount of false positives.

Outline

- Motivation
- Verilog Background
- **Vdiff Algorithm**
- Evaluation
- Conclusions

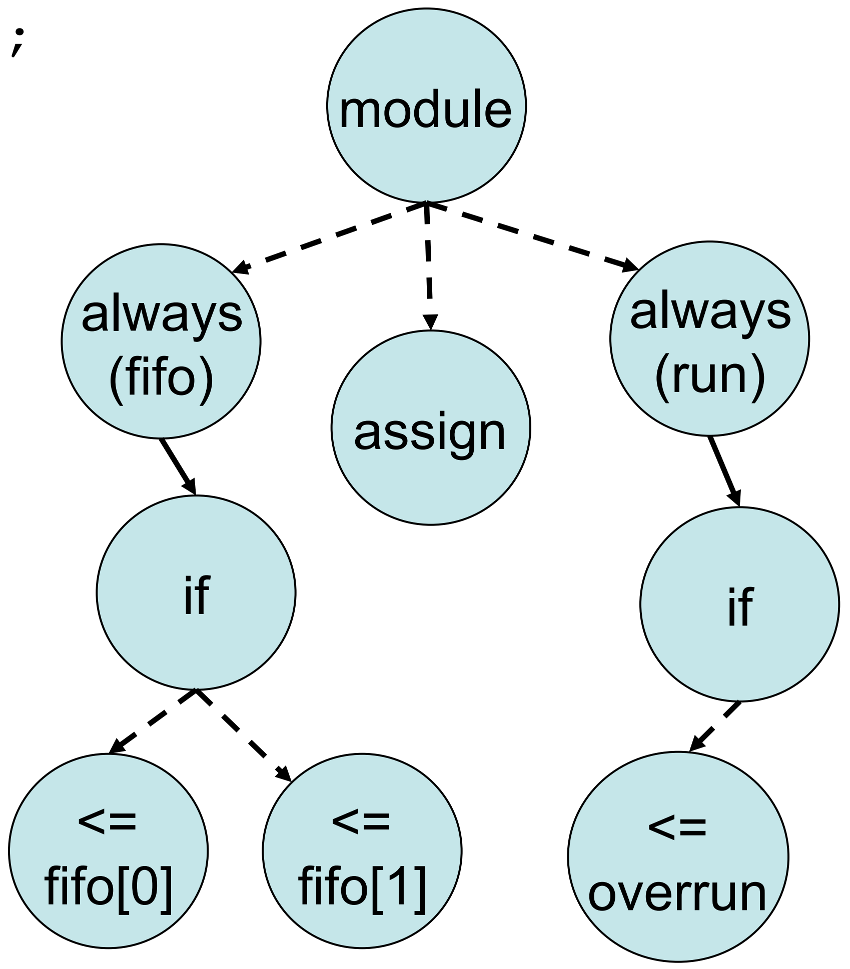
Algorithm

- input: two versions of a Verilog file
- output: syntactic differences in terms of change types

1. extract Abstract Syntax Tree (AST) from each file
2. compare the two trees
3. filter false positives in changes to sensitivity lists using a SAT solver
4. categorize differences based on Verilog syntax

Extract AST

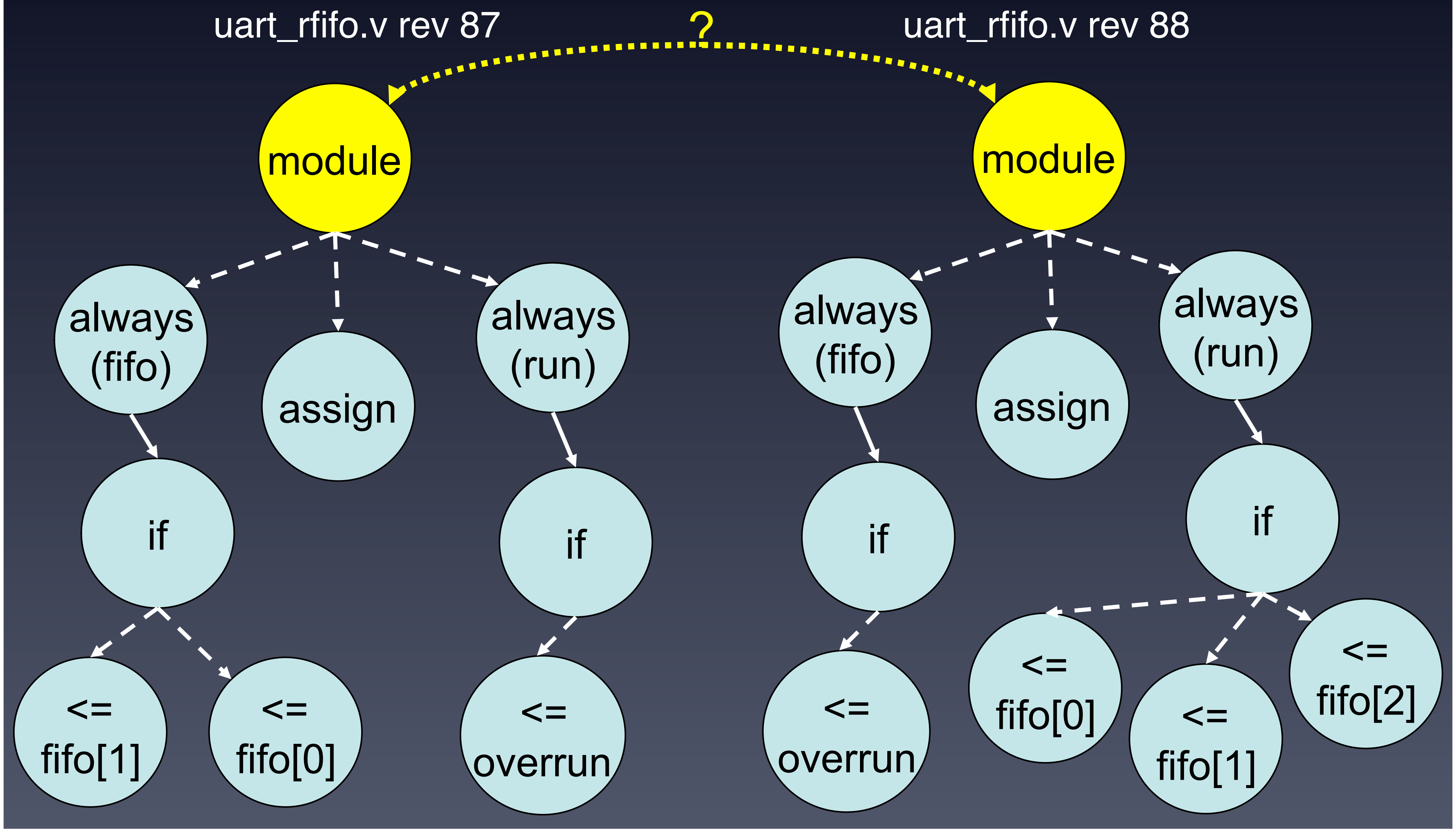
```
module uart_rfifo (clk, reset,
             data_out, overrun);
always @(posedge clk)
begin
  if(reset)
  begin
    fifo[1] <=  0;
    fifo[0] <=  0;
  end
end // always
always @(posedge clk)
begin
  if (reset)
    overrun <= 0;
end // always
assign data_out = fifo[0];
endmodule
```



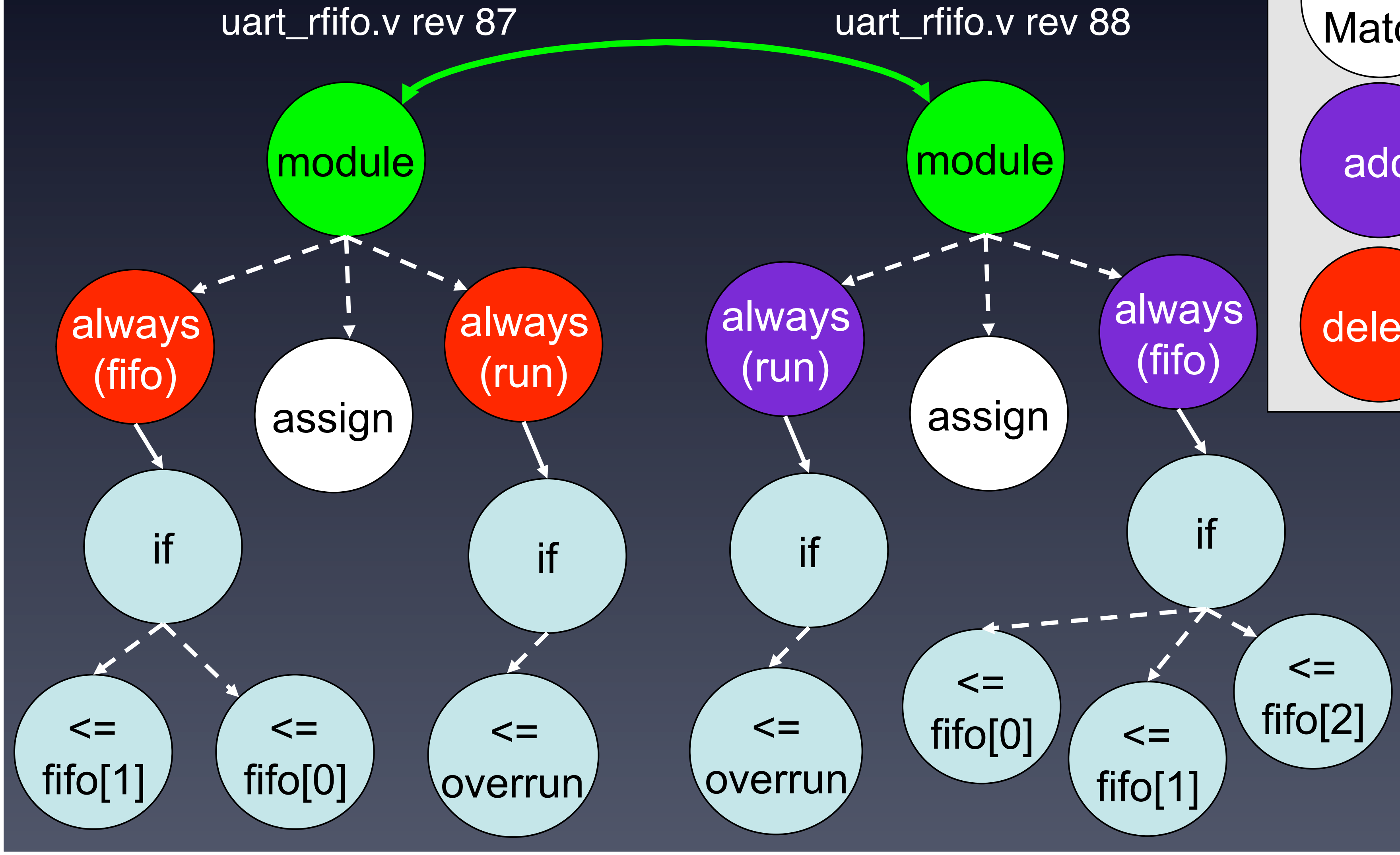
Tree Differencing Algorithm

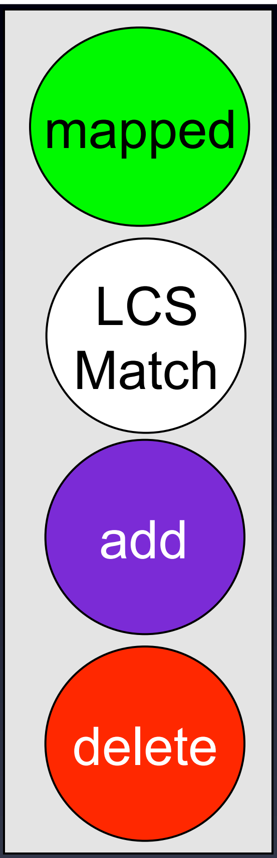
- hierarchically compare tree nodes from the top down
- initially align nodes using the longest common subsequence (LCS) algorithm—unmatched nodes are split into ADD and DELETE sets
- for each pair in ADD x DELETE, calculate the textual similarity
- use greedy weighted bipartite graph matching to associate a DELETE node to a corresponding ADD node

Tree Comparison

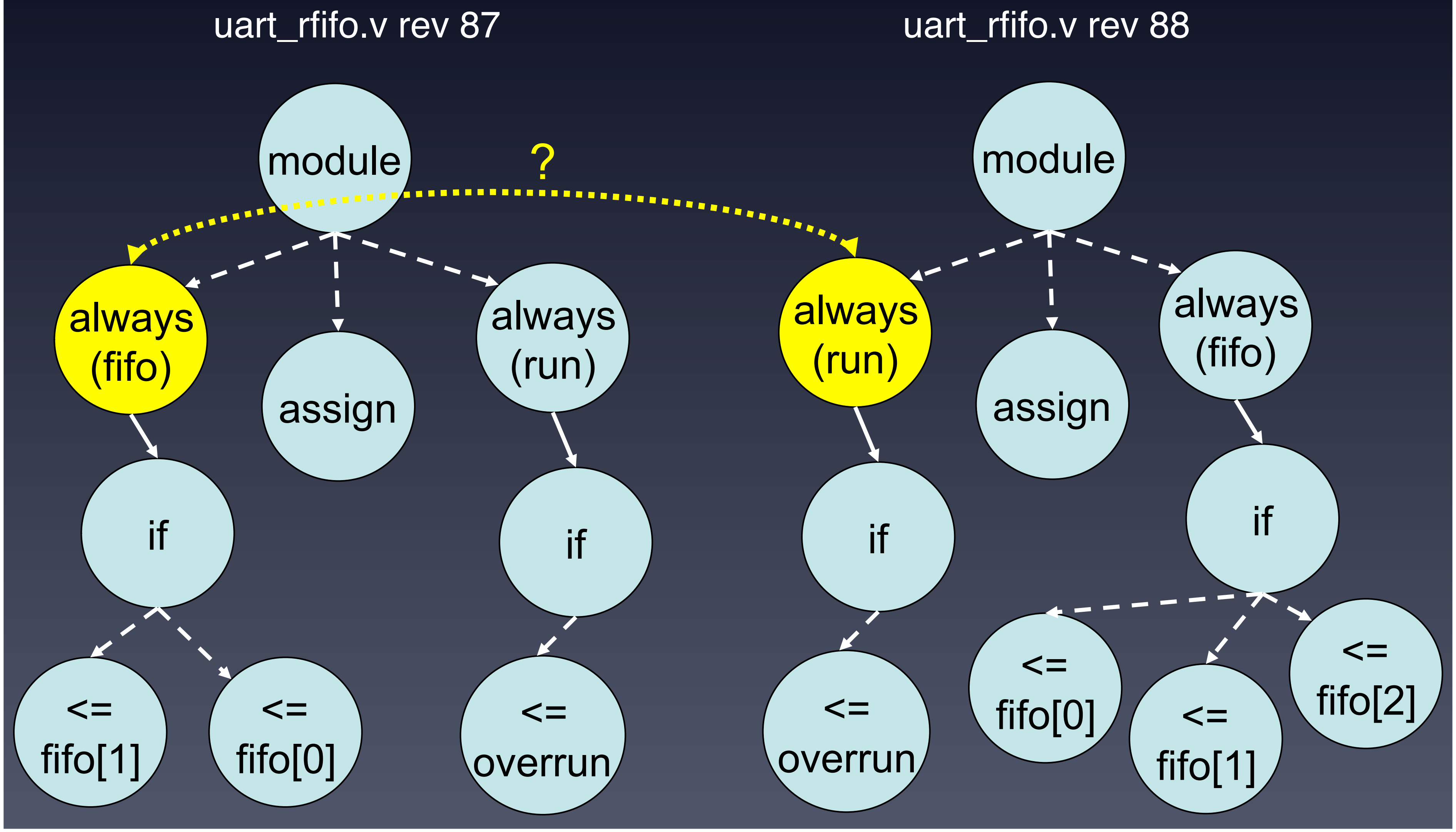


Tree Comparison

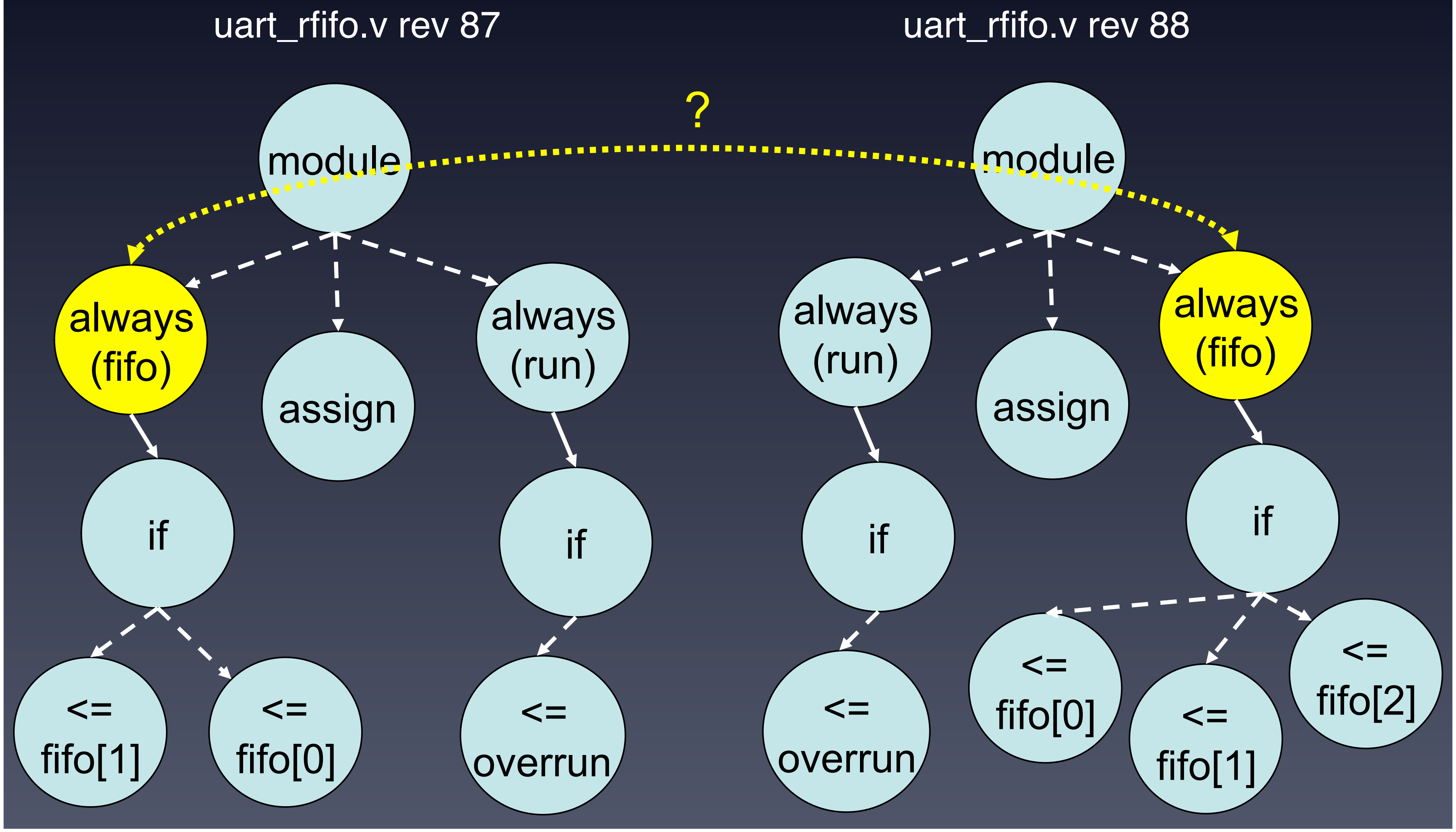




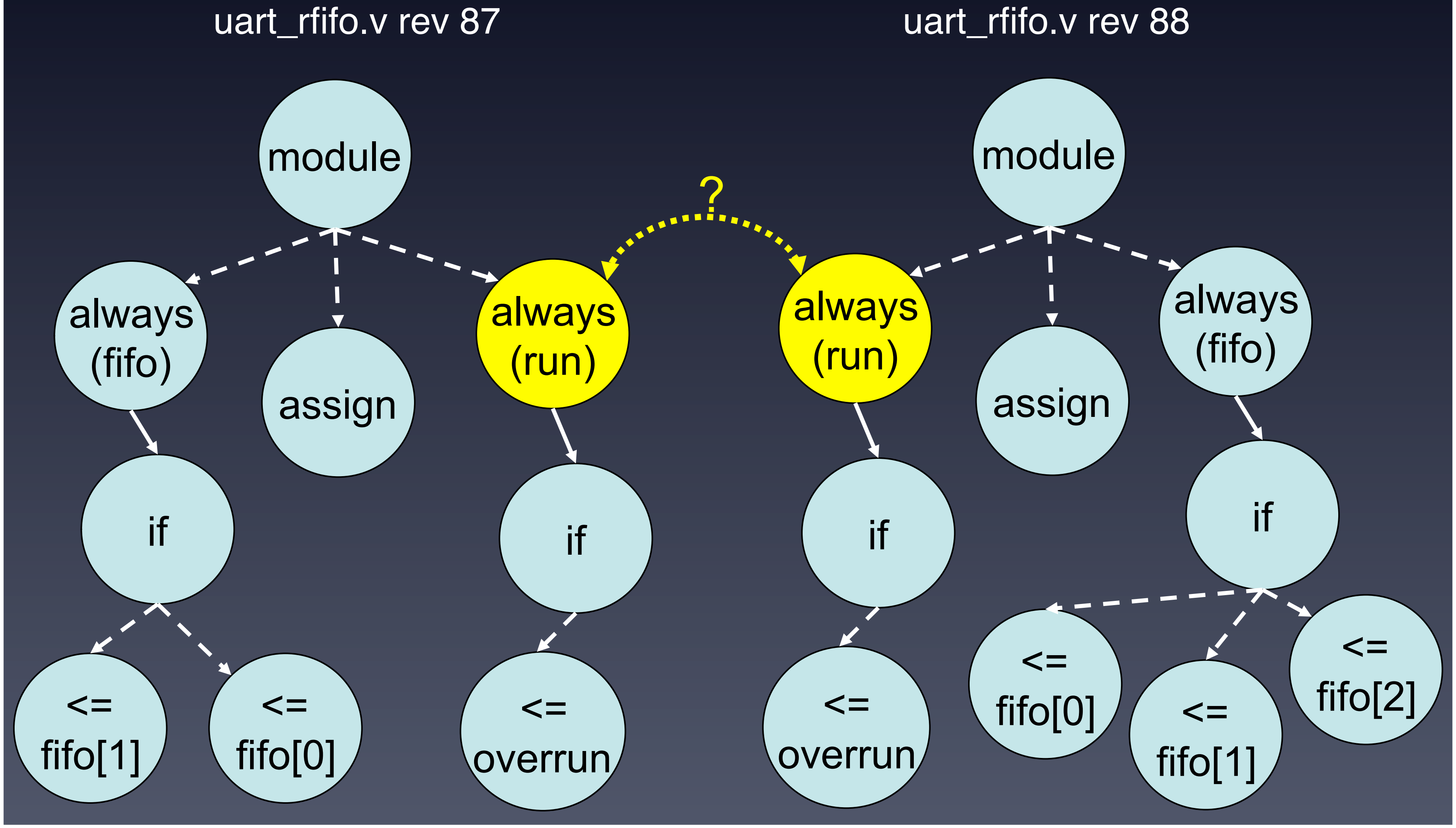
Tree Comparison



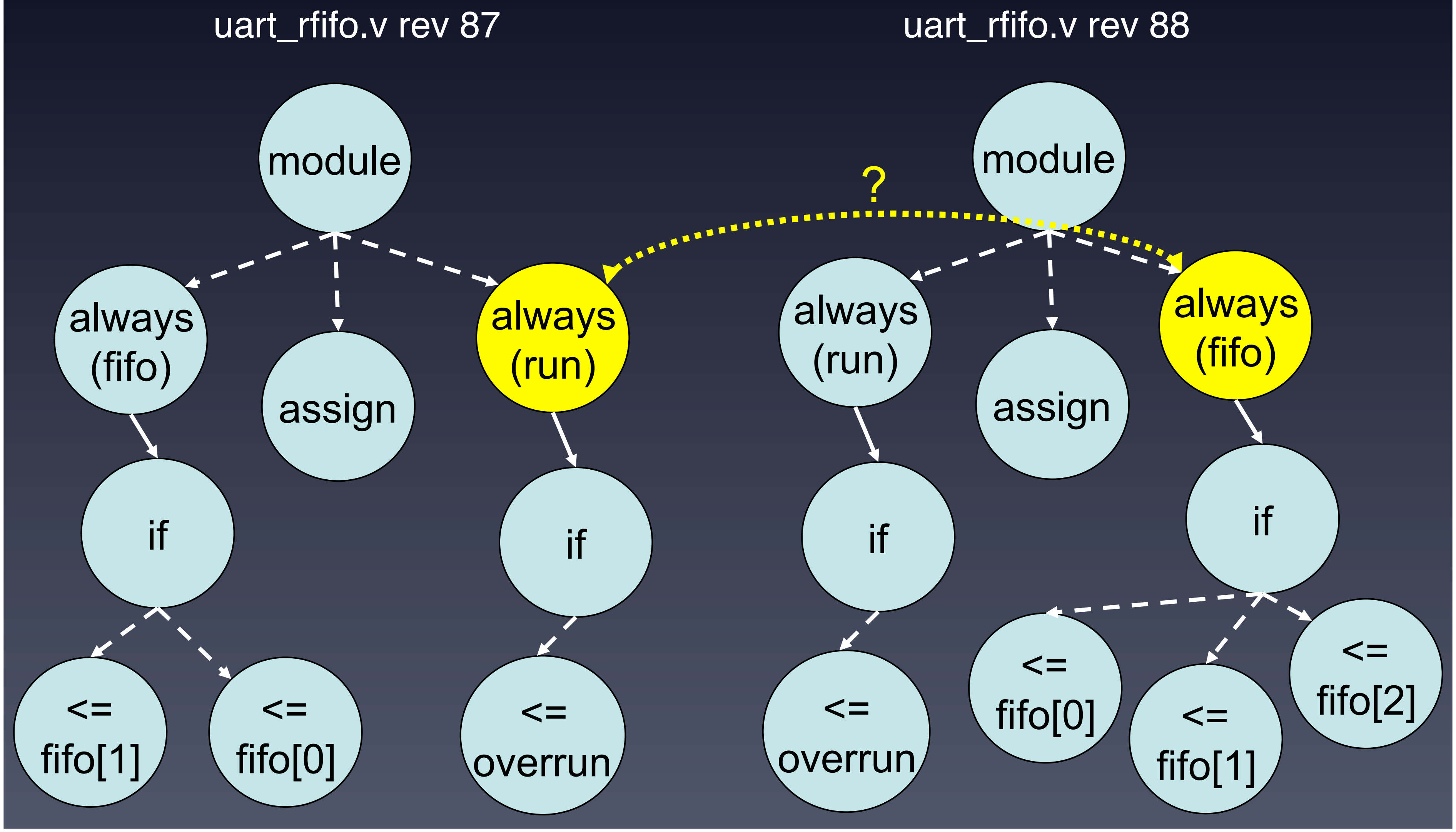
Tree Comparison



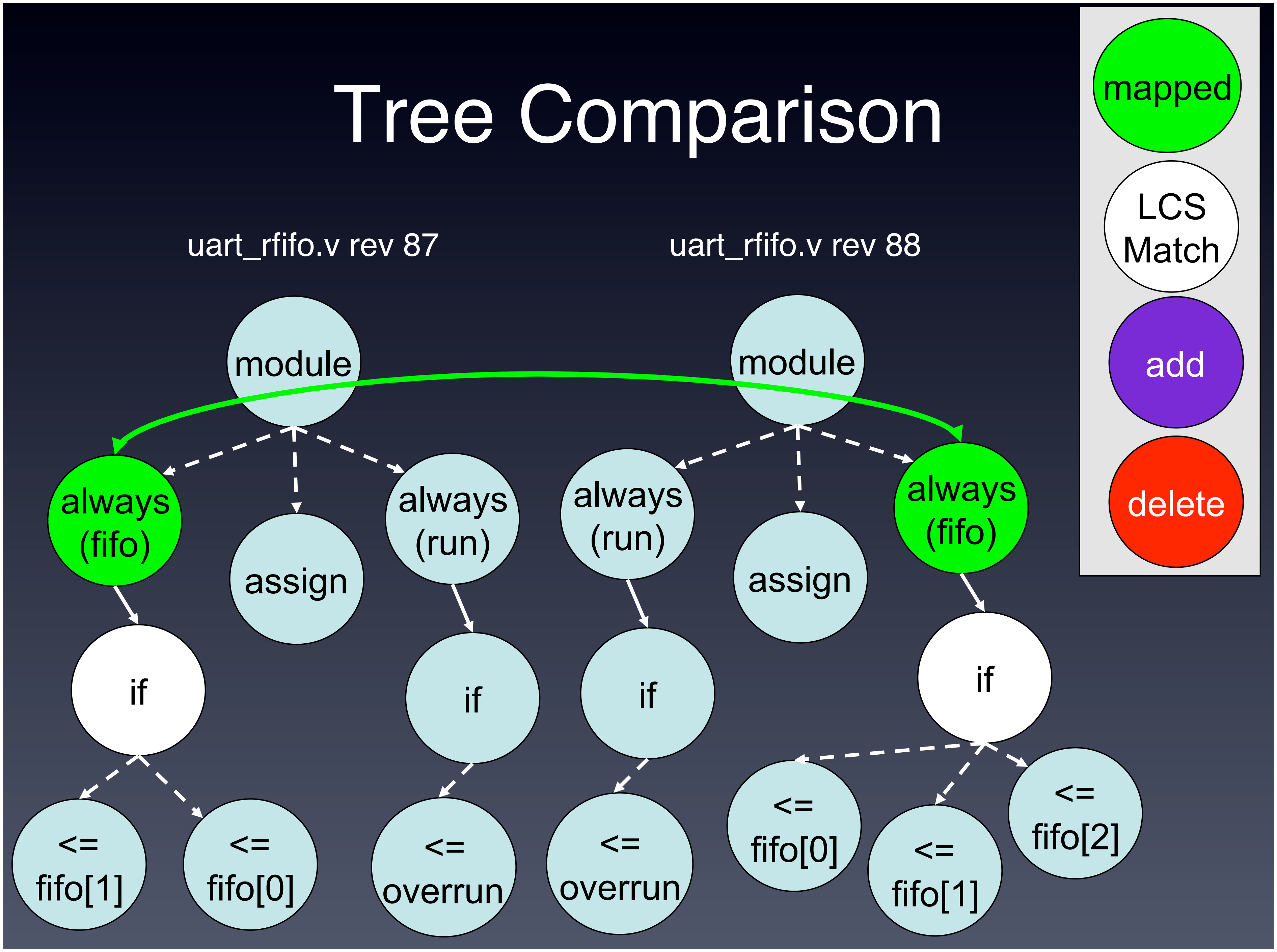
Tree Comparison



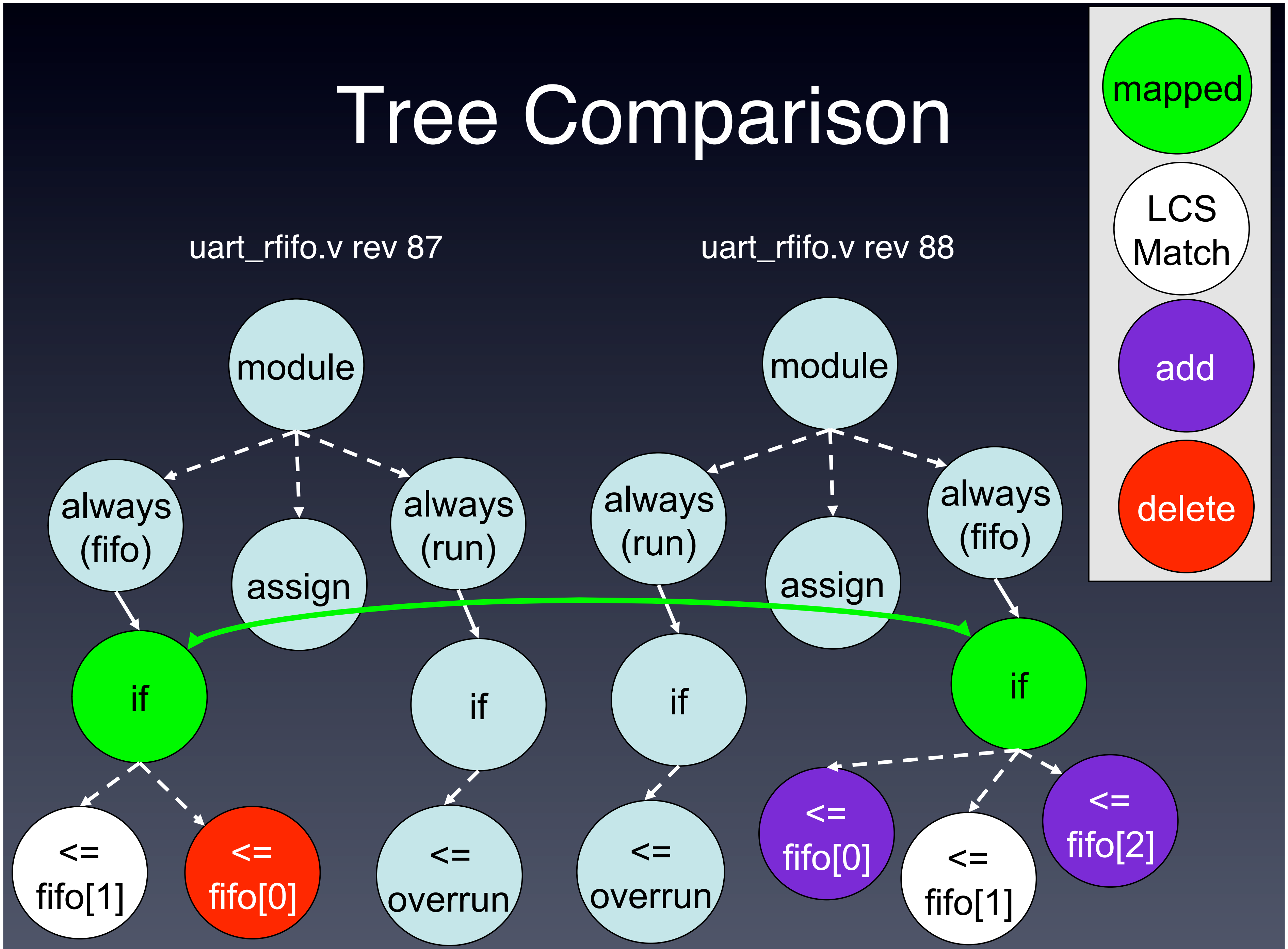
Tree Comparison



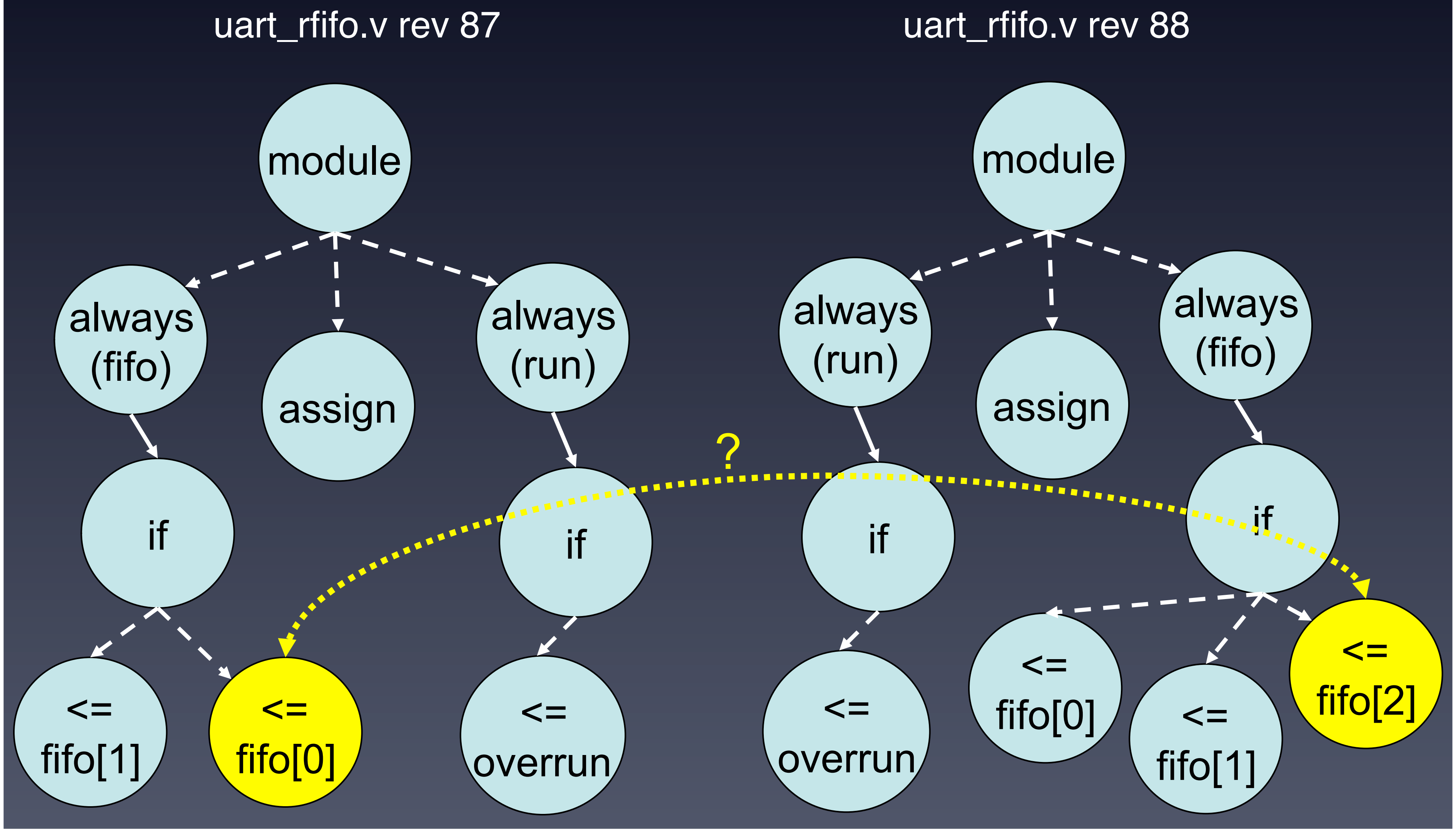
Tree Comparison



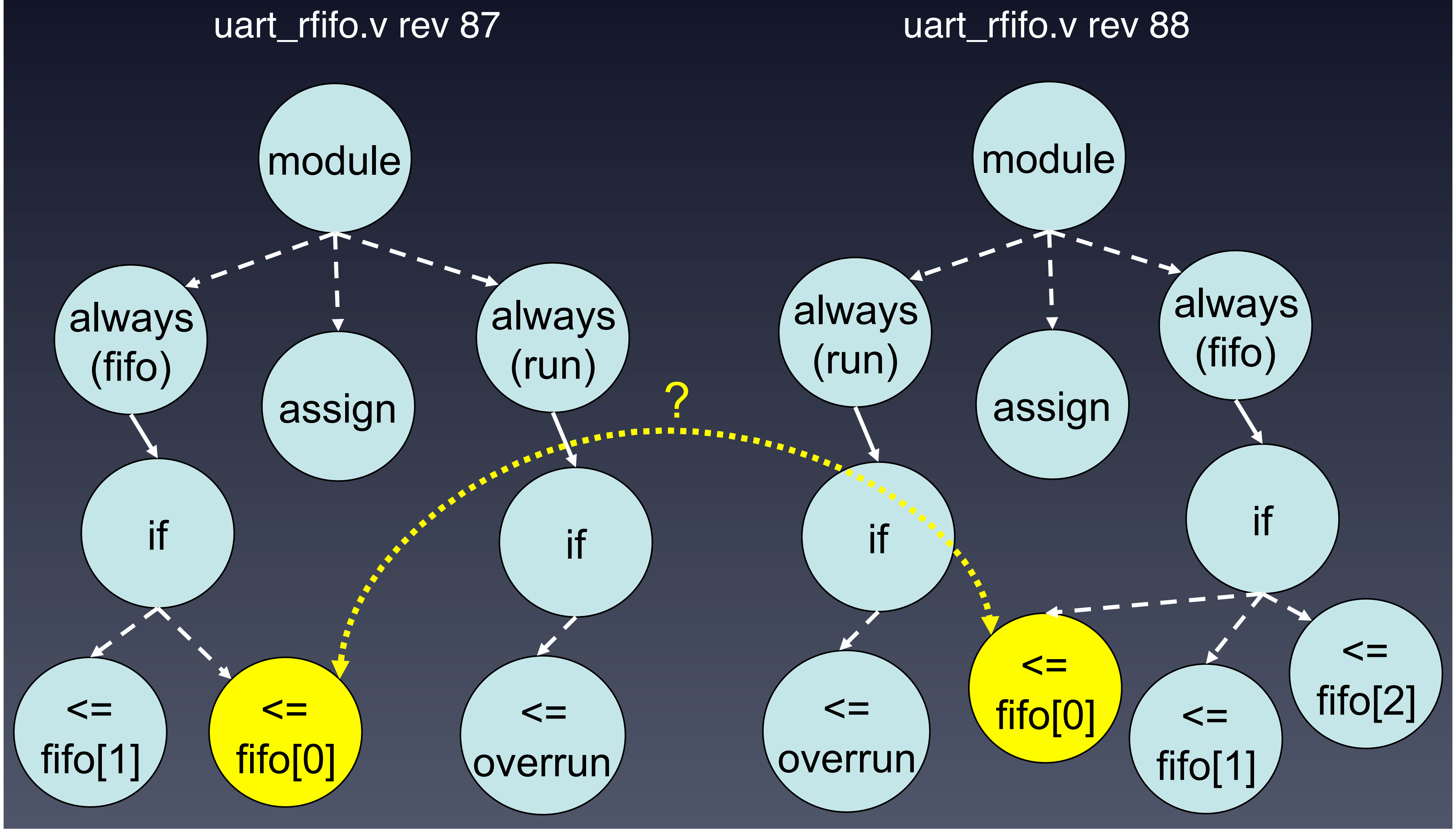
Tree Comparison



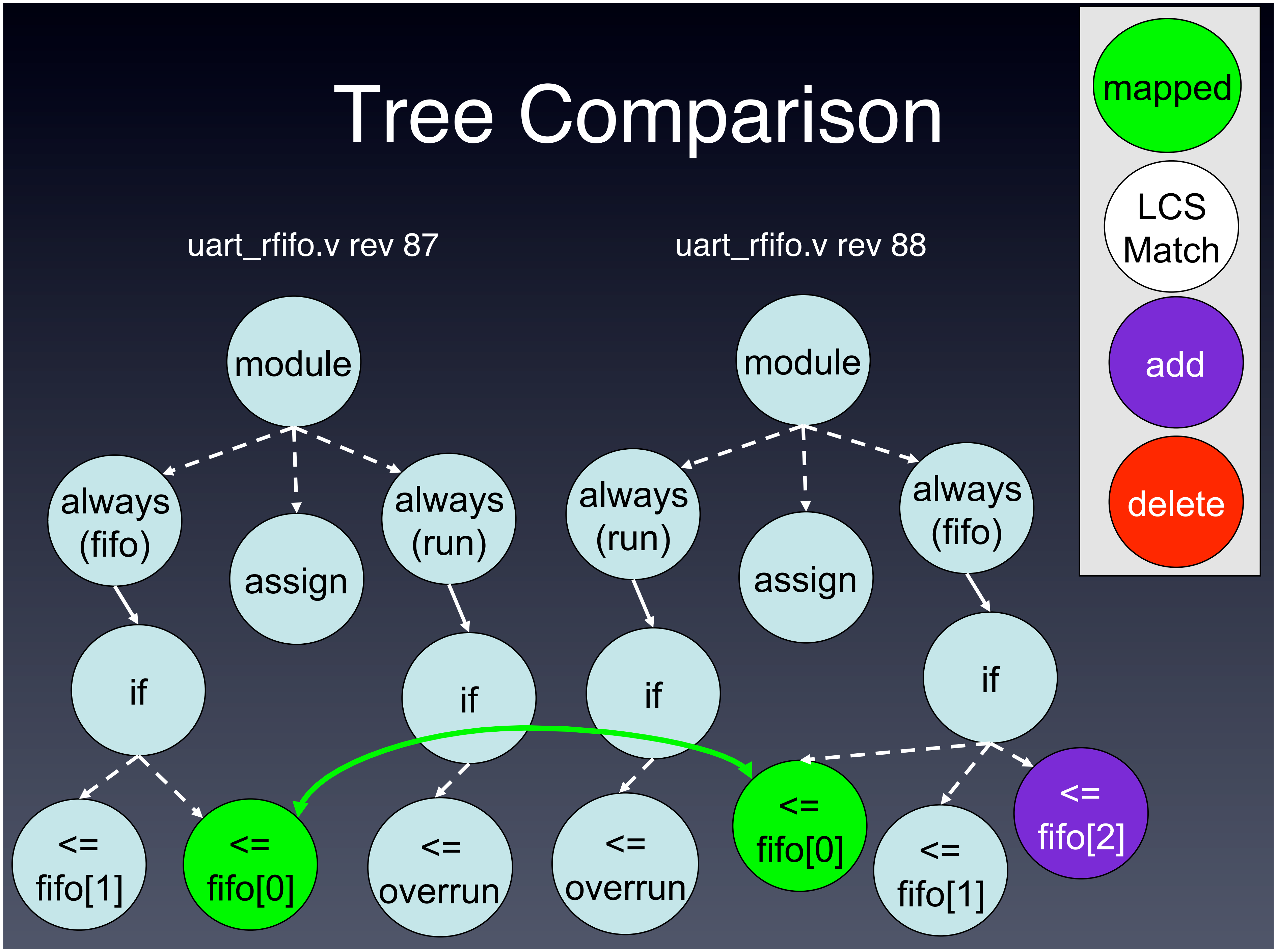
Tree Comparison



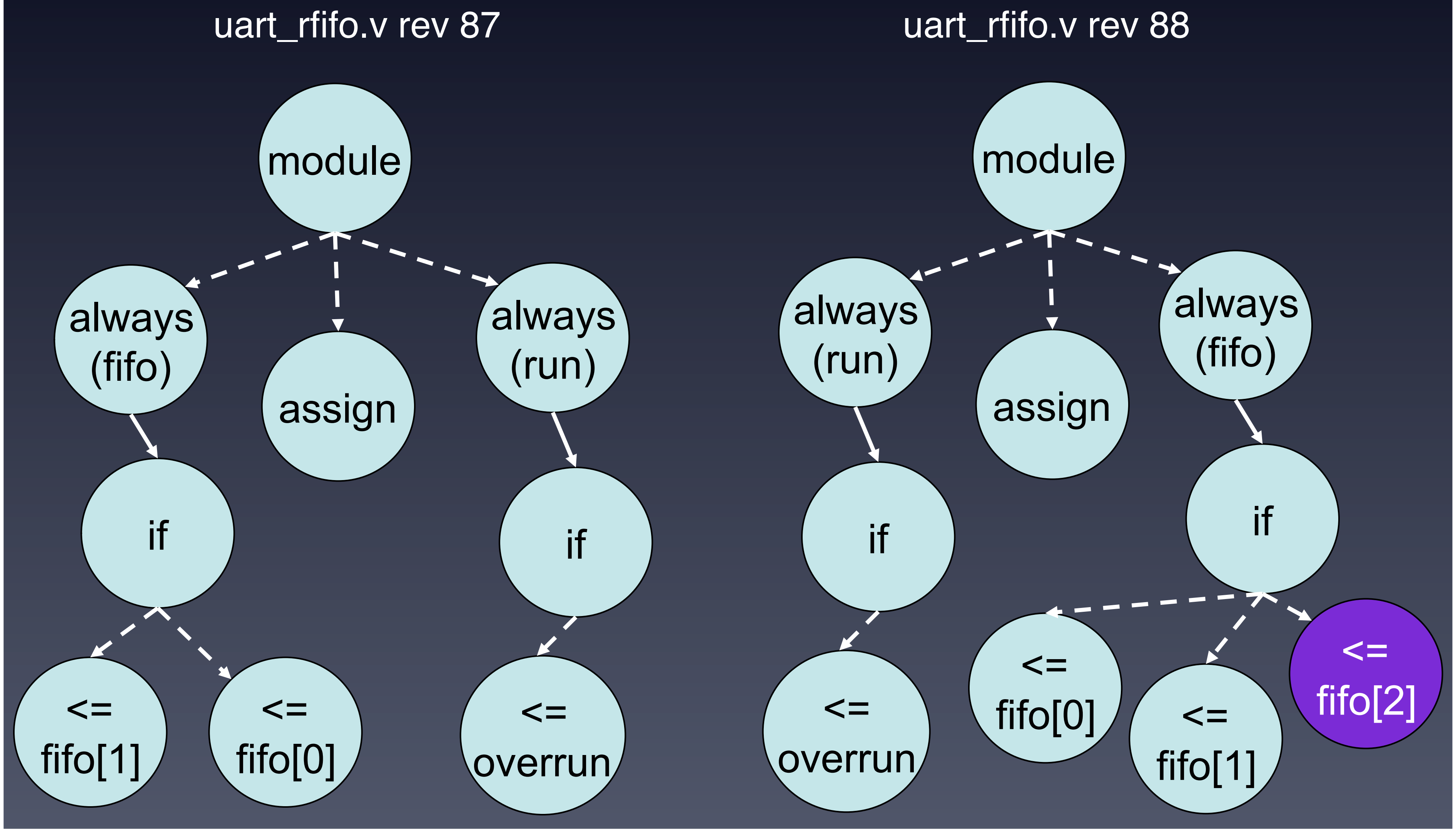
Tree Comparison



Tree Comparison

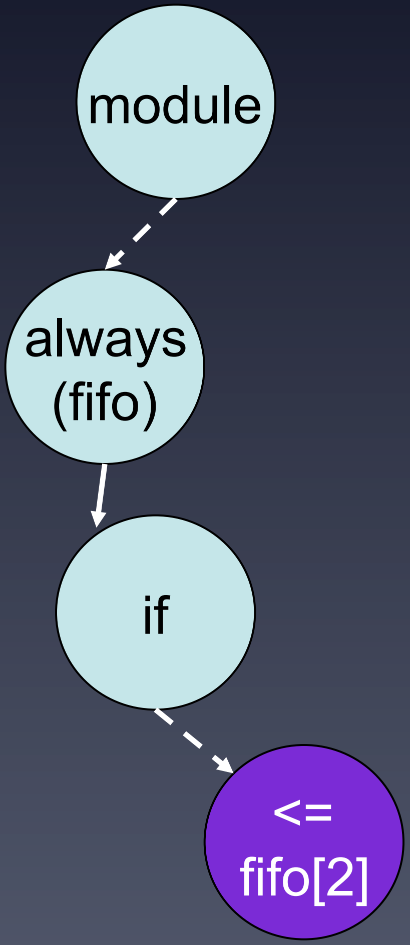


Tree Comparison



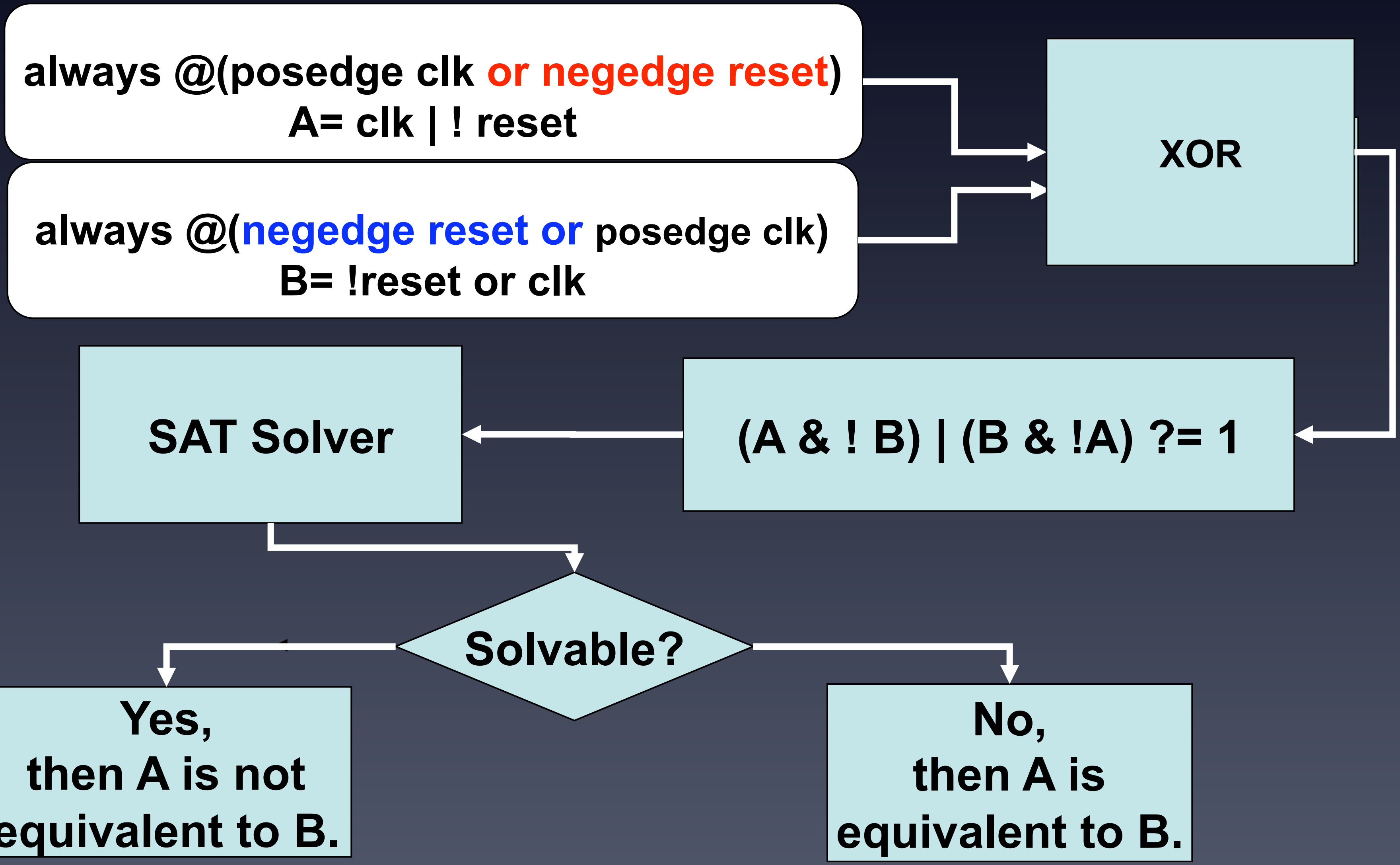
Resulting Syntactic Differences

uart_rfifo.v rev 87-88 diff



```
Line 221, NB_ADD,
A Non-Blocking
assignment has been added
```

Boolean Equivalence Check


always @(posedge clk or negedge reset)
A= clk | ! reset
always @(negedge reset or posedge clk)
B= !reset or clk
XOR
SAT Solver
(A & ! B) | (B & !A) ?= 1
Solvable?
Yes, then A is not equivalent to B.
No, then A is equivalent to B.

Change Type Classification

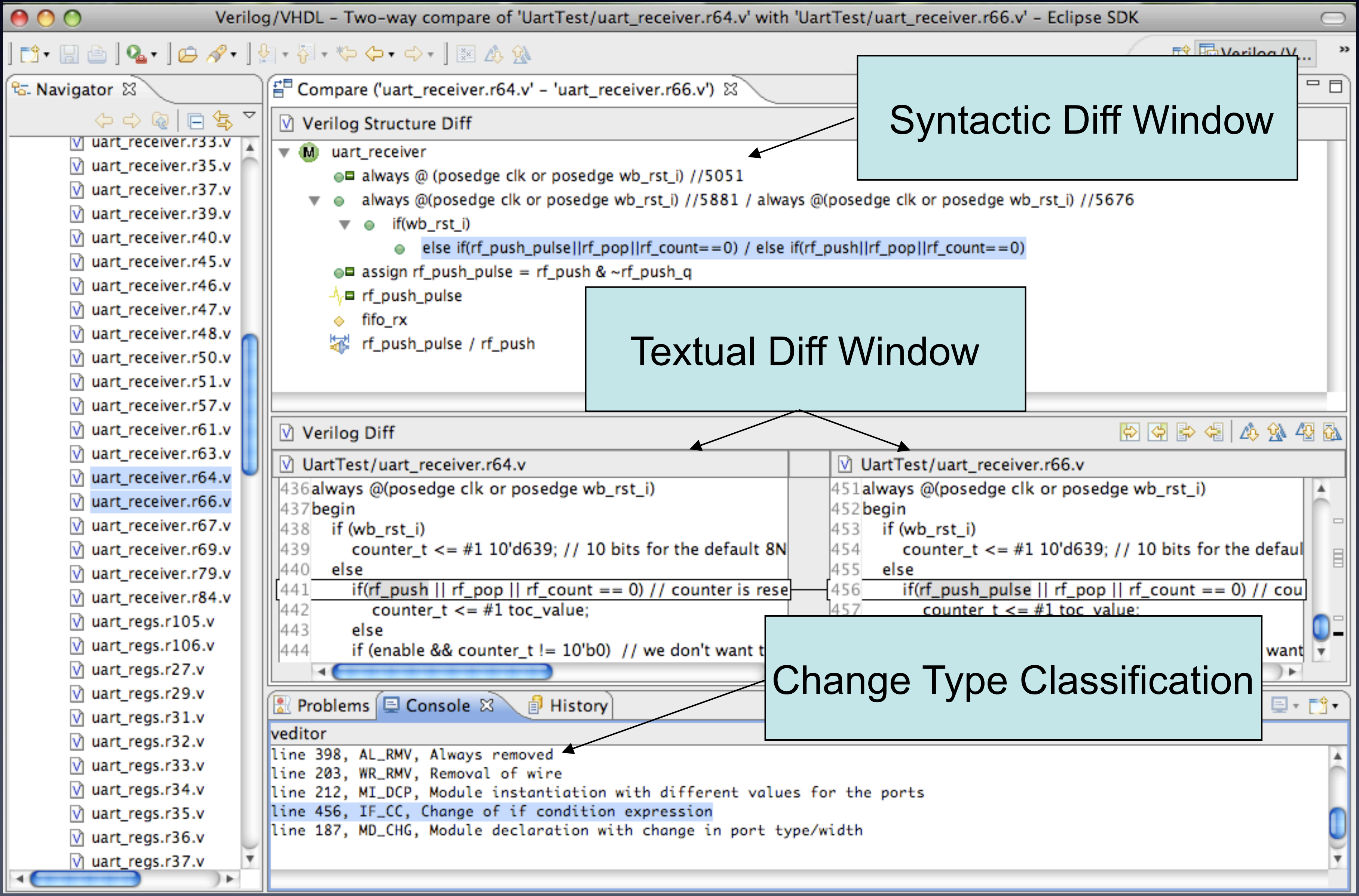
Syntactic Elements	Abbreviation	Description
Non-Blocking	NB_ADD	Non-Blocking assignment added
	NB_RMV	Non-Blocking assignment removed
	NB_CE	Non-Blocking assignment changed
Always	AL_ADD	Always block added
	AL_RMV	Always block removed
	AL_SE	Changes in the sensitivity list

Vdiff Tool Implementation

- Eclipse plugin based on open source *veditor* plug-in
- interfaces with Eclipse *compare* plug-in
- integrates with SVN (*subclipse* plug-in)

http://users.ece.utexas.edu/~miryung/software/Vdiff/web/index.html

Vdiff Eclipse Plugin


Verilog/VHDL – Two-way compare of 'UartTest/uart_receiver.r64.v' with 'UartTest/uart_receiver.r66.v' – Eclipse SDK
Navigator
Compare ('uart_receiver.r64.v' – 'uart_receiver.r66.v')
Verilog Structure Diff
Syntactic Diff Window
Textual Diff Window
Verilog Diff
Change Type Classification
Problems
Console
History
veditor

Navigator:

```
uart_receiver.r33.v
uart_receiver.r35.v
uart_receiver.r37.v
uart_receiver.r39.v
uart_receiver.r40.v
uart_receiver.r45.v
uart_receiver.r46.v
uart_receiver.r47.v
uart_receiver.r48.v
uart_receiver.r50.v
uart_receiver.r51.v
uart_receiver.r57.v
uart_receiver.r61.v
uart_receiver.r63.v
uart_receiver.r64.v
uart_receiver.r66.v
uart_receiver.r67.v
uart_receiver.r69.v
uart_receiver.r79.v
uart_receiver.r84.v
uart_regs.r105.v
uart_regs.r106.v
uart_regs.r27.v
uart_regs.r29.v
uart_regs.r31.v
uart_regs.r32.v
uart_regs.r33.v
uart_regs.r34.v
uart_regs.r35.v
uart_regs.r36.v
uart_regs.r37.v
```

Verilog Structure Diff:

```
uart_receiver
    always @ (posedge clk or posedge wb_rst_i) //5051
    always @(posedge clk or posedge wb_rst_i) //5881 / always @(posedge clk or posedge wb_rst_i) //5676
        if(wb_rst_i)
            else if(rf_push_pulse||rf_pop||rf_count==0) / else if(rf_push||rf_pop||rf_count==0)
    assign rf_push_pulse = rf_push & ~rf_push_q
    rf_push_pulse
    fifo_rx
    rf_push_pulse / rf_push
```

UartTest/uart_receiver.r64.v

```
436 always @(posedge clk or posedge wb_rst_i)
437 begin
438     if (wb_rst_i)
439         counter_t <= #1 10'd639; // 10 bits for the default 8N
440     else
441         if(rf_push || rf_pop || rf_count == 0) // counter is rese
442             counter_t <= #1 toc_value;
443         else
444         if (enable && counter_t != 10'b0)  // we don't want t
```

UartTest/uart_receiver.r66.v

```
451 always @(posedge clk or posedge wb_rst_i)
452 begin
453     if (wb_rst_i)
454         counter_t <= #1 10'd639; // 10 bits for the defaul
455     else
456         if(rf_push_pulse || rf_pop || rf_count == 0) // cou
457             counter_t <= #1 toc_value;
                                                      want
```

Console:

```
line 398, AL_RMV, Always removed
line 203, WR_RMV, Removal of wire
line 212, MI_DCP, Module instantiation with different values for the ports
line 456, IF_CC, Change of if condition expression
line 187, MD_CHG, Module declaration with change in port type/width
```

Outline

- Motivation
- Verilog Background
- Vdiff Algorithm
- **Evaluation**
- Conclusions

Evaluation

- compare Vdiff's results with **manual differencing results**—the first two authors manually inspected the revisions
- subjects
  - UART16550 (opencores.org)
  - RAMP GateLib DRAM controller (ramp.eecs.berkeley.edu)
- Criteria: precision & recall

Results

Project	File Revisions	Eval	Vdiff	$|V \cap E|$	Precision	Recall
Total (UART)	141	600	601	586	97.5%	97.7%
Total (GateLib)	69	497	502	482	96.2%	96.9%
Total	210	1097	1103	1068	96.8%	97.3%

We evaluated 210 Verilog file revisions with more then 1000 differences across two different projects.

Findings

- Vdiff matches position-independent constructs very well
- Vdiff struggles on line edits with low text similarity
- Feedback from a logic designer at IBM:

  "I can see a use for [the change-types] right away. It would be great for team leads because they could look at this log of changes and understand what has changed between versions without having to look at the files [textual differences]."

Example: Expected Results

```
/* Old */

/* If Condition Changed */
if (!srx_pad_i)
      counter_b <= 0x191;
```

```
/* New */
/* If Condition Changed */
if (!srx_pad_i || rstate == sr_idle)
      counter_b <= 0x191;
```

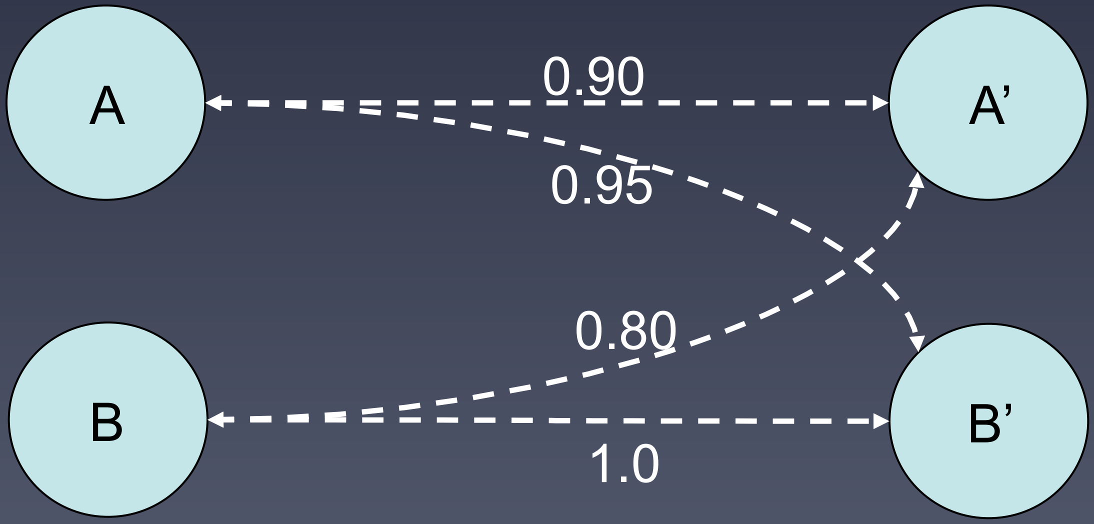
Example: Vdiff Results

```
/* Old */

/* IF Block Removed*/
if (!srx_pad_i)
      counter_b <= 0x191;
```

```
/* New */

/* IF Block Added */
if (!srx_pad_i || rstate == sr_idle)
      counter_b <= 0x191;
```

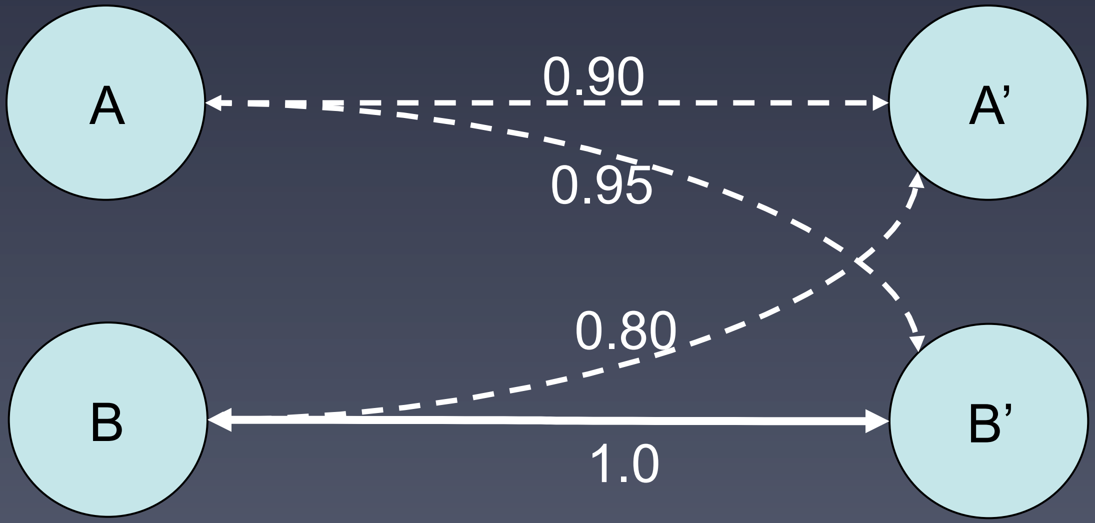
Comparison of AST Matching Algorithms

- Exact matching [Neamtiu 2005]
- In-order matching [Cottrell 2007]
- Greedy weighted bipartite matching [Vdiff]



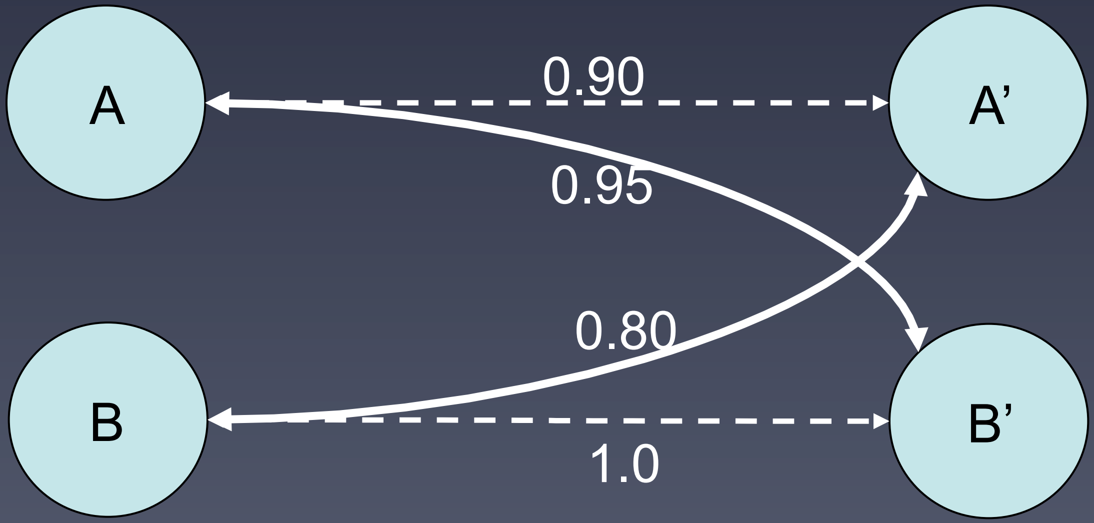
Comparison of AST Matching Algorithms

- **Exact matching [Neamtiu 2005]**
- In-order matching [Cottrell 2007]
- Greedy weighted bipartite matching [Vdiff]



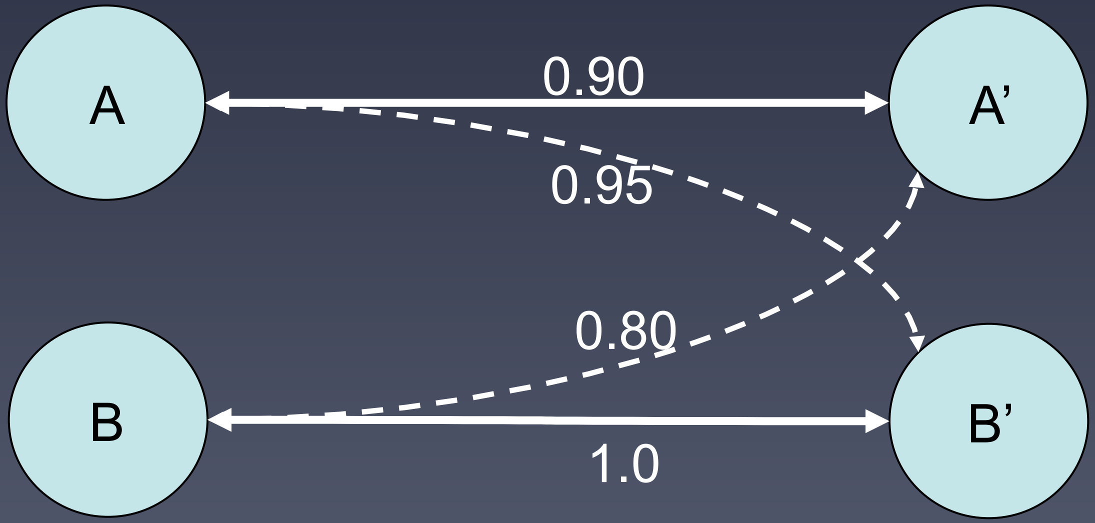
Comparison of AST Matching Algorithms

- Exact matching [Neamtiu 2005]
- **In-order matching [Cottrell 2007]**
- Greedy weighted bipartite matching [Vdiff]



Comparison of AST Matching Algorithms

- Exact matching [Neamtiu 2005]
- In-order matching [Cottrell 2007]
- **Greedy weighted bipartite matching [Vdiff]**



Comparison of AST Matching Algorithms

- Exact matching [Neamtiu 2005]
- In-order matching [Cottrell 2007]
- Greedy weighted bipartite matching [Vdiff]

Average	Exact Match	In-Order Match	Weighted Bipartite
Precision	56.1%	90.9%	97.5%
Recall	67.9%	91.8%	97.7%

Comparison of AST Matching Algorithms

1097 differences from 210 file revisions in 2 real world projects shows that *the ordering of code actually matters in practice* when it comes to computing differences.

Average	Exact Match	In-Order Match	Weighted Bipartite
Precision	56.1%	90.9%	97.5%
Recall	67.9%	91.8%	97.7%

Comparison with General Model Differencing Framework

- EMF [Eclipse EMF compare project]
  - Mapped (1) modules to classes, (2) always blocks and continuous assignments to operations, (3) wires, registers, and ports to fields, and (4) modular instantiations to reference pointers in an EMF ecore model.
  - Results (Recall=47%, Precision=80%) shows a need to expand the Ecore model to be able to handle specific concurrency constructs and non-unique identifiers.
- Sidiff [Treude et al., 2007, Schmidt and Gloetzner, 2008]
  - At the time of our evaluation Sidiff did not provide APIs to allow us to map Verilog language constructs to their general differencing algorithms.

Discussion

- Vdiff is sensitive to subtle changes to variable names and IF-conditions
  - Further investigation of different name similarity measures is required
- renaming of wires, registers, and modules caused false positives
- Vdiff’s algorithm currently cannot recover from node mismatches
- equivalence check using a SAT solver is limited to sensitivity lists

Outline

- Motivation
- Verilog Background
- Vdiff Algorithm
- Evaluation
- **Conclusions**

Related Work

- **Syntactic program differencing**
  - [Yang 1992, Neamtiu et al. 2005, Fluri et al. 2007, Cottrell et al. 2007, Raghavan et al. 2004, etc.]
  - Vdiff is similar to these but identifies syntactic differences robustly even when multiple AST nodes have similar labels and when they are reordered.
- **Model differencing**
  - UMLdiff [Xing and Stroulia 2005], Sidiff [Kelter et al.] and EMF [Eclipse EMF]
- **Change types**
  - Change Distiller [Fluri et al. 2007]
  - Verilog change types [Sudakrishnan et al. 2009]
- **Differential symbolic execution** [Person et al. 2008]

Conclusions

- *Vdiff* is a position-independent differencing algorithm designed for hardware design descriptions
  - – computes syntactic differences with high recall (96.8%) and high precision (97.3%)
  - – classifies differences in terms of Verilog specific change types
  - – can enable analysis of evolving hardware design

Acknowledgment

Vdiff website:

http://www.ece.utexas.edu/~miryung/software/Vdiff/web/index.html

The authors thank Greg Gibeling and Dr. Derek Chiou for providing accesses to the RAMP repository and Dr. Adnan Aziz and anonymous reviewers for their detailed comments on our draft.

Questions

?

Backup