

*Development Environment for
Language Processors based on
Attribute Grammars*

Akira Sasaki

2004/01/13

Abstract

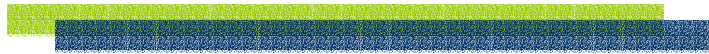


- Attribute grammar (AG) based development environment for language processors
 - Circular AGs with Remote References
 - Systematic Debugging Methods for AGs

Contents

- Overview
- Attribute Grammars
- Circular AGs with Remote References
- Systematic Debugging Methods for AGs
- Conclusion

Overview

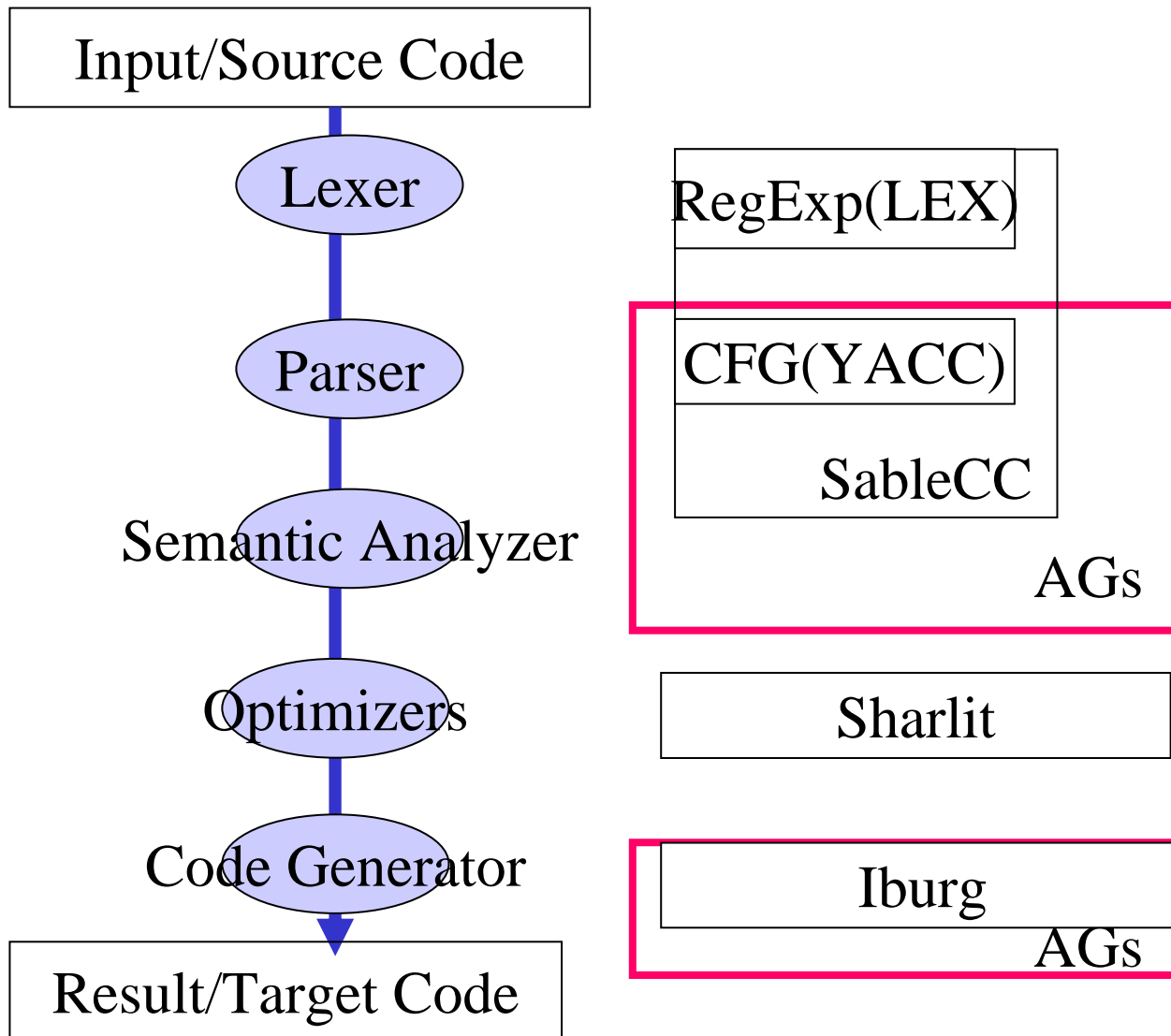


Background



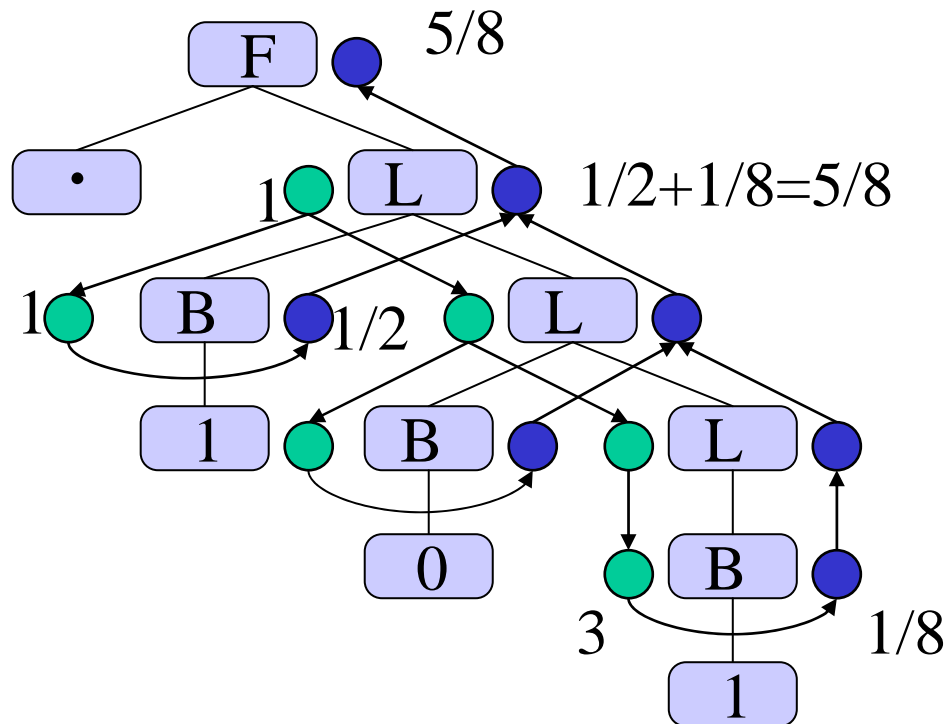
- Language processing is the basis of application software
- Language processing is complex
 - Modules should treat a large data with complex structures
- Development tools for language processors based on formalisms
 - lex, yacc

Formalizations and Tools for Language Processing



Attribute Grammars

- Integration of syntax and semantics of languages
 - CFG + Semantic Rules



$$F \Rightarrow \cdot L$$

$$\{ L.pos = 1 ;$$

$$F.val = L.val \}$$

$$L_0 \Rightarrow B L_1$$

$$\{ L_1.pos = L_0.pos + 1 ;$$

$$B.pos = L_0.pos ;$$

$$L_0.val = B.val + L_1.val \}$$

$$| B$$

$$\{ B.pos = L_0.pos + 1 ;$$

$$L_0.val = B.val \}$$

$$B \Rightarrow 1$$

$$\{ B.val = 2^{-B.pos} \}$$

$$| 0$$

$$\{ B.val = 0 \}$$

Motivation: Features of AG

- Advantages
 - Simplicity
 - Attribute Evaluators
 - Maturity of Theories
- Disadvantages
 - Poor expressive power
 - Difficulty of debugging

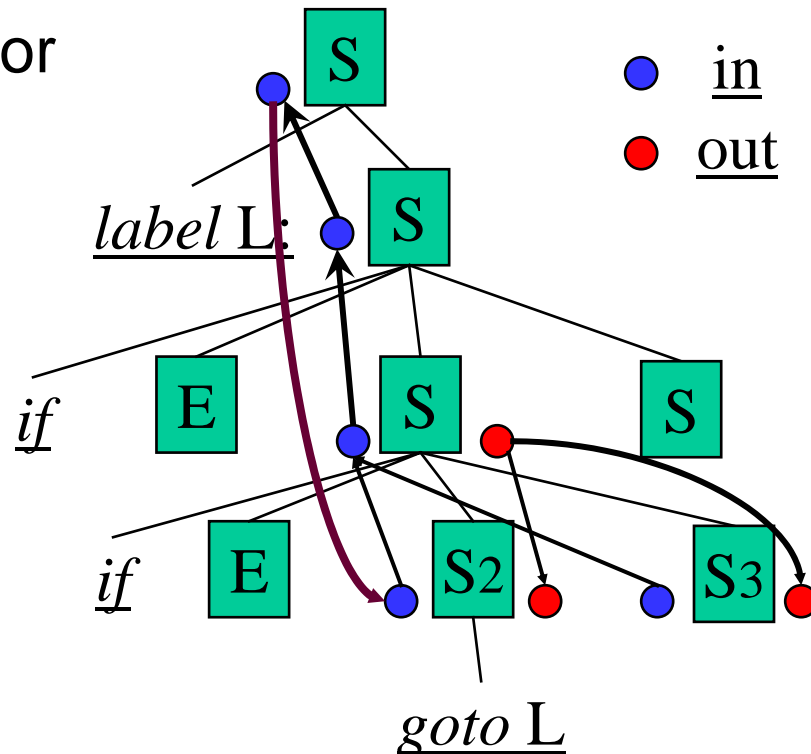
Contributions



- (1) Circular AGs with remote references
- (2) Systematic Debugging Methods

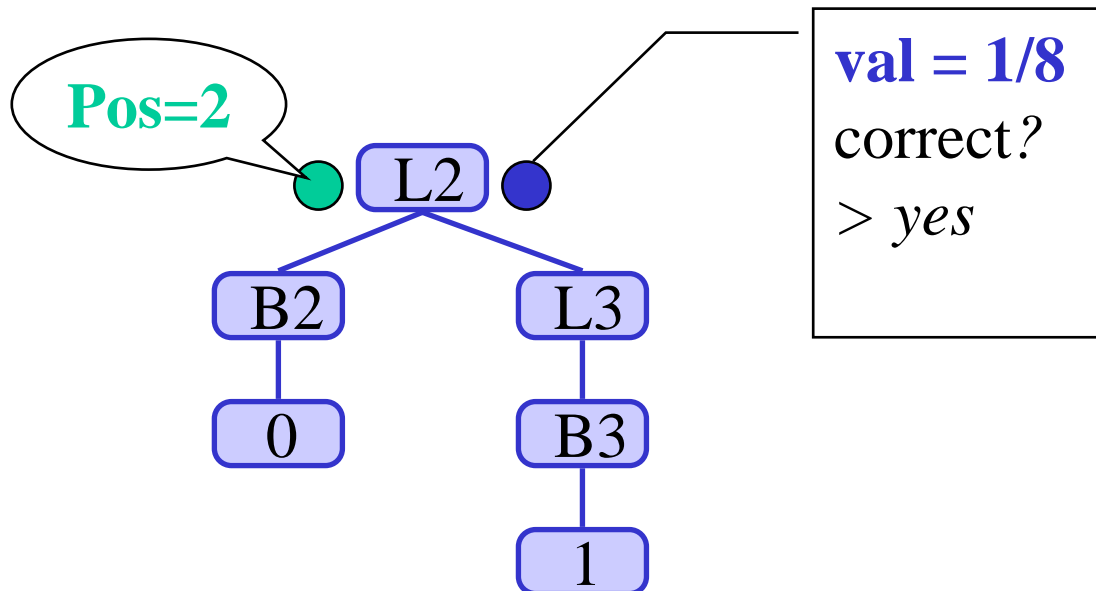
(1) Circular AGs with remote references

- CRAG: AG extension that allows both of:
 - Circularity
 - Remote Attribute References
- Efficient attribute evaluators for CRAGs
 - Static evaluator

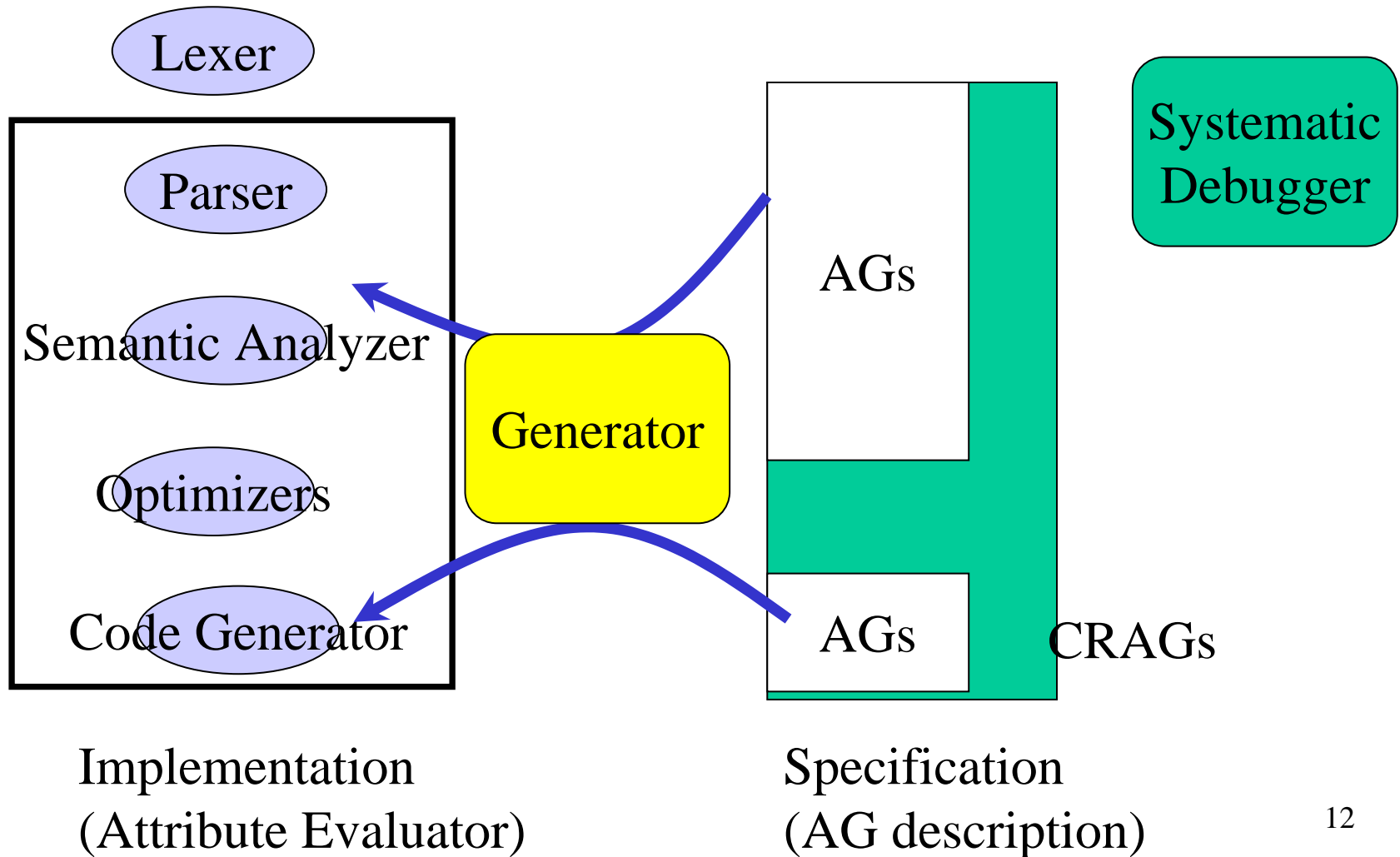


(2) Systematic debugging methods for AGs

- Generalization of Systematic Debugging Methods of AGs
 - Unified Theories
- Integration of previous methods



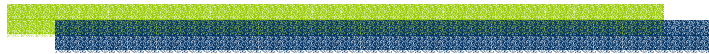
Development Environment based on AGs



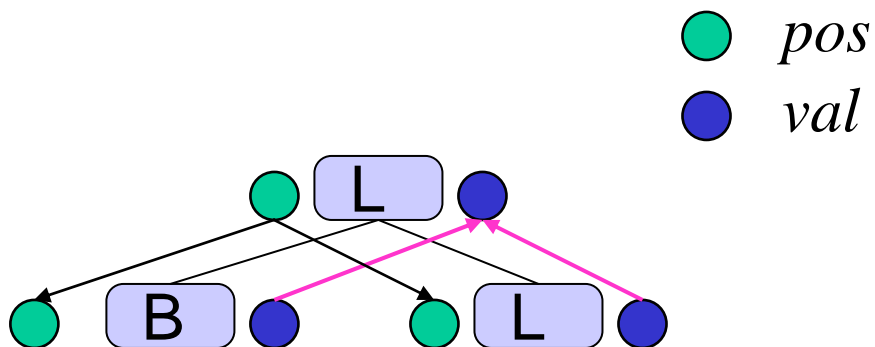
Contents

- Overview
- **Attribute Grammars**
- Circular AGs with Remote References
- Systematic Debugging Methods for AGs
- Conclusion

Attribute Grammars



Attribute Grammar: Example (value of binary number)



$$F \Rightarrow \cdot L$$

$$\{ L.pos = 1 ;$$

$$F.val = L.val \}$$

$$L_0 \Rightarrow B L_1$$

$$\{ L_1.pos = L_0.pos + 1 ;$$

$$B.pos = L_0.pos ;$$

$$L_0.val = B.val + L_1.val \}$$

| B

$$\{ B.pos = L_0.pos + 1 ;$$

$$L_0.val = B.val \}$$

$$B \Rightarrow 1$$

$$\{ B.val = 2^{-B.pos} \}$$

| 0

$$\{ B.val = 0 \}$$



Contents

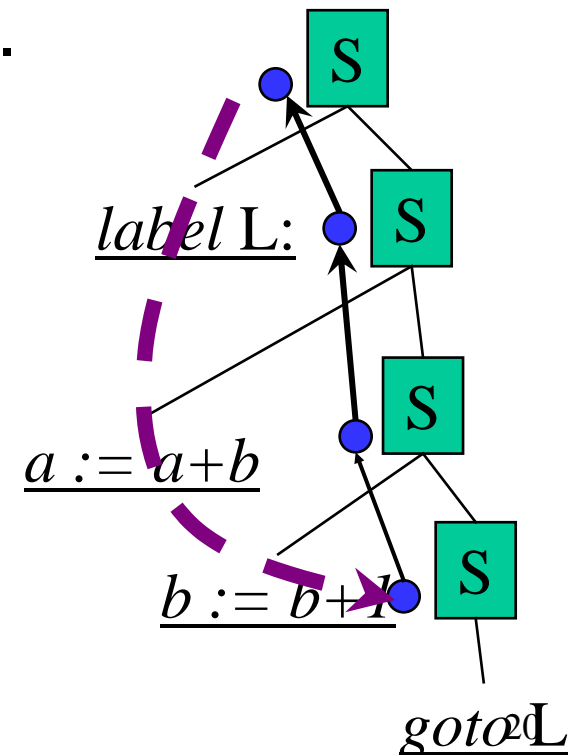
- Overview
- Attribute Grammars
- **Circular AGs with Remote References**
- Systematic Debugging Methods for AGs
- Conclusion

Circular AGs with Remote References

Two horizontal bars are positioned below the title. The top bar is a solid light green color, and the bottom bar is a solid dark blue color. Both bars are of the same length and are slightly offset to the left.

Background

- Difficulty in Compiler Construction using AG
 - goto-label, break, continue,
 - symbol table access
 - Definition-use
- Restrictions in AGs prohibit....
 - Reference to an attribute in a distant node
 - distant node relation
 - Circular attribute dependency
 - recursive definition (equation)

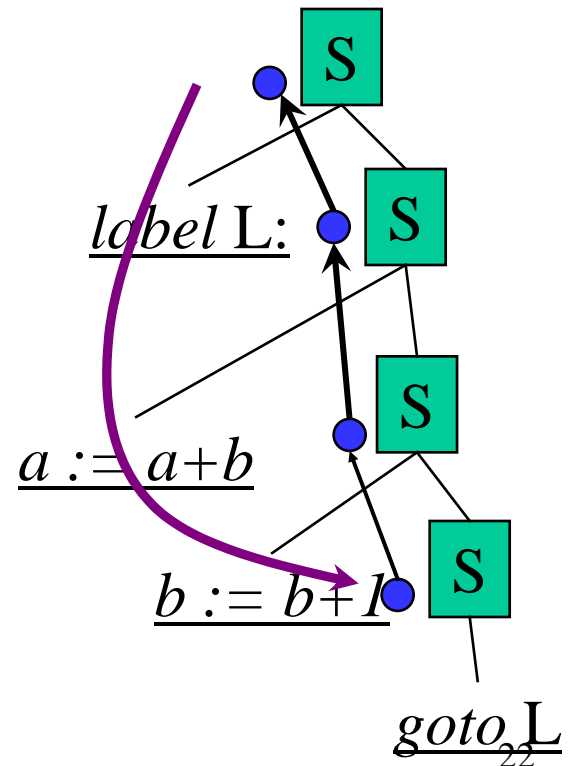


Extensions for AGs

- Previous work
 - (1) Remote Attribute References
 - [Kastens82],[Reps84],[Hedin99]
 - (2) Circularity in Attribute Dependence
 - [Farrow86], [Jones90]
- Circular and Remote AG (CRAG)
simultaneously allows (1) & (2)

Evaluator For Circular AGs

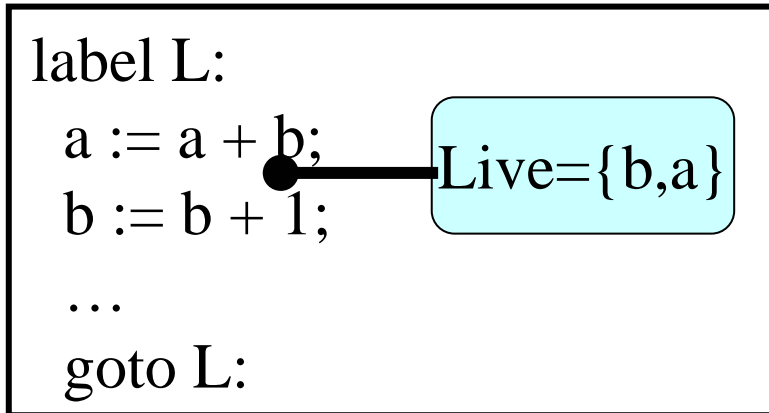
- Previous work
 - Static Evaluator [Farrow86]
 - Dynamic Evaluator [Jones90]
- Efficient evaluator for CRAG
 - Static evaluator
 - Mostly static evaluator



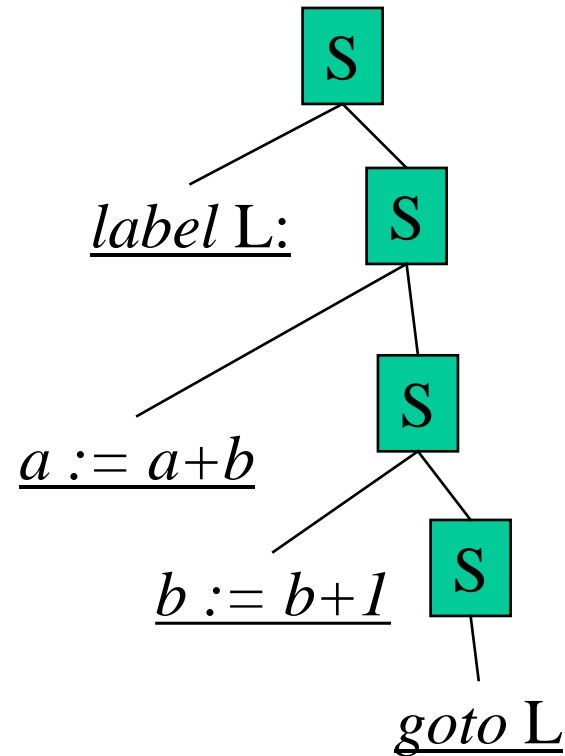
CRAG: circular AG with remote references

CRAG example: live variable analysis

source program:

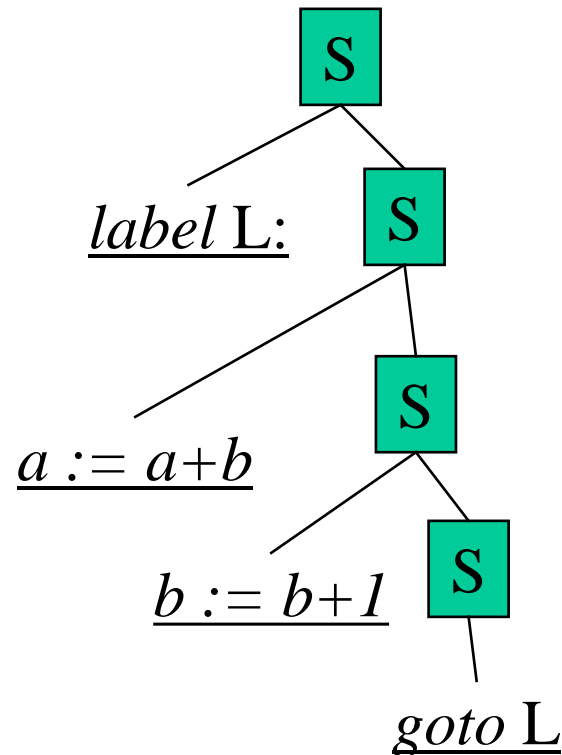


derivation tree:



CRAG example: live variable analysis

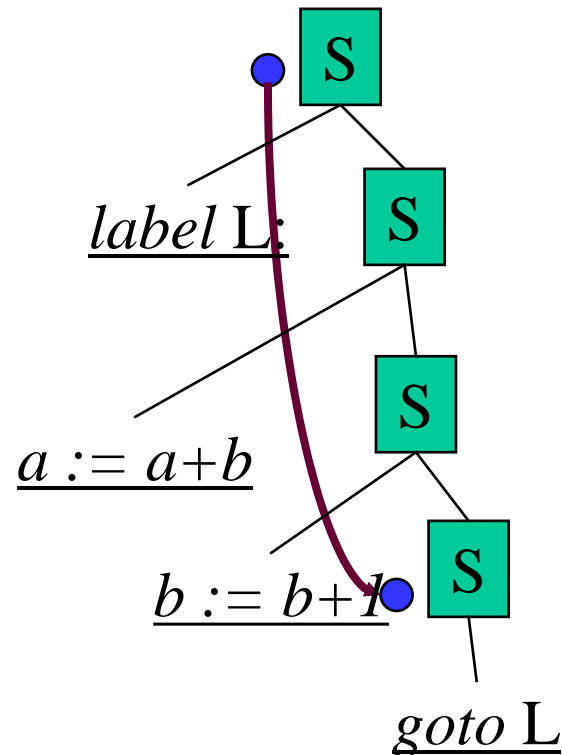
```
S ::= label lname : S
{ S2.out = S.out;
  S.in = S2.in; }
S ::= goto lname
{ S.in = ??? }
```



The live variables at “goto” is equal to the live variables at “label”.

CRAG example: live variable analysis

```
S ::= label lname : S %prod lab
{ S2.out = S.out;
  S.in = S2.in; }
S ::= goto lname %ref lab
{ S.in = S.in@lab }
```



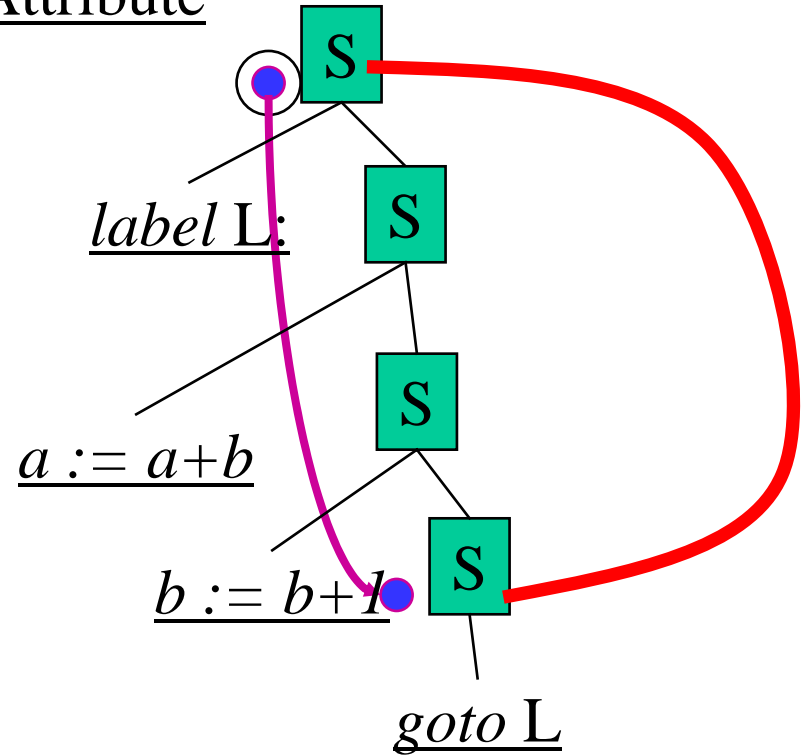
- CRAG allows **remote attribute references** to distant nodes

Remote Attribute Reference

Remote attributes are **referred** through **links** between two nodes

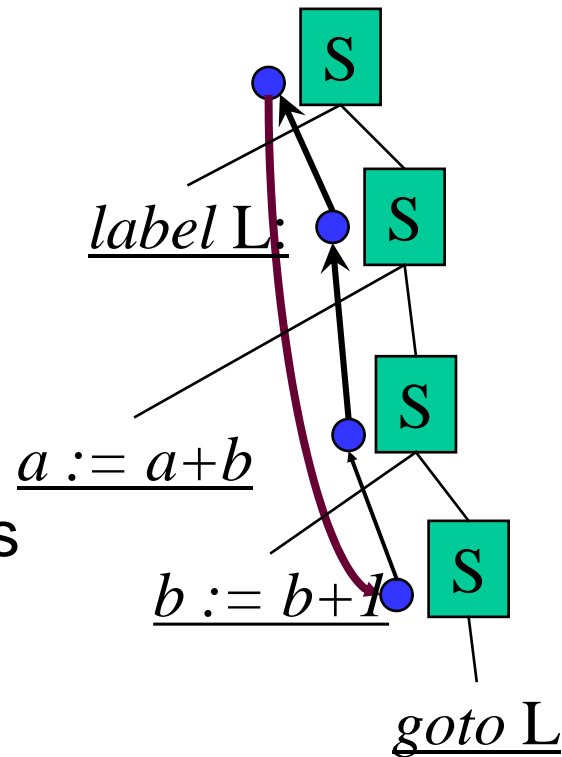
```
S ::= label lname : S %prod lab  
{ S2.out = S.out;  
  S.in = S2.in; }  
S ::= goto lname %ref lab  
{ S.in = S.in@lab }
```

Remote Attribute



CRAG example: live variable analysis

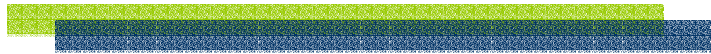
- Remote attribute references causes circularity
- CRAG allows **circular defined attributes**
 - under appropriate conditions



Conditions for convergence[Farrow86]

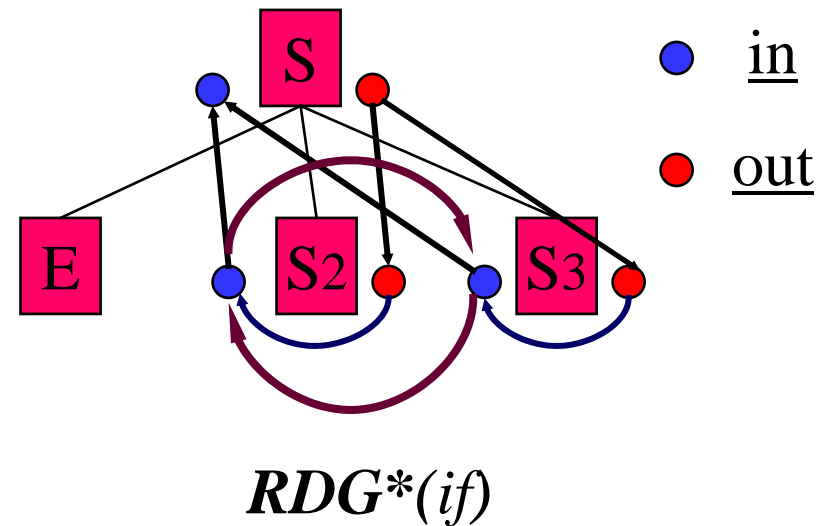
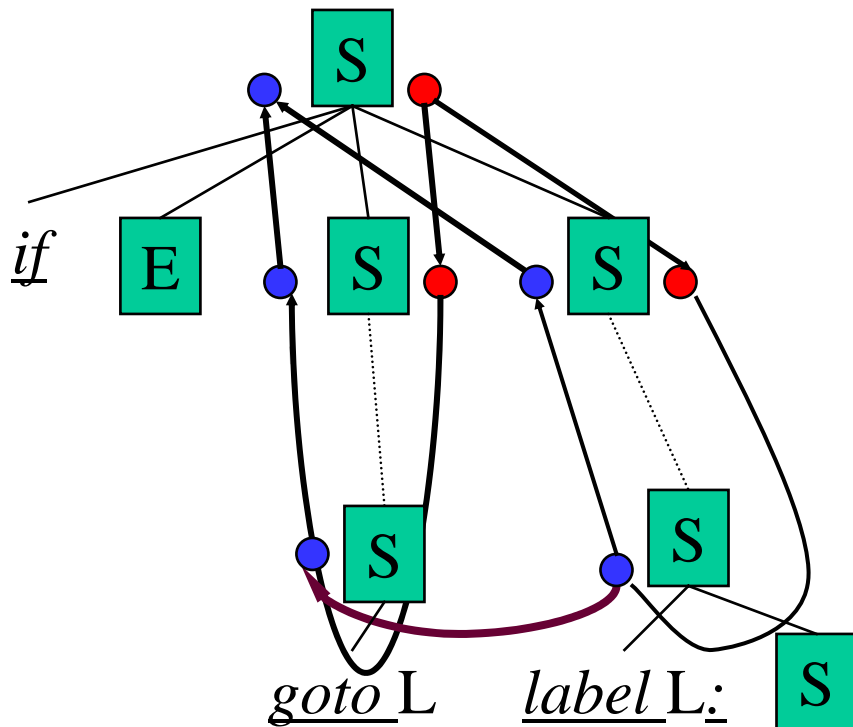
- An AG is a finitely recursive AG *iff*
 - The domain of circularly defined attribute values constitutes a c.p.o., in which it is possible to test pair for equality
 - Each function f associated with them is monotonic s.t. for $s[0] < s[1] < \dots$
 $f(s[0]) < f(s[1]) \dots < f(s[k]) = f(s[k+1])$
- Applicable to AGs with remote references
CRAG

Efficient evaluators for CRAG



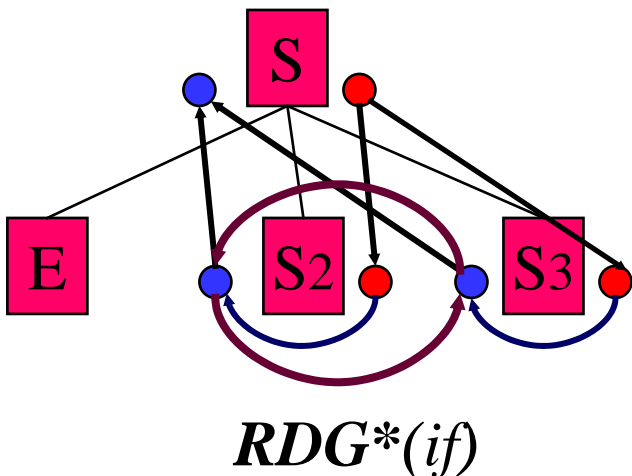
Generation of static evaluator for CRAG

- $RDG^*(p)$: augmented remote dependency graph
 - indirect remote dependency edges



Evaluator for CRAG

- Evaluation of circularly defined attributes
 - Iterative evaluation until convergence



case if:

```
S3.out = S.out;
```

```
S2.out = S.out;
```

```
initialize attributes in the cycle;
```

```
do {
```

```
  S3.in = funcS.in(S3.out, nthSub(3,Tree));
```

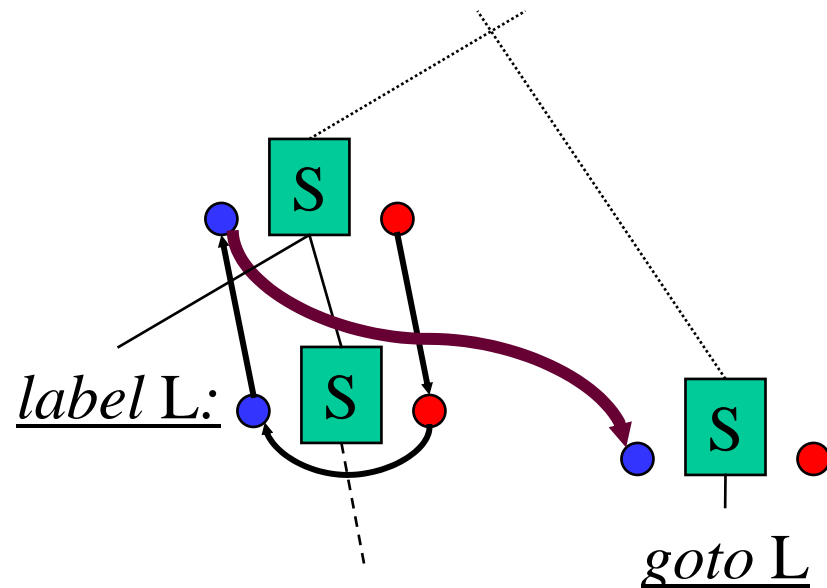
```
  S2.in = funcS.in(S2.out, nthSub(2,Tree));
```

```
} while (!converge);
```

```
S.in = union(S2.in, S3.in, E.use);
```

Remote References

- Remotely referenced attributes
 - values are stored into a cache
- Remote attribute references
 - Refer remote attribute values in the cache



case label:

```
S2.out = S.out;
```

```
S2.in = funcS.in(S2.out,  
                nthSub(1,Tree));
```

```
S.in = S2.in;
```

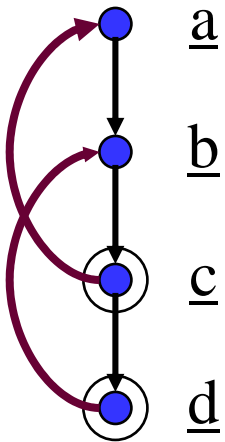
```
saveAttr(Tree, "S.in", S.in);
```

case goto:

```
S.in = refAttr(Tree, "label", "S.in");
```

Attributes for checking termination of iterations

- Only **remote attributes** in the cycle (provided that remote attributes cause the cycle)

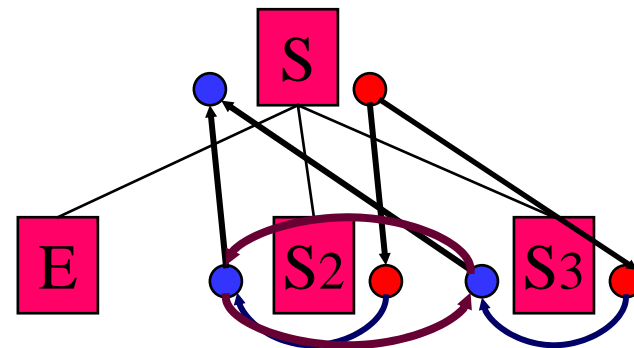


```
initialize;  
do{  
  a=Fa(c);  
  b=Fb(a,d);  
  c=Fc(b);  
  d=Fd(c);  
}while notConvergent;
```

Values at the k -th iteration are determined from remote attribute values at the $(k-1)$ th iteration

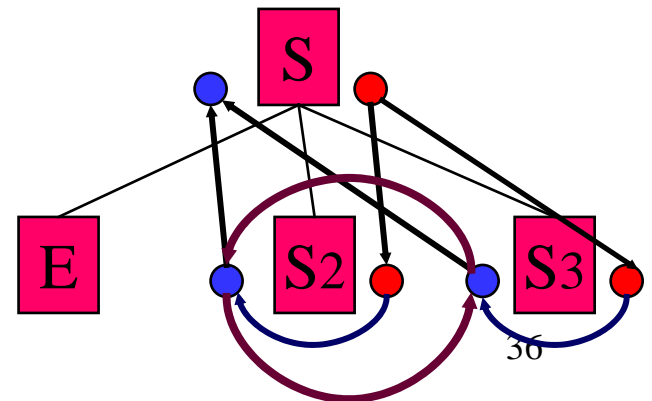
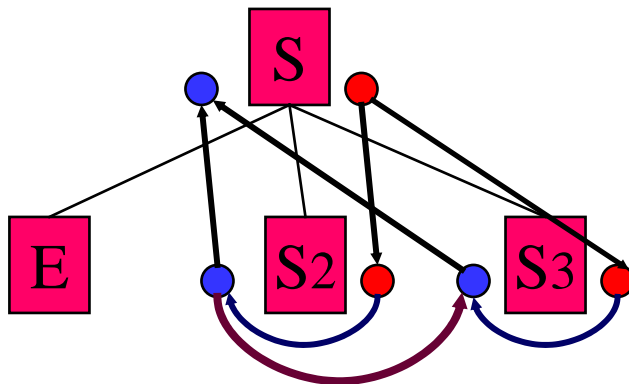
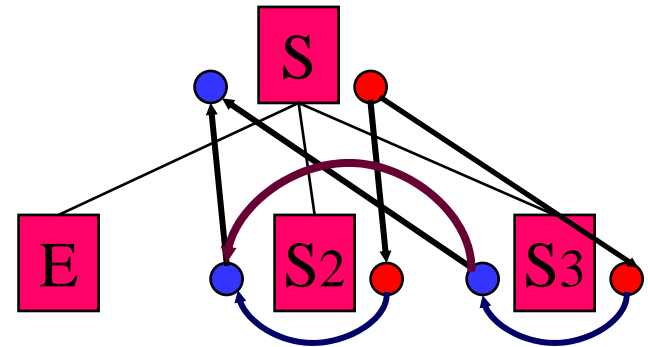
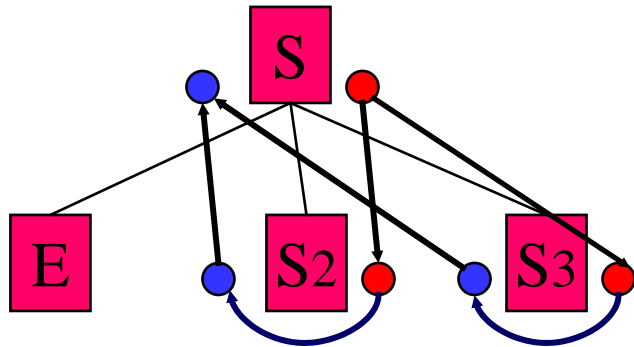
Revision of Static Evaluator: Mostly Static Evaluators

- Rough estimation of remote dependency edges in static evaluators
 - indirect remote dependency edges are unioned
 - causes unnecessary iterative evaluation
- Removal of unnecessary cycles needed



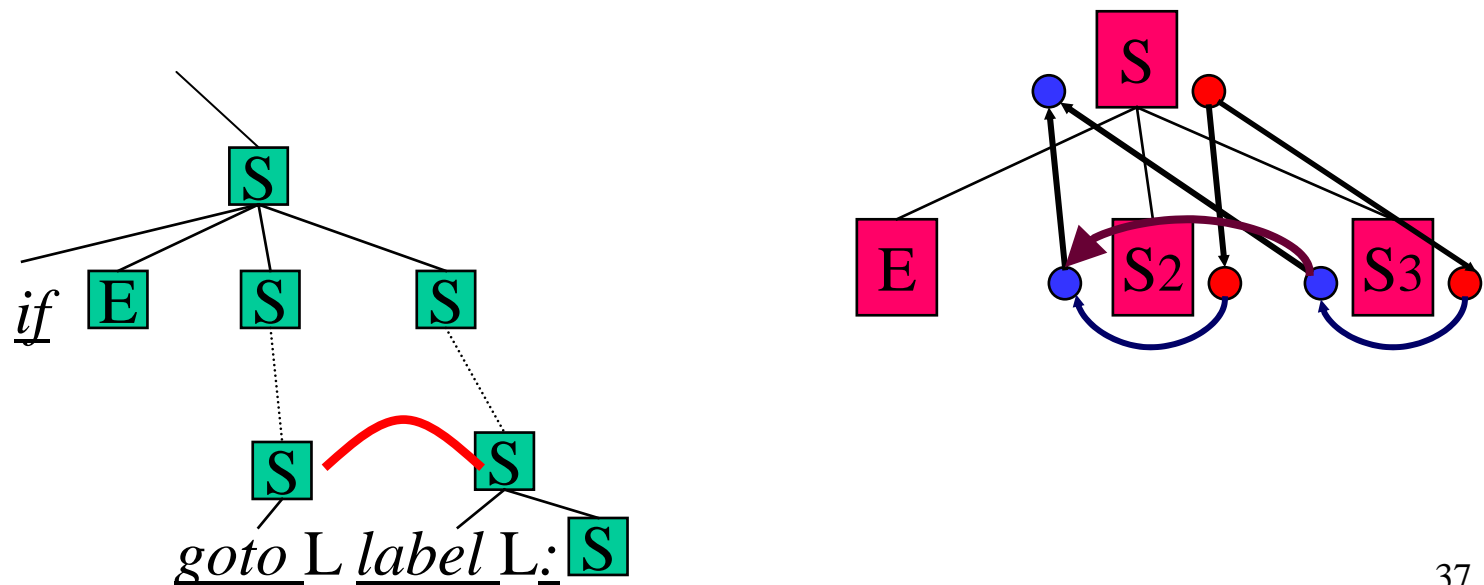
Division of unioned edges

- Selection of an evaluation order by how links are actually connected



Attribute Evaluation by Mostly Static Evaluator

- Link analysis phase (preprocess)
 - analyze where links are connected
- Attribute Evaluation
 - Appropriate routines are selected



Comparison of Evaluators

Liveness Analysis

	Graph	SCC	EvalD	Dynamic	Static	Edge	EvalM	MStatic
<i>Loop10</i>	6.5	8.5	6.0	21.0	6.0	1.0	4.0	5.0
<i>Loop30</i>	6.0	8.5	7.5	22.0	6.5	1.0	4.5	5.5
<i>Loop50</i>	6.0	9.5	8.0	23.5	6.5	1.0	5.0	6.0
<i>Loop70</i>	6.0	9.5	8.5	24.0	6.5	1.5	5.5	7.0
<i>Loop90</i>	6.0	10.0	10.0	26.0	6.5	1.5	6.0	7.5
<i>LoopSeq</i>	6.5	12.0	11.0	29.5	9.5	1.5	8.5	10.0
<i>Nest2</i>	6.0	10.0	8.5	24.5	7.0	1.0	5.5	6.5
<i>Nest3</i>	6.0	11.0	8.5	25.5	7.5	1.0	5.5	6.5

(CMU Common Lisp on UltraSparcIli)

Conclusion

- Circular Remote Reference AGs
 - Remote attribute references
 - Circularly defined attributes
- Static evaluator for CRAG
 - Effective iterative evaluation algorithm
 - Selection of appropriate evaluation orders

Contents

- Overview
- Attribute Grammars
- Circular AGs with Remote References
- **Systematic Debugging Methods for AGs**
- Conclusion

Systematic Debugging Methods for Attribute Grammars

Two horizontal bars are positioned below the title. The top bar is a solid light green color, and the bottom bar is a solid dark blue color. Both bars are of the same length and are partially offset to the left.

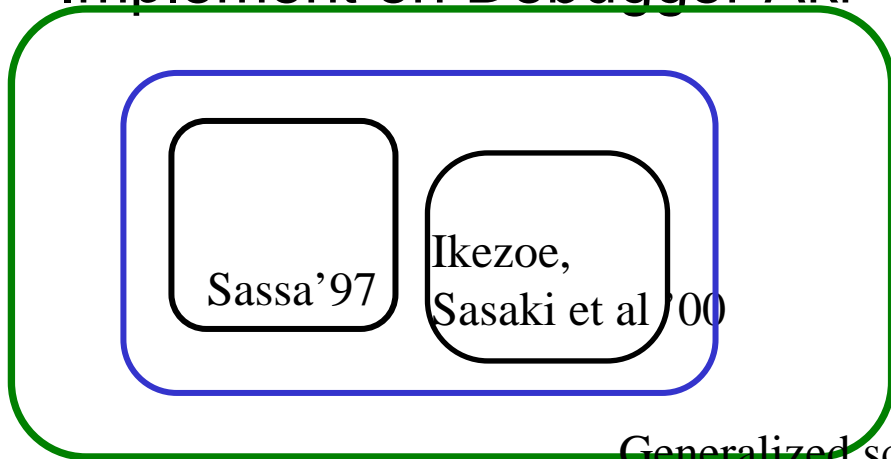
Background



- Attribute Grammars
 - Simple, Declarative
- Difficulty in debugging of AG description
 - Complex attribute dependency, Recursive structure of syntax
 - AG description -> Attribute evaluator

Overview

- Scheme for systematic debugging for AGs
 - Query based debugging
- Generalized methods including previous methods
- An integration of previous methods
 - Slice based and Algorithmic debugging based
 - Implement on Debugger Aki

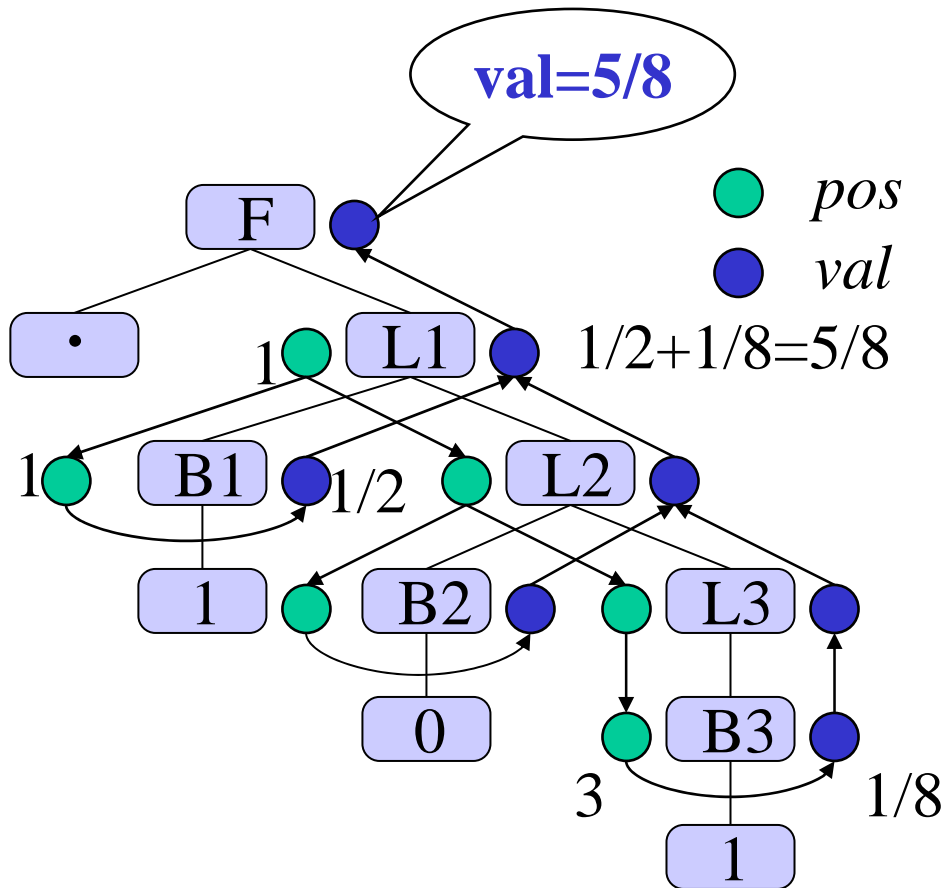


Previous work



- Debugging methods from other paradigm
 - Algorithmic Debugging [Sassa '97]
 - Slice based [Ikezoe Sasaki et al '00]
 - Cannot integrate, problem for practical use
- Declarative Debugging Scheme[Naish'97]
 - Applicable to various languages and bug classes
 - Not sufficient for AGs

Attribute Evaluation: Example (“.101”)



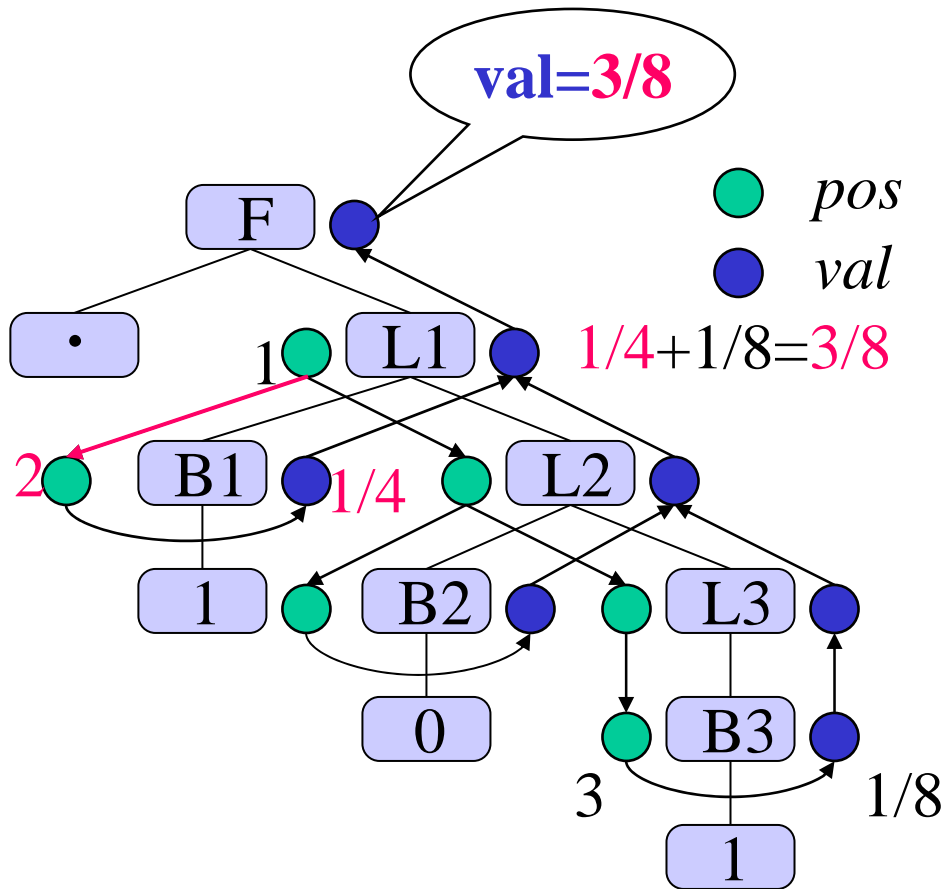
$F \Rightarrow \cdot L$
 $\{ L.pos = 1 ;$
 $F.val = L.val \}$

$L_0 \Rightarrow B L_1$
 $\{ L_1.pos = L_0.pos + 1 ;$
 $B.pos = L_0.pos ;$
 $L_0.val = B.val + L_1.val \}$

$| B$
 $\{ B.pos = L_0.pos + 1 ;$
 $L_0.val = B.val \}$

$B \Rightarrow 1$
 $\{ B.val = 2^{-B.pos} \}$
 $| 0$
 $\{ B.val = 0 \}$

Attribute Evaluation (bug included)



$F \Rightarrow \cdot L$

$\{ L.pos = 1 ;$
 $F.val = L.val \}$

$L_0 \Rightarrow B L_1$

$\{ L_1.pos = L_0.pos + 1 ;$
 $B.pos = L_0.pos + 1 ;$
 $L_0.val = B.val + L_1.val \}$

$| B$

$\{ B.pos = L_0.pos + 1 ;$
 $L_0.val = B.val \}$

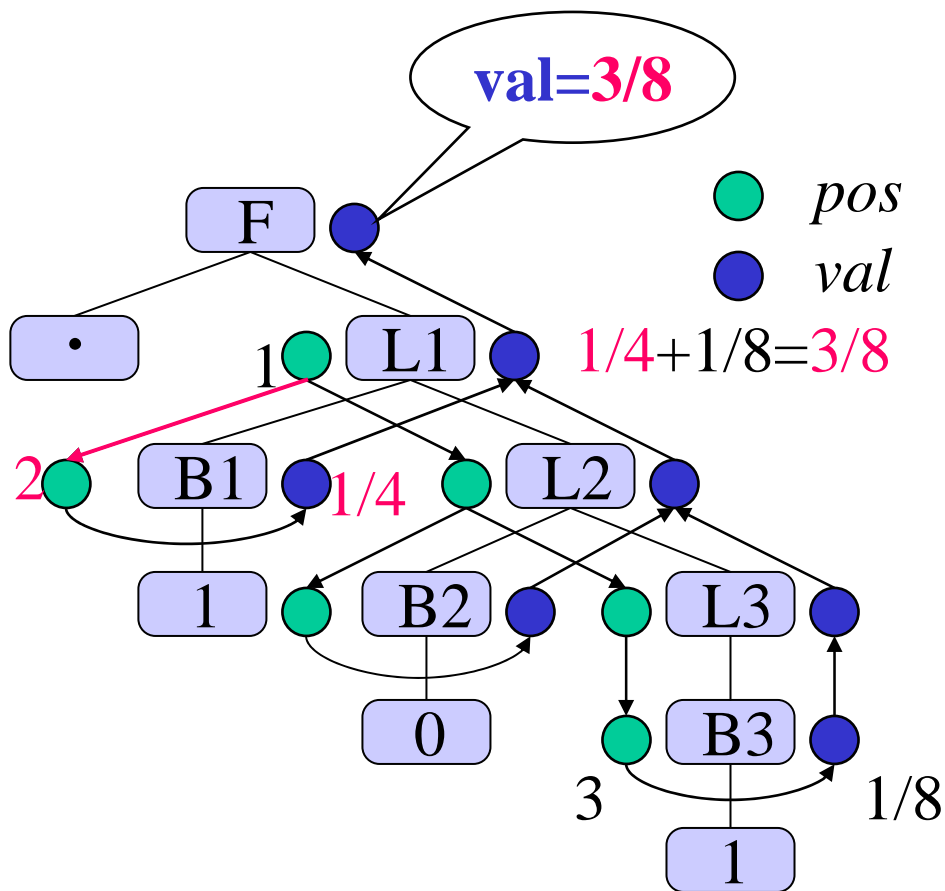
$B \Rightarrow 1$

$\{ B.val = 2^{-B.pos} \}$

$| 0$

$\{ B.val = 0 \}$

Algorithmic Debugging for AGs [Sassa et. al '97]



$L1.pos = 1; \{1\}$

$L2.pos = L1.pos + 1; \{2\}$

$L3.pos = L2.pos + 1; \{3\}$

$B3.pos = L3.pos; \{3\}$

$B3.val = 2^{-B3.pos}; \{1/8\}$

$L3.val = B3.val; \{1/8\}$

$B2.pos = L2.pos + 1; \{3\}$

$B2.val = 0; \{0\}$

$L2.val = B2.val + L3.val; \{1/8\}$

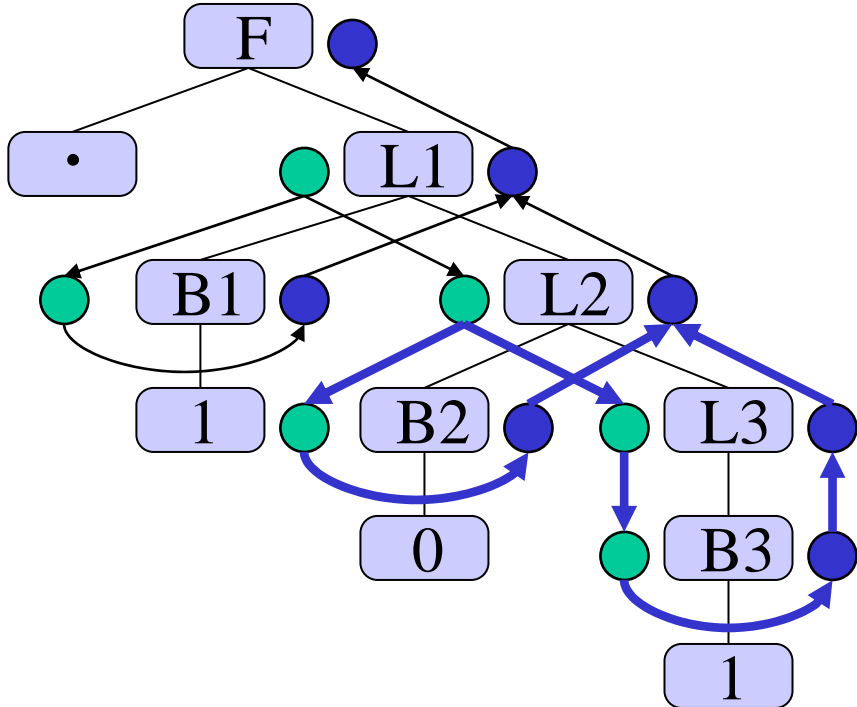
$B1.pos = L1.pos + 1; \{2\}$

$B1.val = 2^{-B1.pos}; \{1/4\}$

$L1.val = B1.val + L2.val; \{3/8\}$

$F.val = L1.val; \{3/8\}$

Algorithmic Debugging for AGs



$L1.pos = 1; \{1\}$

$L2.pos = L1.pos + 1; \{2\}$

$L3.pos = L2.pos + 1; \{3\}$

$B3.pos = L3.pos; \{3\}$

$B3.val = 2^{-B3.pos}; \{1/8\}$

$L3.val = B3.val; \{1/8\}$

$B2.pos = L2.pos + 1; \{3\}$

$B2.val = 0; \{0\}$

$L2.val = B2.val + L3.val; \{1/8\}$

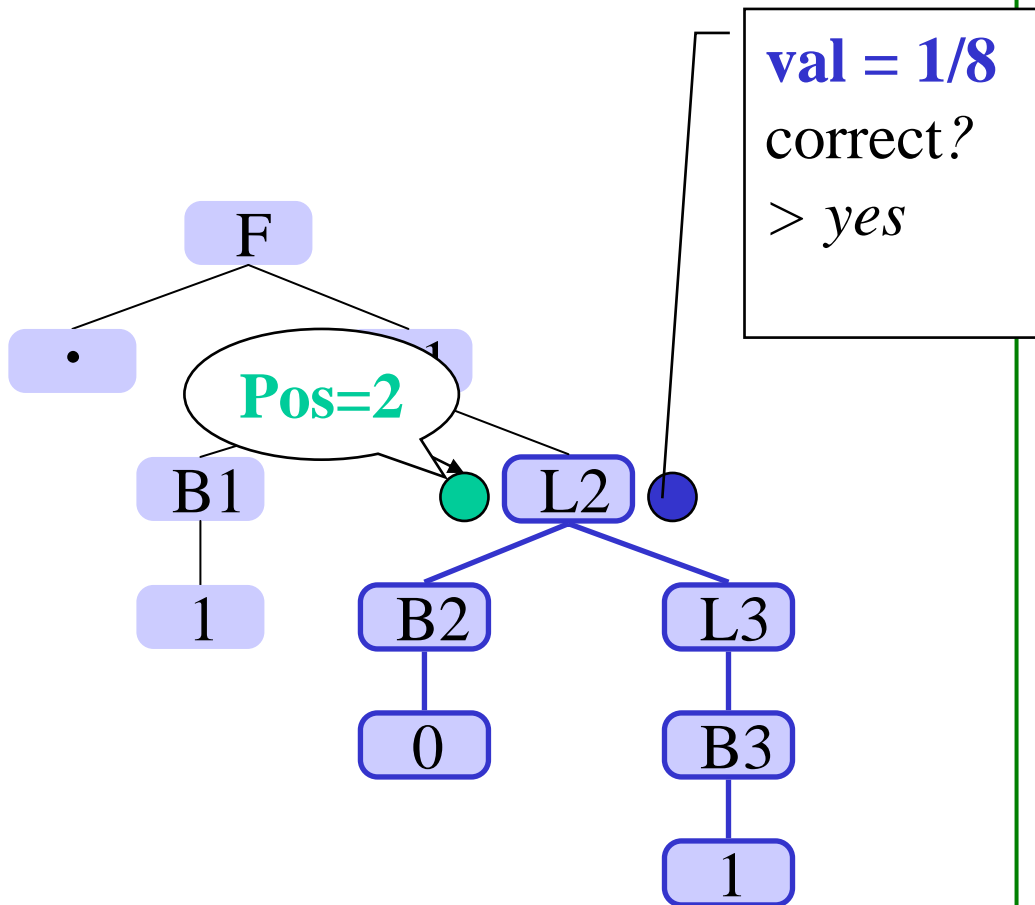
$B1.pos = L1.pos + 1; \{2\}$

$B1.val = 2^{-B1.pos}; \{1/4\}$

$L1.val = B1.val + L2.val; \{3/8\}$

$F.val = L1.val; \{3/8\}$

Query of Algorithmic Debugging



$L1.pos = 1; \{1\}$

$L2.pos = L1.pos + 1; \{2\}$

$L3.pos = L2.pos + 1; \{3\}$

$B3.pos = L3.pos; \{3\}$

$B3.val = 2^{(-B3.pos)}; \{1/8\}$

$L3.val = B3.val; \{1/8\}$

$B2.pos = L2.pos + 1; \{3\}$

$B2.val = 0; \{0\}$

$L2.val = B2.val + L3.val; \{1/8\}$

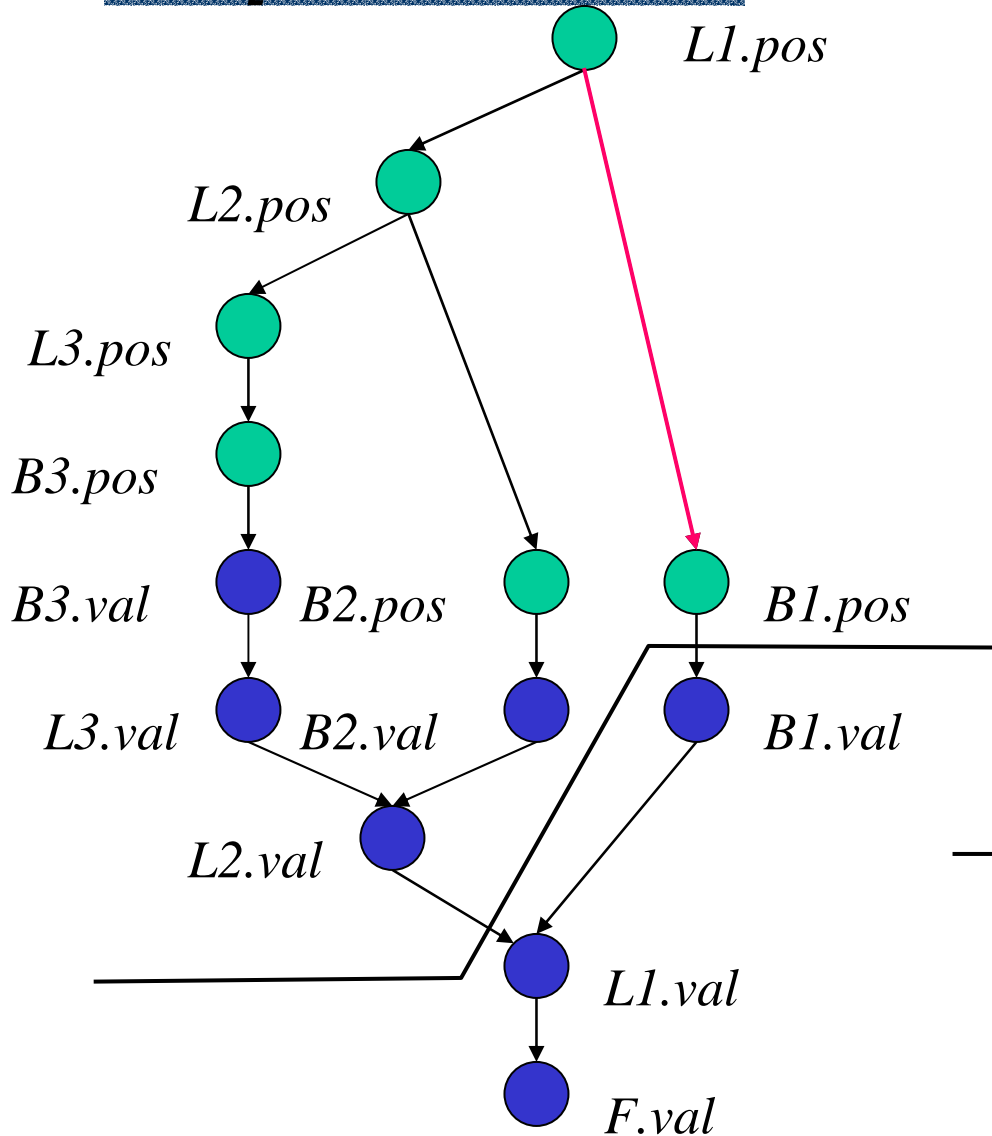
$B1.pos = L1.pos + 1; \{2\}$

$B1.val = 2^{(-B1.pos)}; \{1/4\}$

$L1.val = B1.val + L2.val; \{3/8\}$

$F.val = L1.val; \{3/8\}$

Systematic Debugging based on partition of slice [Ikezoe Sasaki et.al '00]



$$L1.pos = 1; \{1\}$$

$$L2.pos = L1.pos + 1; \{2\}$$

$$L3.pos = L2.pos + 1; \{3\}$$

$$B3.pos = L3.pos; \{3\}$$

$$B3.val = 2^{-B3.pos}; \{1/8\}$$

$$L3.val = B3.val; \{1/8\}$$

$$B2.pos = L2.pos + 1; \{3\}$$

$$B2.val = 0; \{0\}$$

$$L2.val = B2.val + L3.val; \{1/8\}$$

$$B1.pos = L1.pos + 1; \{2\}$$

$$B1.val = 2^{-B1.pos}; \{1/4\}$$

$$L1.val = B1.val + L2.val; \{3/8\}$$

$$F.val = L1.val; \{3/8\}$$

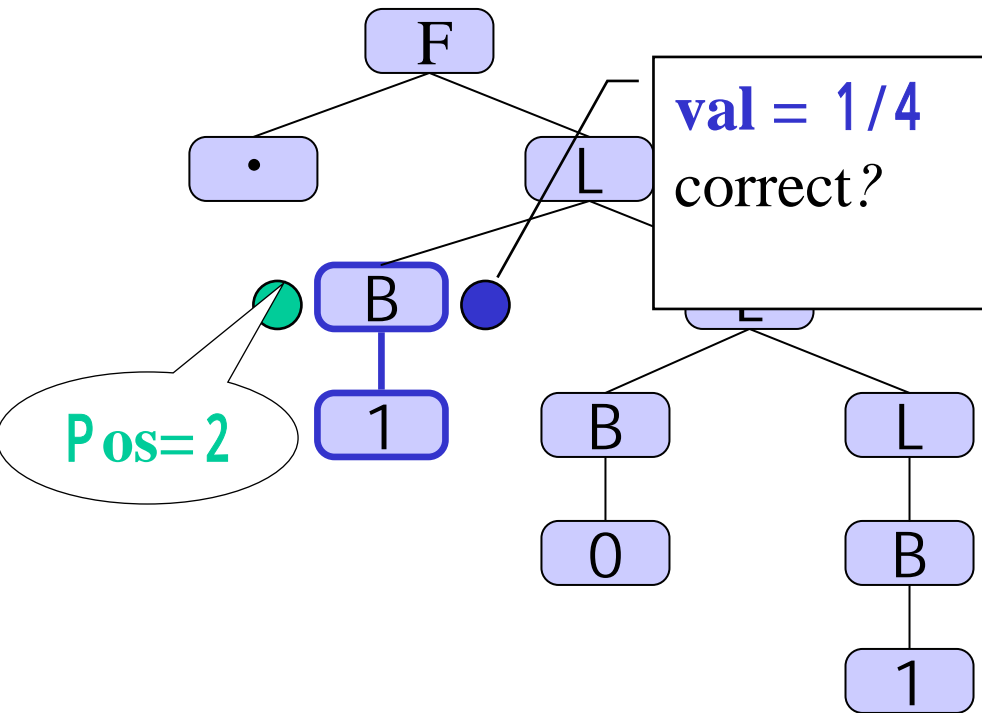
$S1$

$S2$

Problem with previous methods

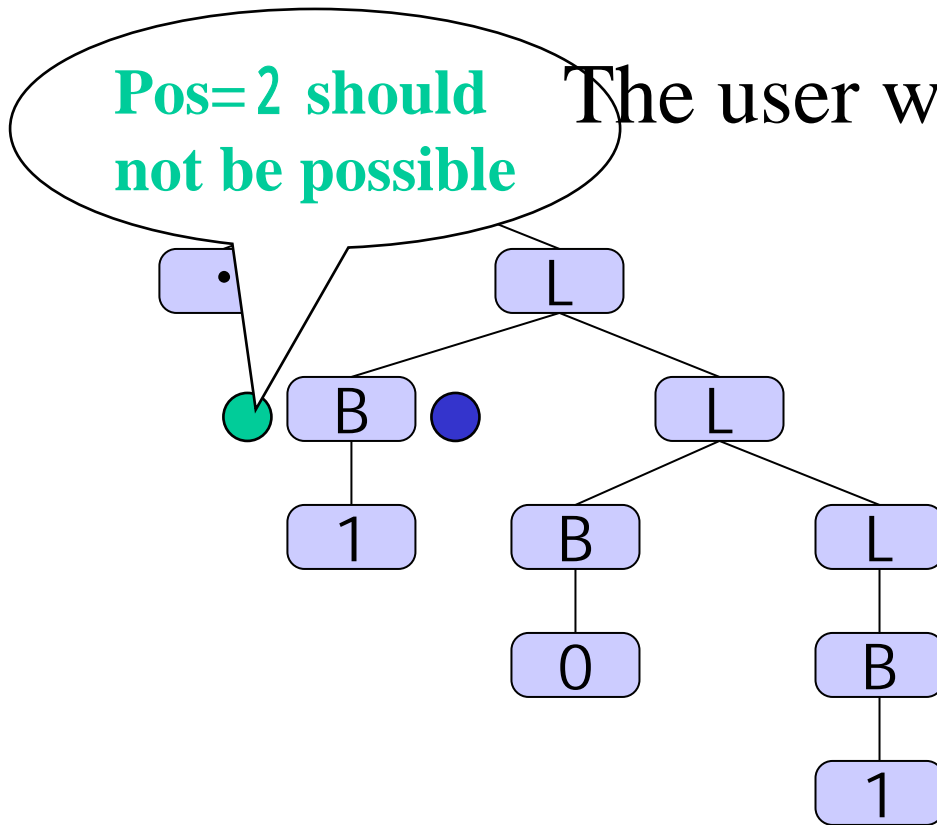
- Two methods cannot be simultaneously used
- Passive debugging style for user
- Query hard to answer
 - Query involved with large trees
 - Query for undefined value at runtime error

Problem: passive debugging style



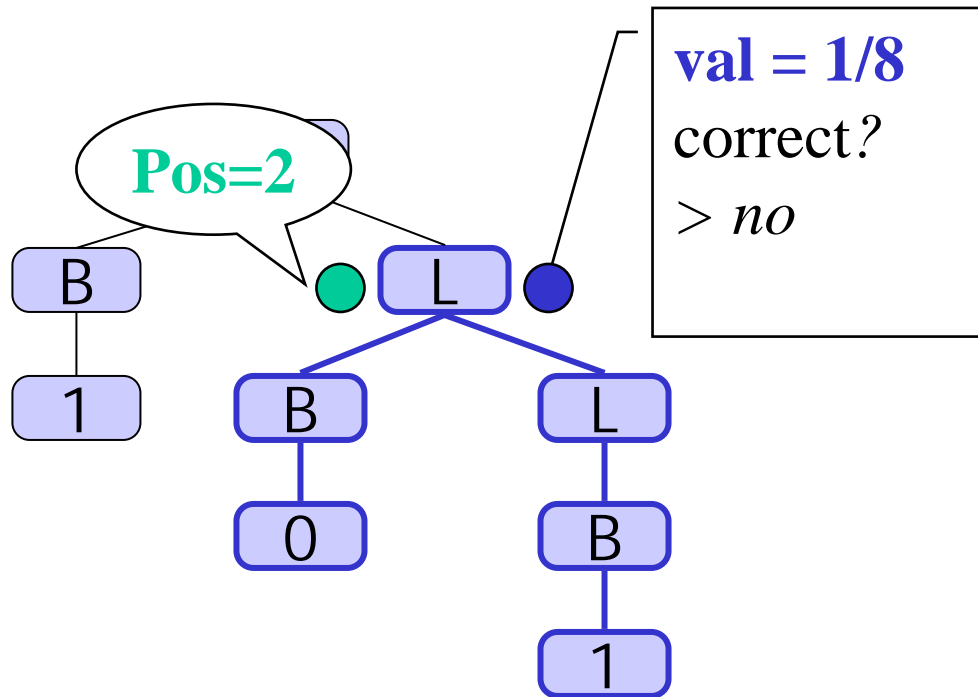
The proposition $\text{pos}=2$ was wrong

Problem: passive debugging style

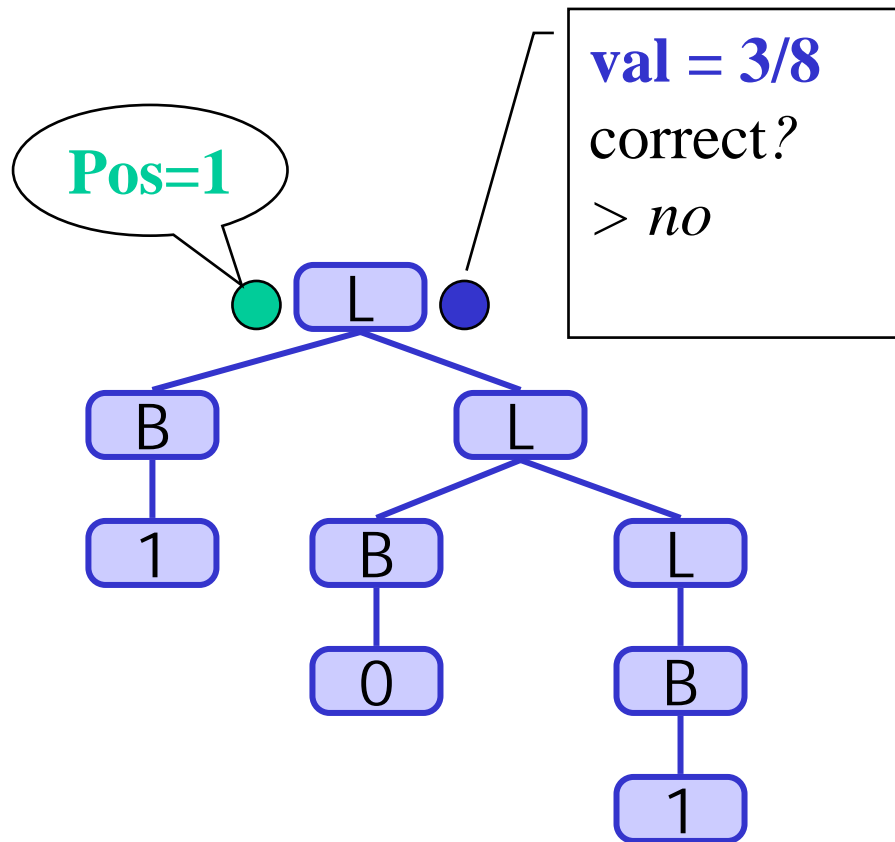


The user cannot give any debugging information to the debugger except through queries

Problem: Query for large trees

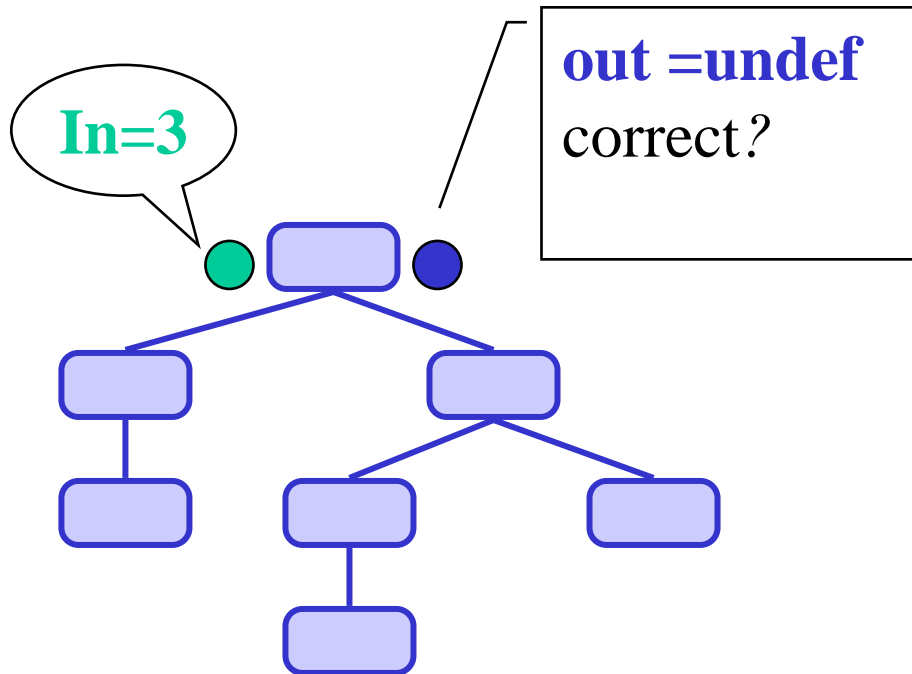


Problem: Query for large trees



Queries about attributes near the root node force the user to look over a large tree

Problem: Undefined value on runtime error

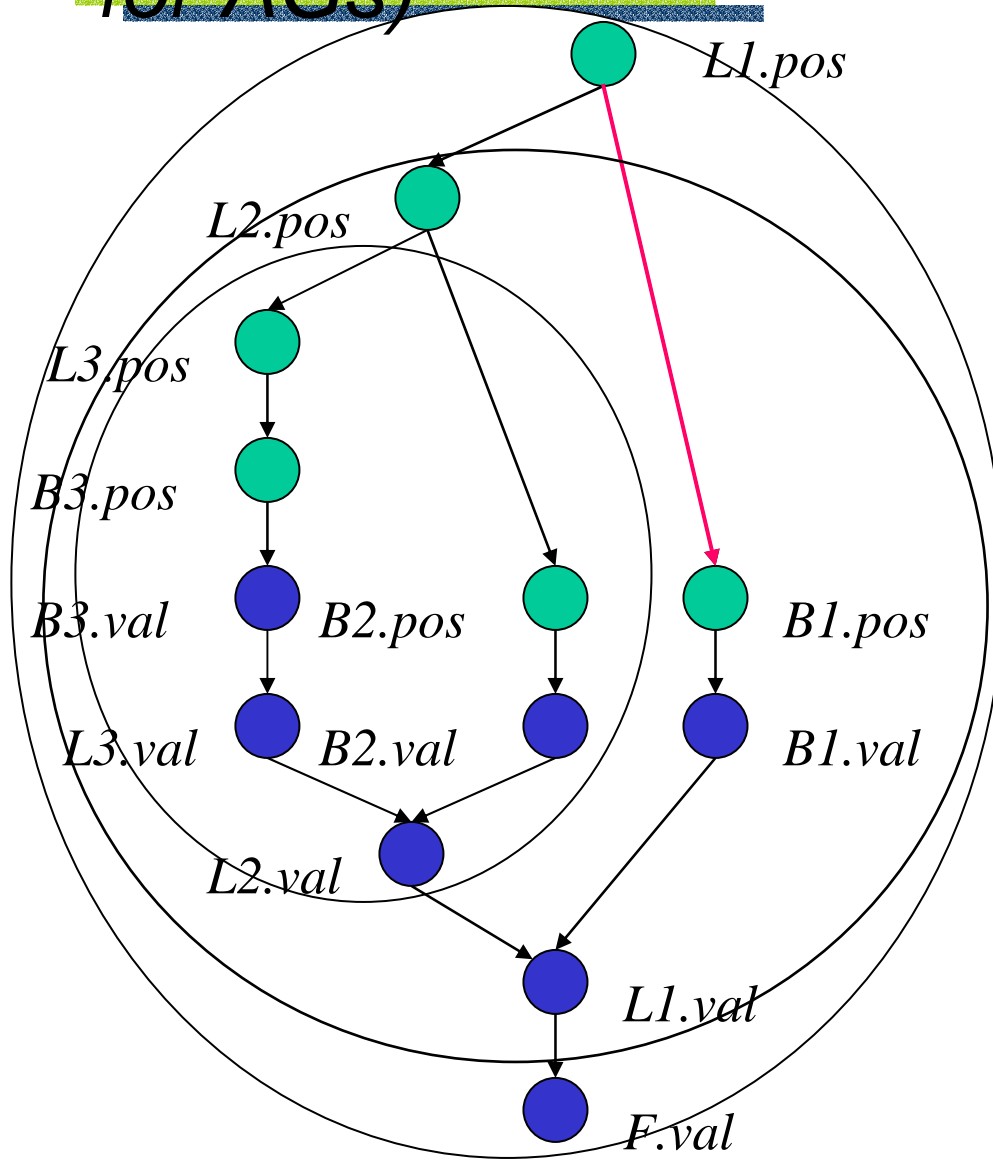


Approach



- Generalized algorithmic debugging methods
 - Generalized scheme including previous two methods
- Integration of previous two methods

Generalization: (Algorithmic Debugging for AGs)



$L1.pos = 1; \{1\}$

$L2.pos = L1.pos + 1; \{2\}$

$L3.pos = L2.pos + 1; \{3\}$

$B3.pos = L3.pos; \{3\}$

$B3.val = 2^{-B3.pos}; \{1/8\}$

$L3.val = B3.val; \{1/8\}$

$B2.pos = L2.pos + 1; \{3\}$

$B2.val = 0; \{0\}$

$L2.val = B2.val + L3.val; \{1/8\}$

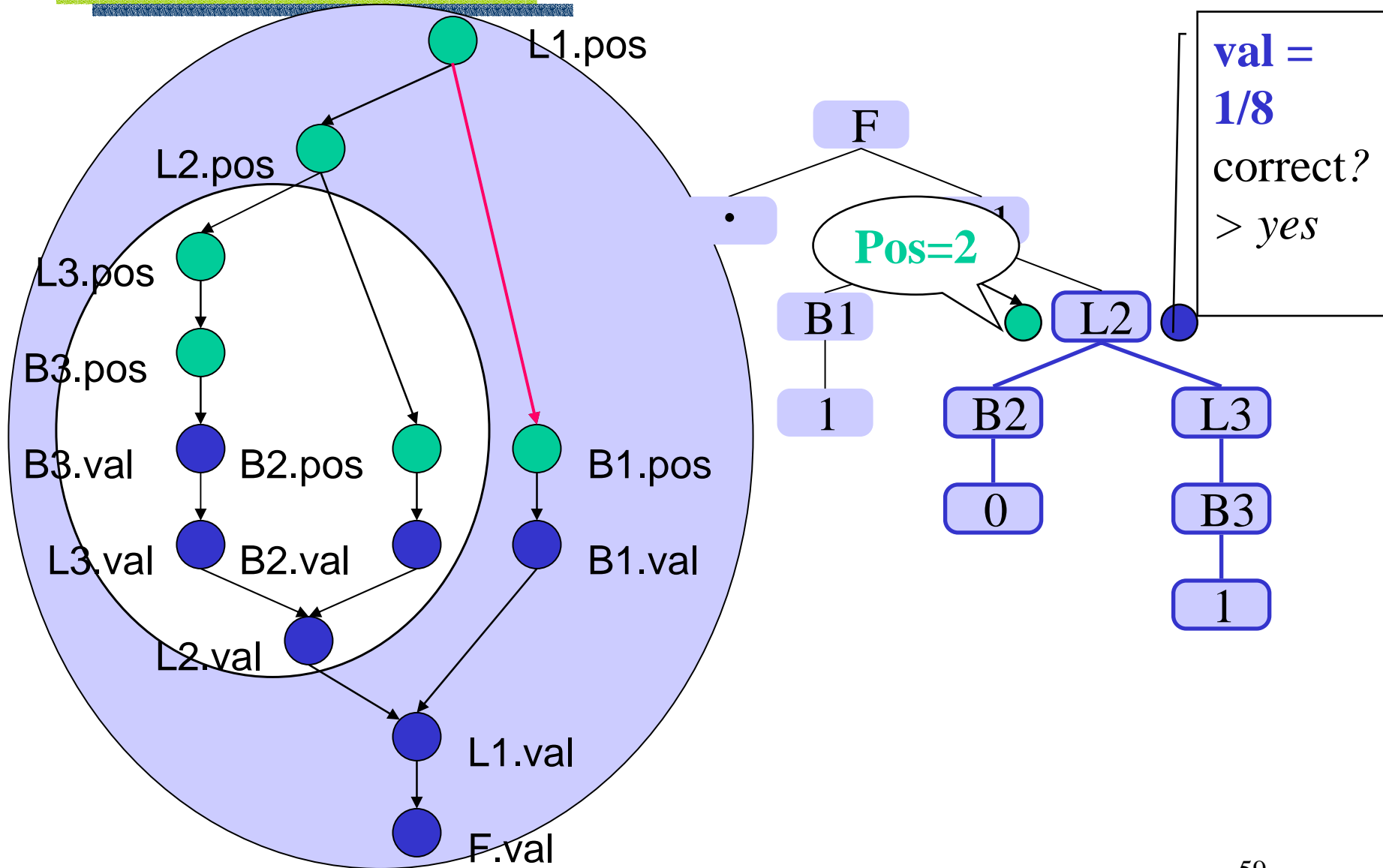
$B1.pos = L1.pos + 1; \{2\}$

$B1.val = 2^{-B1.pos}; \{1/4\}$

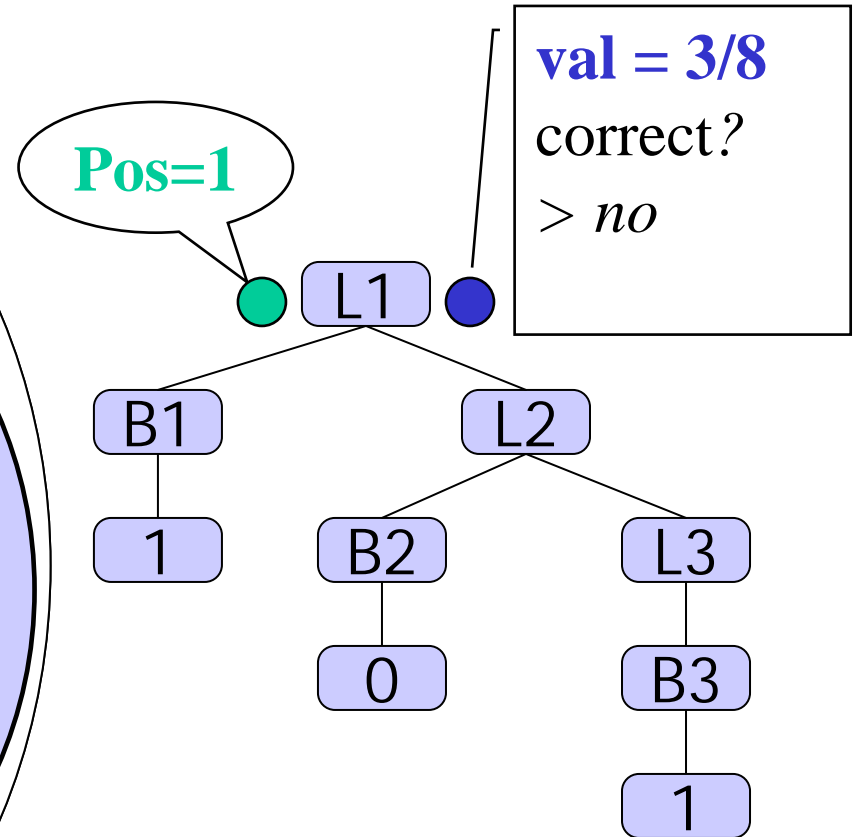
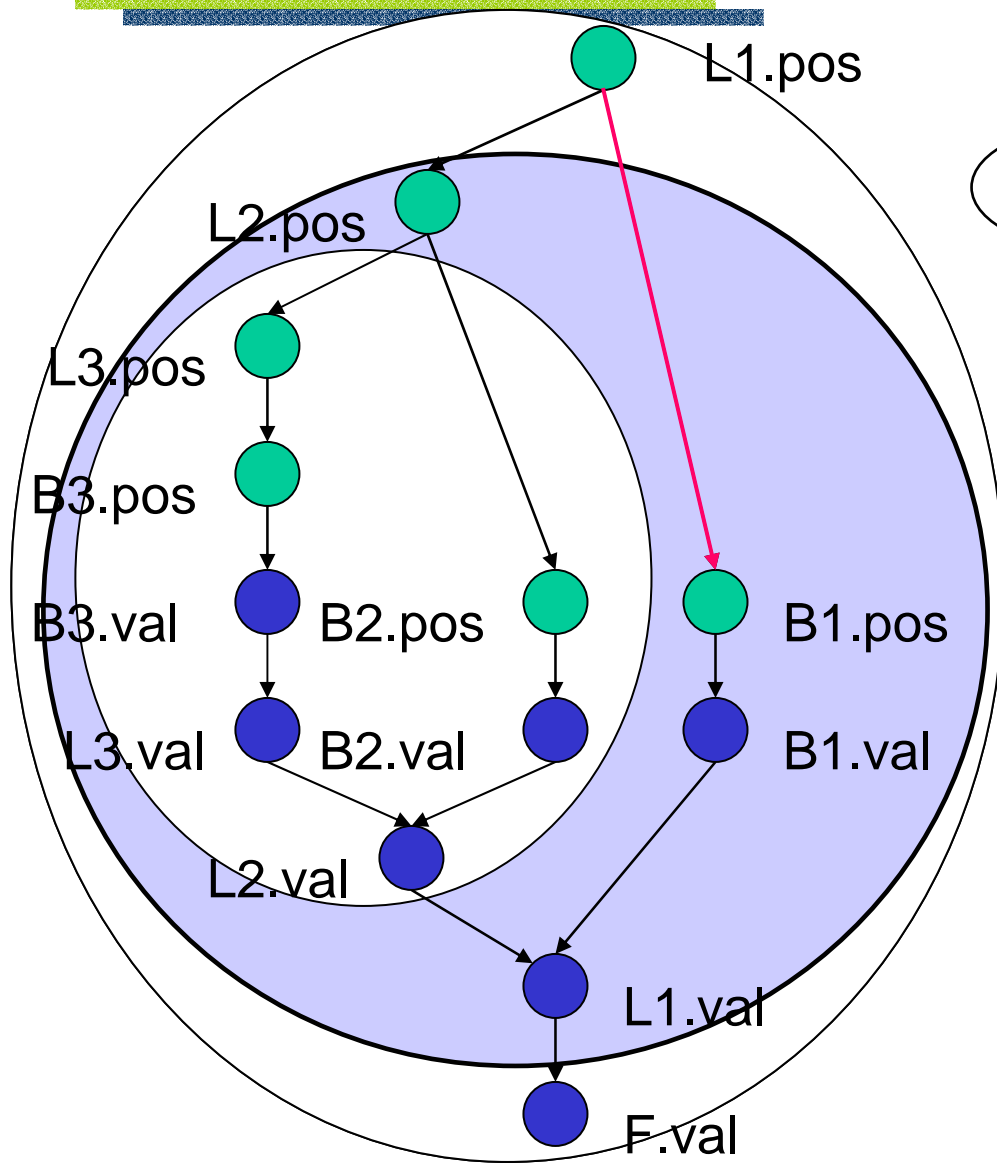
$L1.val = B1.val + L2.val; \{3/8\}$

$F.val = L1.val; \{3/8\}$

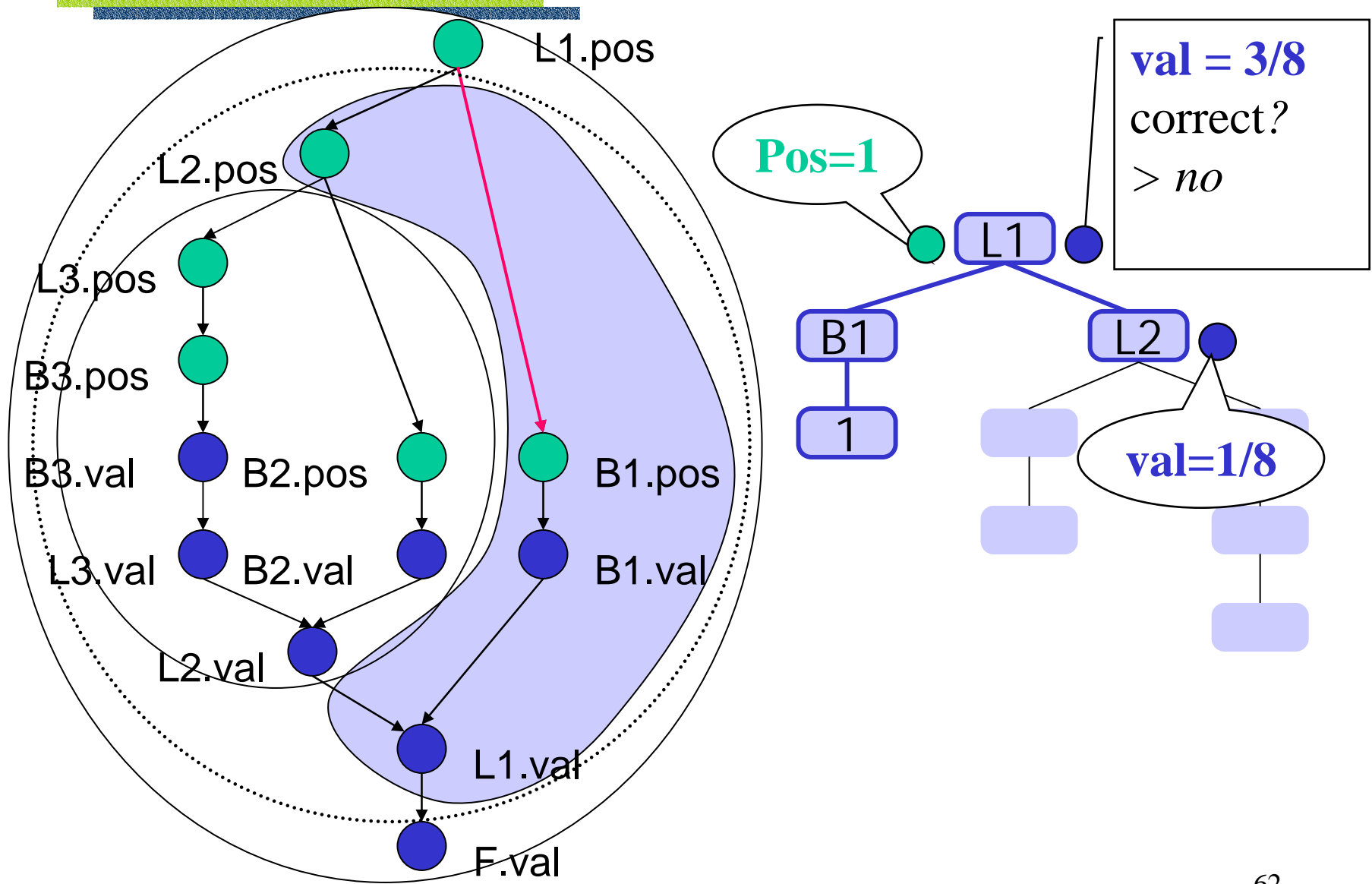
Generalization: (Algorithmic Debugging)



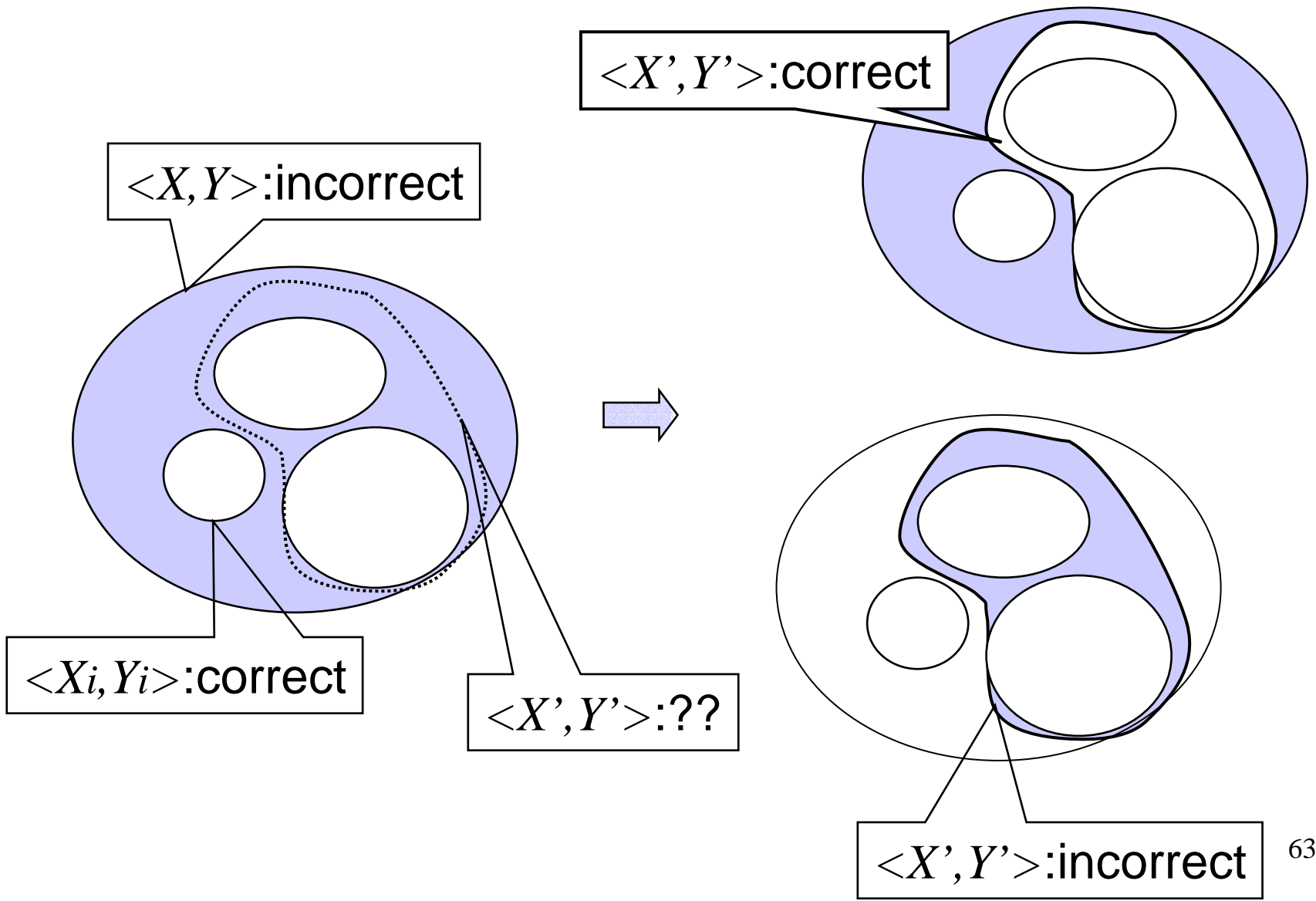
Generalization: (Algorithmic Debugging)



Generalized Algorithmic Debugging for AGs



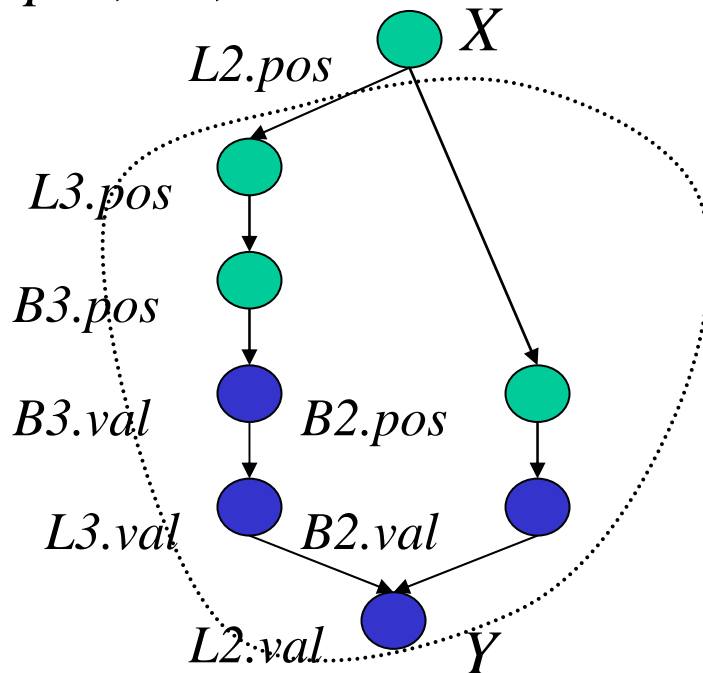
Debugging Algorithm



Attribute Computation Composition

- A function obtained by composing attribute computations
- For the value of attributes X (input), determined the value of Y (output) $\langle X, Y \rangle$

Example: $\langle L2.pos, L2.val \rangle$



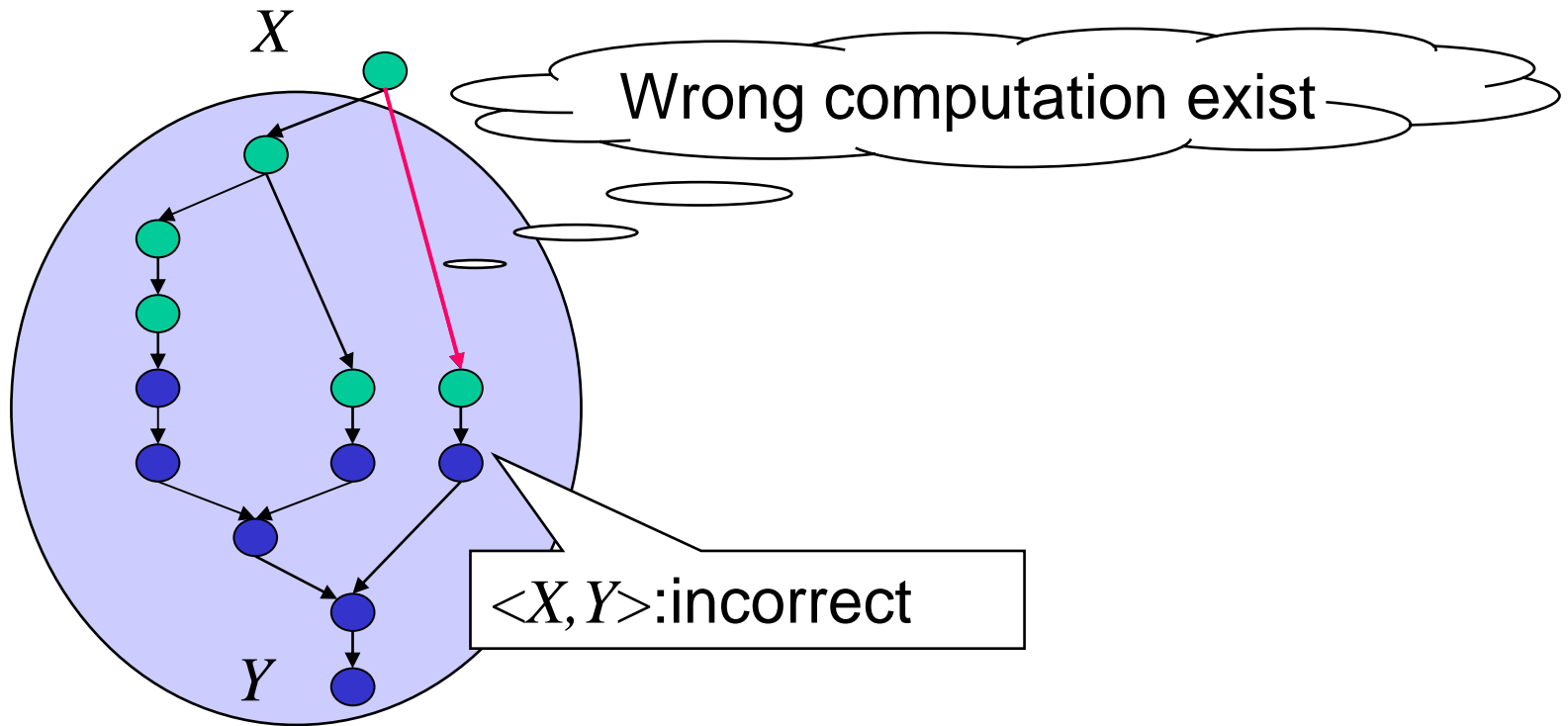
Query

- Question to the user whether the behavior of attribute computation composition does or does not satisfy the specification
- Correctness of the output attribute values for the input attribute values
- *Behavior for $\langle X, Y \rangle$ is correct/incorrect*
 - $\text{Query}(\langle X, Y \rangle) = \text{correct}$
 - $\text{Query}(\langle X, Y \rangle) = \text{incorrect}$

Theorem 1

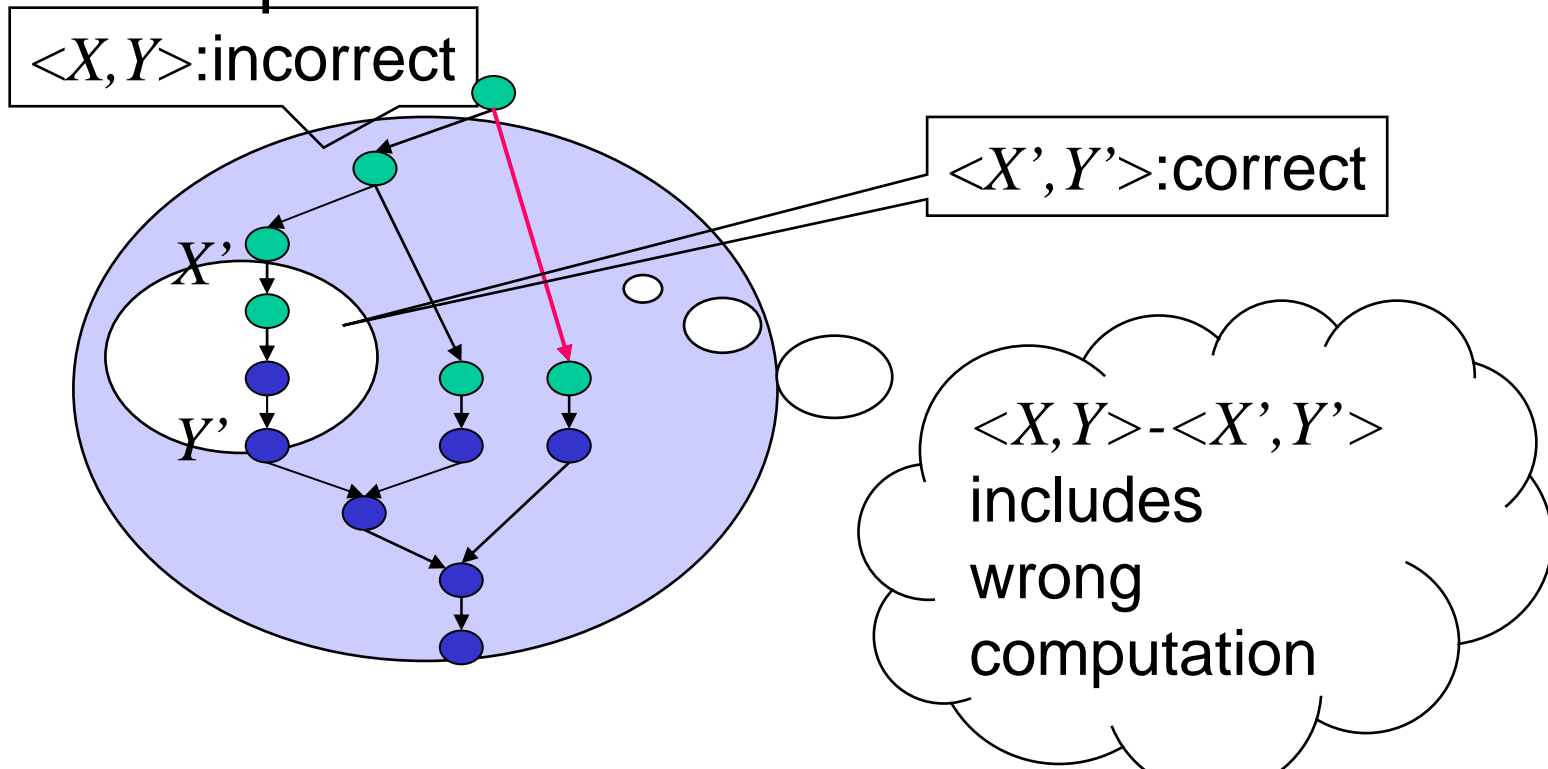
- $\text{Query}(\langle X, Y \rangle) = \text{incorrect}$

$\langle X, Y \rangle$ includes a wrong computation



Theorem 2

- Provided $\text{Query}(\langle X, Y \rangle) = \text{incorrect}$.
For closed $\langle X', Y' \rangle$ included by $\langle X, Y \rangle$,
 $\text{Query}(\langle X', Y' \rangle) = \text{correct}$
 $\langle X, Y \rangle - \langle X', Y' \rangle$ includes at least one wrong
computation



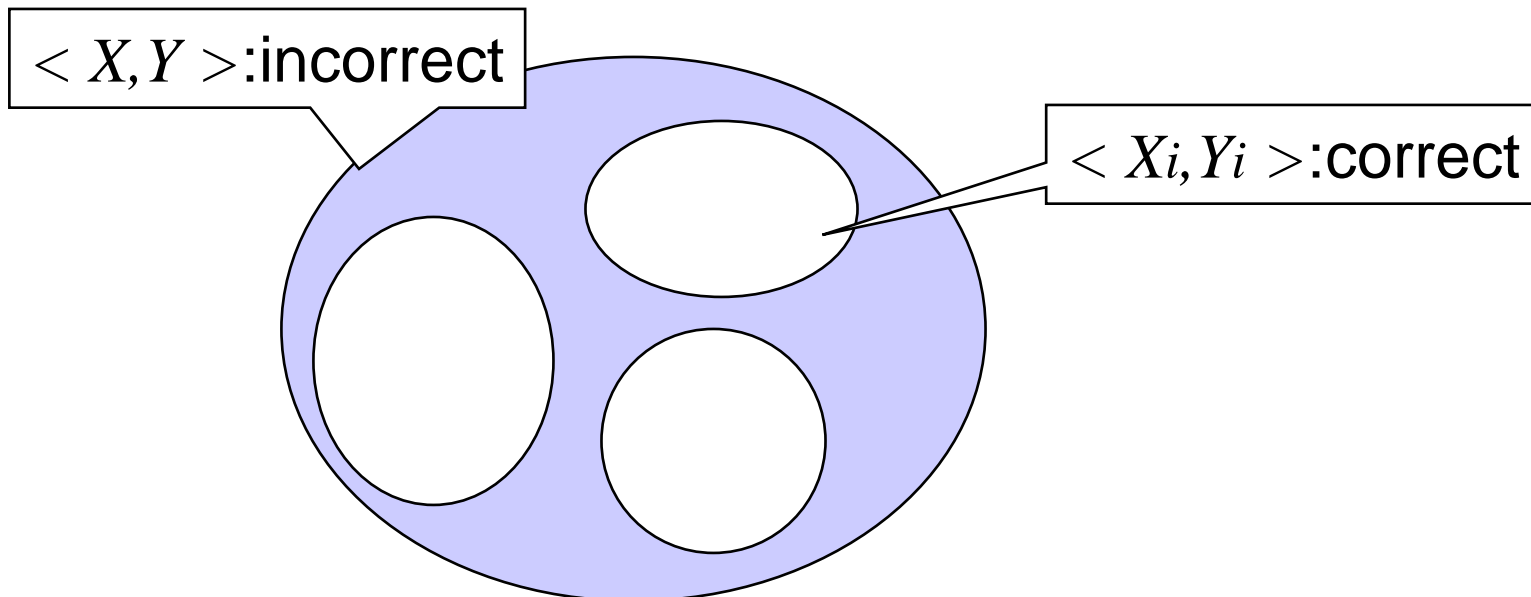
Colloraly 1

- Provided $\text{Query}(\langle X, Y \rangle) = \text{incorrect}$.

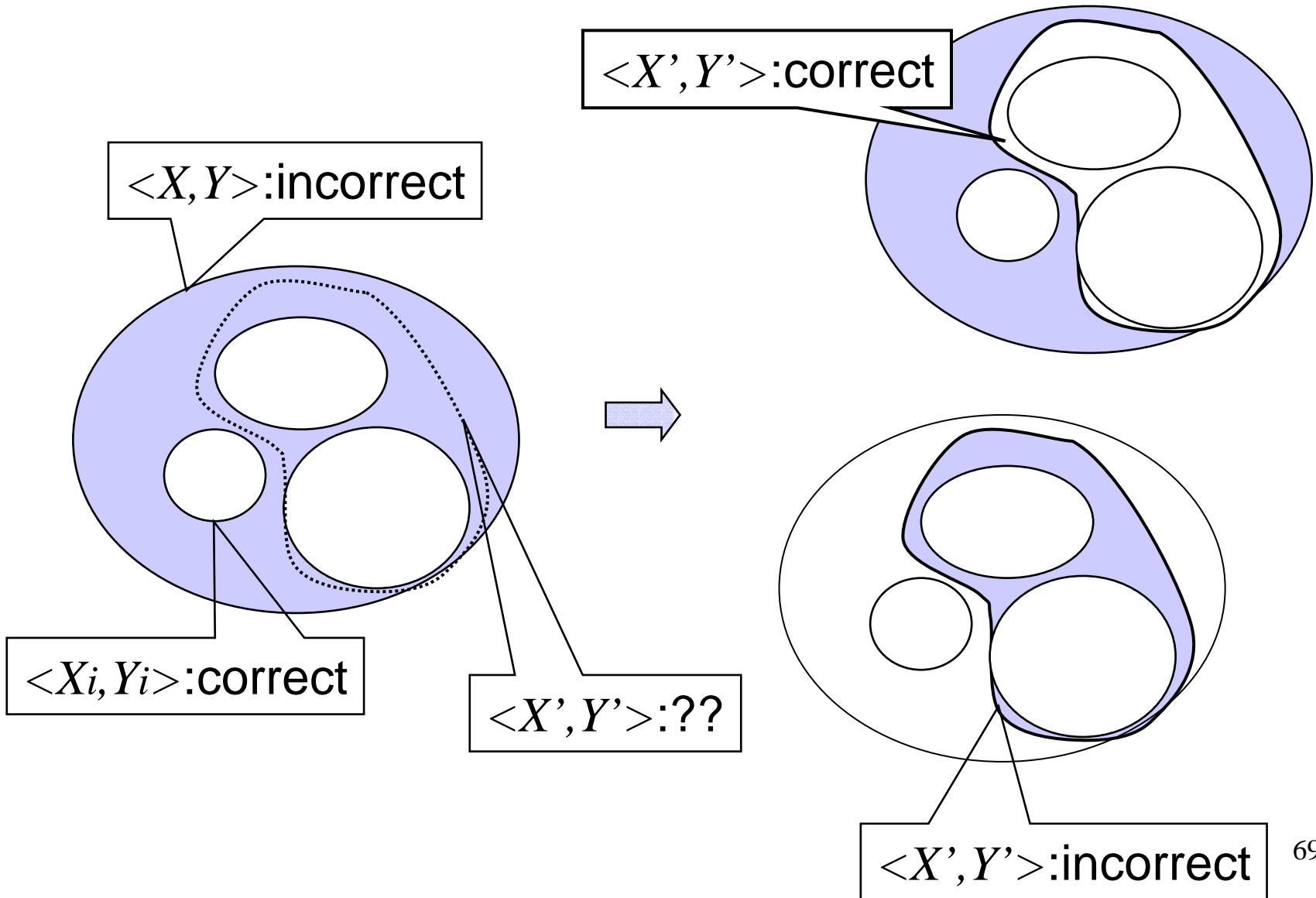
For $\langle X_i, Y_i \rangle \subset \langle X, Y \rangle$, $\langle X, Y \rangle - U_i \langle X_i, Y_i \rangle$

includes at least one wrong computation under:

- $\text{Query}(\langle X_i, Y_i \rangle) = \text{correct}$ and $\langle X_i, Y_i \rangle$ is closed
- $\langle X_j, Y_j \rangle$ and $\langle X_k, Y_k \rangle$ are exclusive



Collorary 2 (Debugging Algorithm)

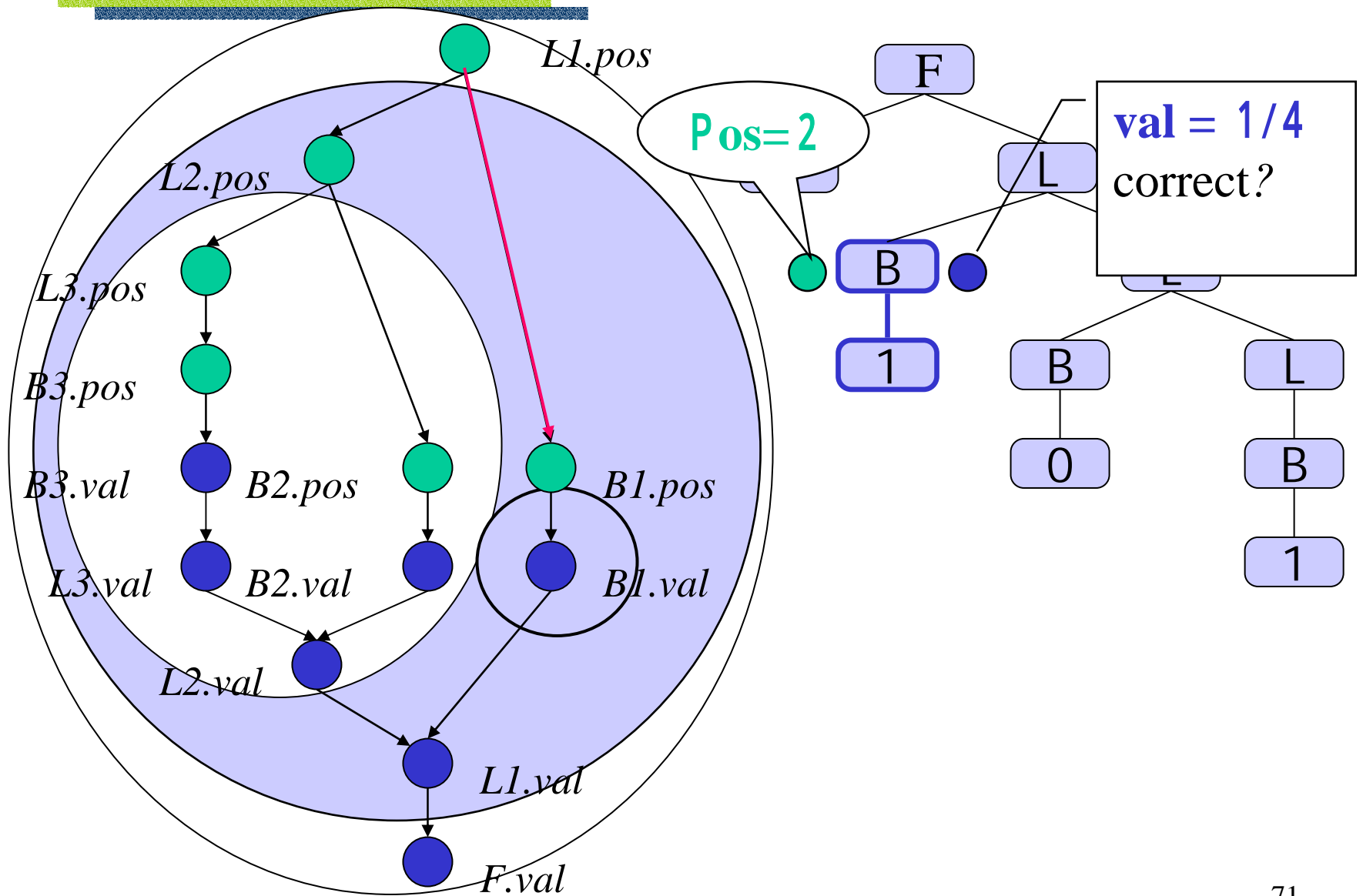


Integration of previous debugging methods

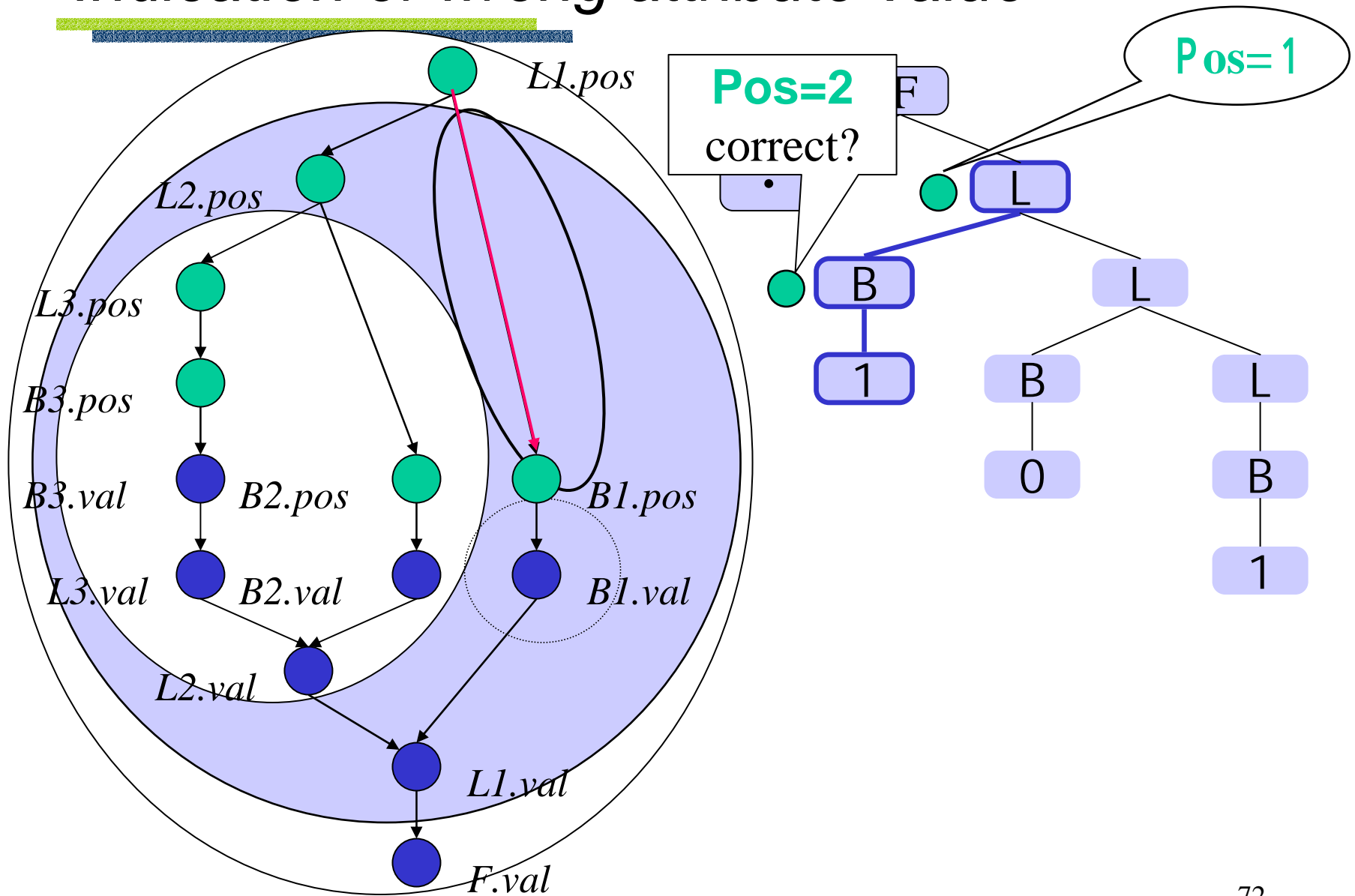


- Using generalized scheme
 - Indication of wrong attribute
 - Question hard to answer
 - Query with large tree
 - Undefined value at runtime error

Indication of wrong attribute value



Indication of wrong attribute value



Experimental Result

- Comparison of the number of queries
 - Type checking (or static analysis) of the Tiger Language [Appel98]
 - 25 productions, 105 semantic rules

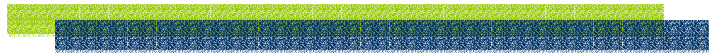
	# attrs	# nds	Slice	AD	GAD
<i>A</i>	78	52	7(1)	8(3)	4(1)[6(1)]
<i>B</i>	146	103	6(1)	9(4)	5(1)
<i>C</i>	60	43	4(1)	8(3)	4(1)
<i>D</i>	56	34	9(1)	6(4)	7(1)
<i>E</i>	45	32	5(1)	5(2)	7(2)
<i>F</i>	104	69	7(1)	6(2)	2(2)

().. Num of semantic rules identified

Conclusion

- Generalized algorithmic debugging for AGs
 - Scheme for systematic debugging for AGs
- Integration of the previous methods

Conclusion



Conclusion

- **Circular and Remote AG**
 - AG extension that allows both of:
 - Circularity
 - Remote Attribute References
 - Efficient attribute evaluators for CRAGs
 - Static evaluator and Mostly static evaluator
- **Systematic Debugging for AGs**
 - Generalized systematic debugging
 - Integration of previous methods