Specification-Based Sketching with Sketch#

Hesam Samimi

UCLA

Kaushik Rajan

MSR India

This Talk

- Motivation
- Sketch#
- Evaluation

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

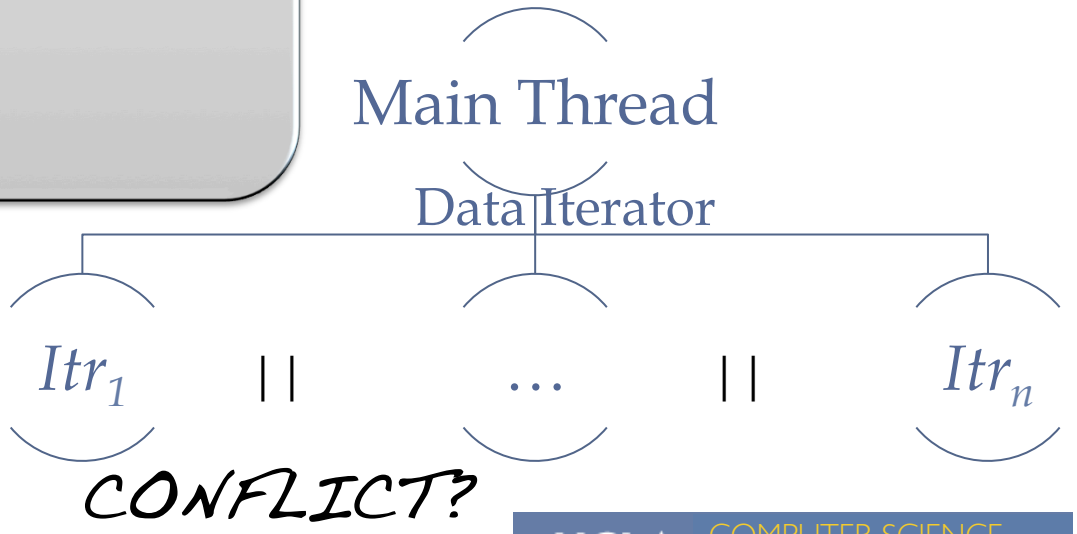
Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

PRESERVE SEQUENTIAL SEMANTICS!



Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(5);
```

```
Add(4);
Remove(4);
```

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(5);
```

```
Add(4);
Remove(4);
```

✓

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(5);
```

```
Add(4);
Remove(4);
```

✓

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(5);
```

```
Add(4);
Remove(4);
```

✓ NO CONFLICTS

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(4);
```

```
Add(4);
Remove(4);
```

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Contains(4);
```

```
Add(4);
Remove(4);
```

✗

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND

Itr_1 || Itr_2

```
Add(4);
```

```
Contains(4);
```

```
Remove(4);
```

✗ CONFLICT

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND
- INVERSE

Itr$_1$ || Itr$_2$

```
Contains(4);
```

```
Add(4);
```

```
Remove(4);
```

ROLLBACK

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND
- INVERSE

Itr$_1$ || Itr$_2$

```
Contains(4);
```

```
Add(4);
```

```
Remove(4); Add(4);
```

ROLLBACK

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND
- INVERSE

Itr_1 || Itr_2

```
Contains(4);
```

```
Add(4);
```

ROLLBACK

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND
- INVERSE

Itr$_1$ || Itr$_2$

```
Contains(4);
```

```
Add(4); Remove(4);
```

ROLLBACK

Optimistic Parallel Runtimes

Optimistic Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'11]

- COMMUTATIVITY COND
- INVERSE

Itr_1 || Itr_2

```
Contains(4);
```

RETRY

Optimistic Parallel Runtimes

- COMMUTATIVY COND
- INVERSE

```
class ArrayList {
    int IndexOf(int v1)
        Commutes: Insert(i2, v2) when
                    (result < 0 && v1 != v2) ||
                    (0 <= result && result < i2) ||
                    (result == i2 && v1 == v2);
}
```

NOT TRIVIAL!

Optimistic Parallel Runtimes

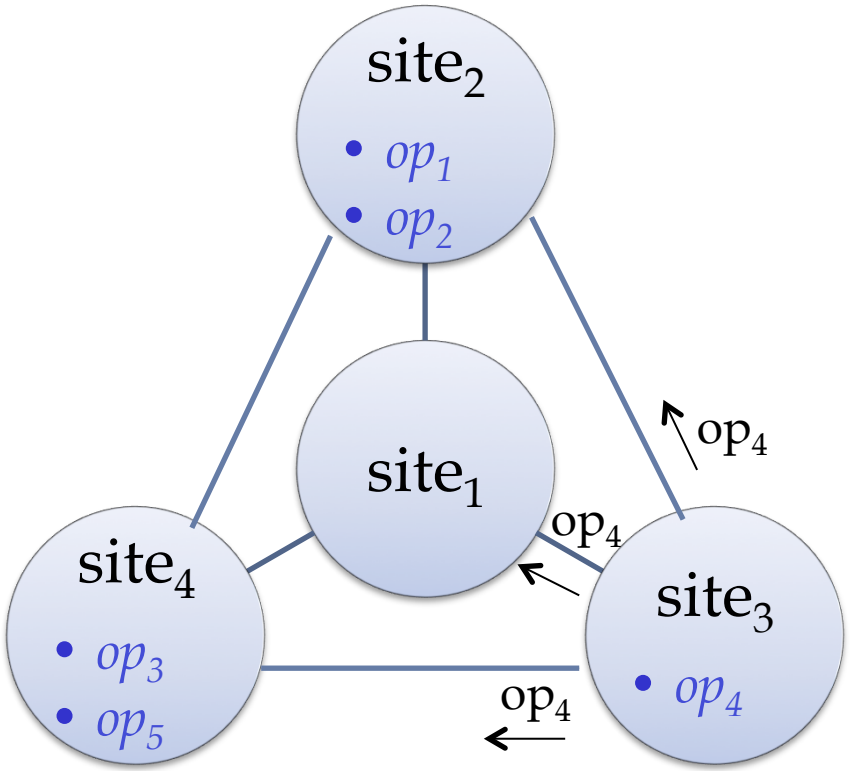
Distributed Systems

(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

Optimistic Parallel Runtimes



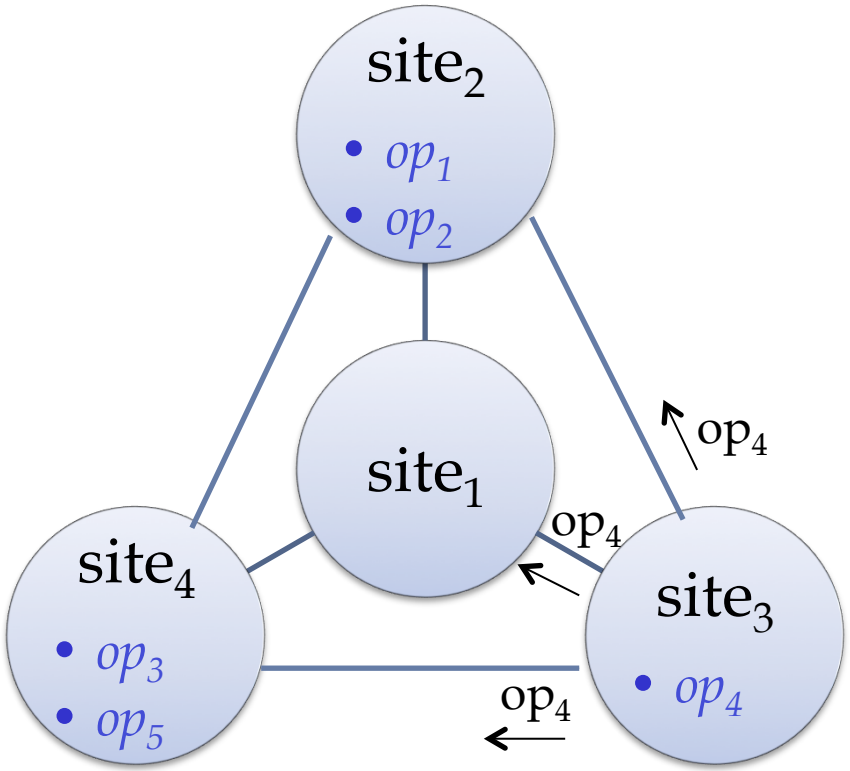
Distributed Systems

(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

Optimistic Parallel Runtimes



EVENTUAL CONSISTENCY!

Distributed Systems

(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

Optimistic Parallel Runtimes

Insert(2,'a'); Insert(5,'o')
≠
Insert(5,'o'); Insert(2,'a')

ns

ive

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes

Insert(2,'a'); *T(Insert(5,'o'), Insert(2,'a'))*
? =
Insert(5,'o'); *T(Insert(2,'a'), Insert(5,'o'))*

ns

ive

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes

Insert(2,'a'); Insert(**6**,'o')
=
Insert(5,'o'); Insert(**2**,'a')

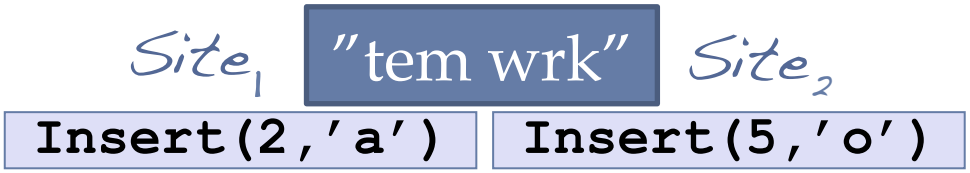
ns

ive

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

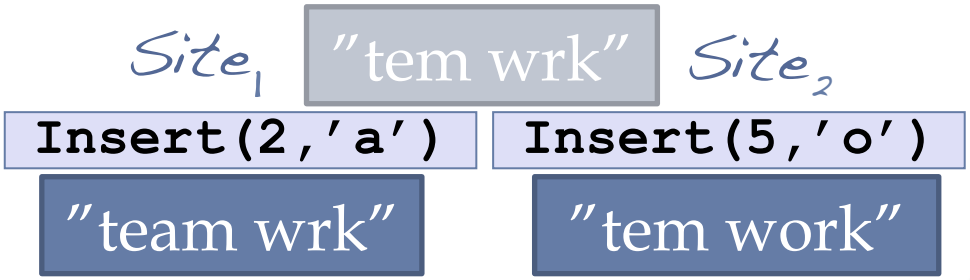
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

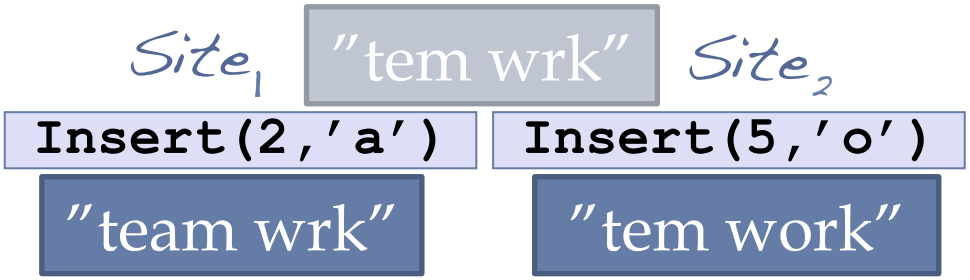
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

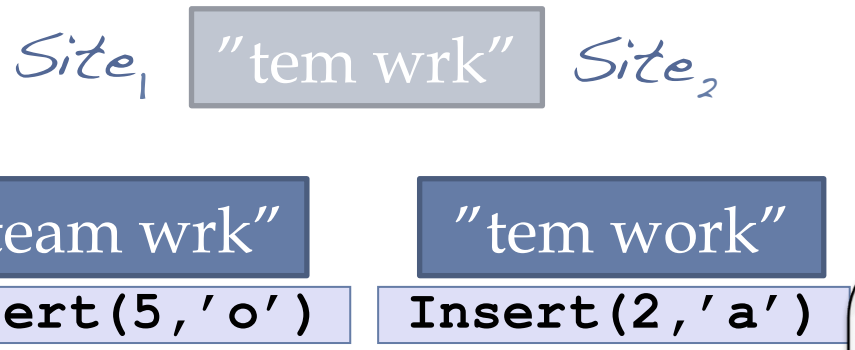
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

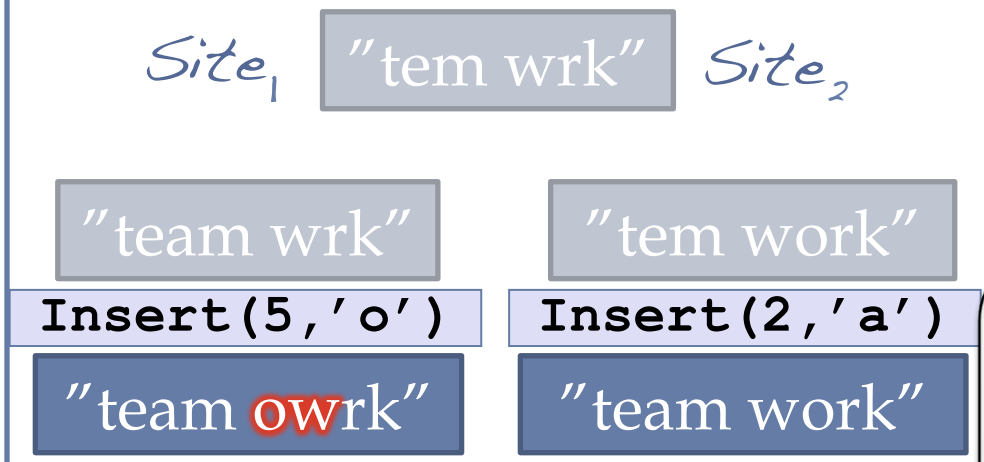
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

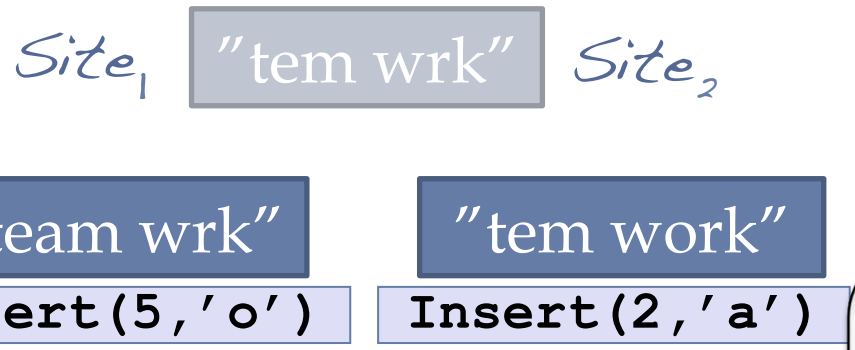
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

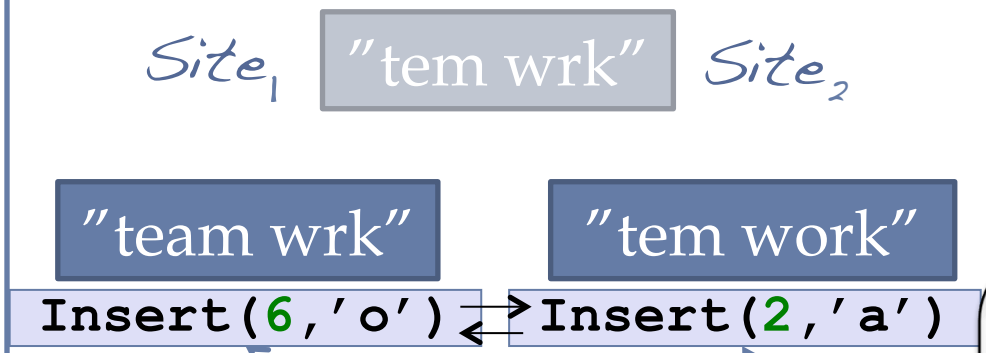
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

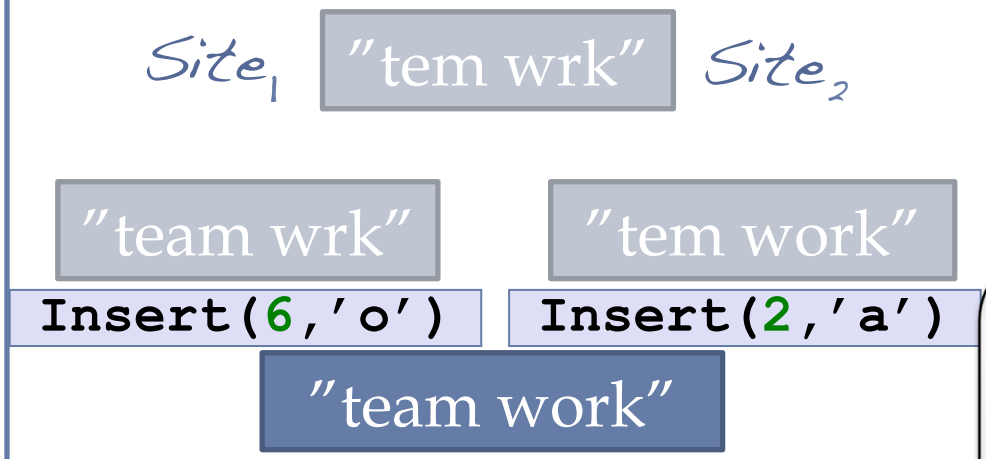
(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes



Distributed Systems

(Real-Time Collaborative Editing)

Google Wave (R.I.P.) ['09]

CoWord ['04]

OPERATIONAL TRANSFORMS (OT)

Optimistic Parallel Runtimes

```
class ArrayList {
    void Insert(int pos, int val)
        OT: Insert(int pos2, int val2) {
                if (pos > pos2 ||
                    (pos == pos2 && val > val2))
                    Insert(pos + 1, val);
                else
                    Insert(pos, val);
            }
}
```

NOT TRIVIAL!

OPERATIONAL TRANSFORMS (OT)

Problem…

Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'1

- COMMUT
- INVERSE

MANUALLY FINDING PRECISE, CORRECT ONES IS DIFFICULT, ERROR-PRONE

ed Systems

Collaborative
ting)

.P.) ['09]

Covvoru ['04]

- OPERATIONAL TRANSFORMS (OT)

Goal…

Parallel Systems

Galois [PLDI'07]

CommSets [PLDI'1

- COMMUT
- INVERSE

AUTOMATICALLY FIND PRECISE, PROVEN-CORRECT ONES

ed Systems

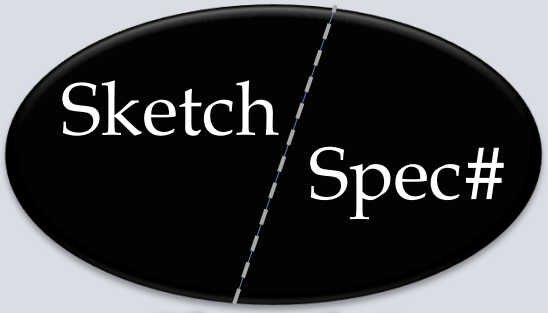
Collaborative
ting)

Google Wave (R.I.P.) ['09]

CoWord ['04]

- OPERATIONAL TRANSFORMS (OT)

Sketch#



Sketch#

- COMMUTATIVY COND
- INVERSE
- OPERATIONAL TRANSFORMS (OT)

- Synthesized these for:
  - Stack: `Push`, `Pop`
  - ArrayList: `Insert`, `Delete`
  - XMLNode: `InsertAt`
- General-purpose tool with specifications
  - proof-of-concept

This Talk

- Motivation
- Sketch#
- Evaluation

Spec#


```
class Stack {
    int size, int[] elements;
    boolean Equals(Stack s) ensures …; { … }
    void Push(int val) requires …; ensures … ; { … }
    int Pop() requires …; ensures … ; { … }









}
```

Spec#

Barnett, Leino

CASSIS '04

Spec#


```
class Stack {
    int size, int[] elements;
    boolean Equals(Stack s) ensures …; { … }
    void Push(int val) requires …; ensures … ; { … }
    int Pop() requires size > 0;
      ensures result == old(elements[0]);
      ensures forall {int i in (1 : old(size));
            elements[i-1] == old(elements[i])};
      ensures size == old(size) - 1;
    { … }

}
```

Spec#


```
class Stack {
    int size, int[] elements;
  ✓ boolean Equals(Stack s) ensures …; { … }
  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

  ? void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        s1.Pop();
    }
}
```

SEMANTICS OF OP INVERSE

Spec#


```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

  ✓ void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        s1.Pop();
    }
}
```

ONLY SPECS MATTER

MODULAR REASONING

Sketch# = Spec# + Sketch

Sketching

Solar-Lezama, Bodik
PLDI'05-'07

```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

    void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        ?;
    }
}
```

Sketch# = Spec# + Sketch


```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

    void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        if (?) s1.Push(?) else s1.Pop();
    }
}
```

MODULAR SKETCHING

Sketch# = Spec# + Sketch


```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

    void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        if (?) s1.Push(?) else s1.Pop();
    }
}
```

Start

Sketch# = Spec# + Sketch


```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

    void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        if (false) s1.Push(0) else s1.Pop();
    }
}
```

Sketch# = Spec# + Sketch


```
class Stack {
    int size, int[] elements;

  ✓ void Push(int val) requires …; ensures … ; { … }
  ✓ int Pop() requires …; ensures … ; { … }

    void Push_Inv_Test(Stack s1, Stack s2, int val)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Push(val);
        s1.Pop();
    }
}
```

This Talk

- Motivation
- Sketch#
  - Syntax
- Evaluation

Sketch# Syntax

Symbol	Type	Translation
`?`	int/bool	`?`
`?(`v_1`,…)`	int	`? * `v_1` + …`
`?(`v_1`,…)`	bool	`? * `v_1` + … >= 0`

`?{}`	stmt	`if (?(…))`
		`  if (?) `$\mathbf{Op_1}$`(?(…), …)`
		`  else if (?) `$\mathbf{Op_2}$`(?(…), …))`
		`  …`
		`else if(?(…)) …`

CHOICE OF OP + ARGS

CONDITION

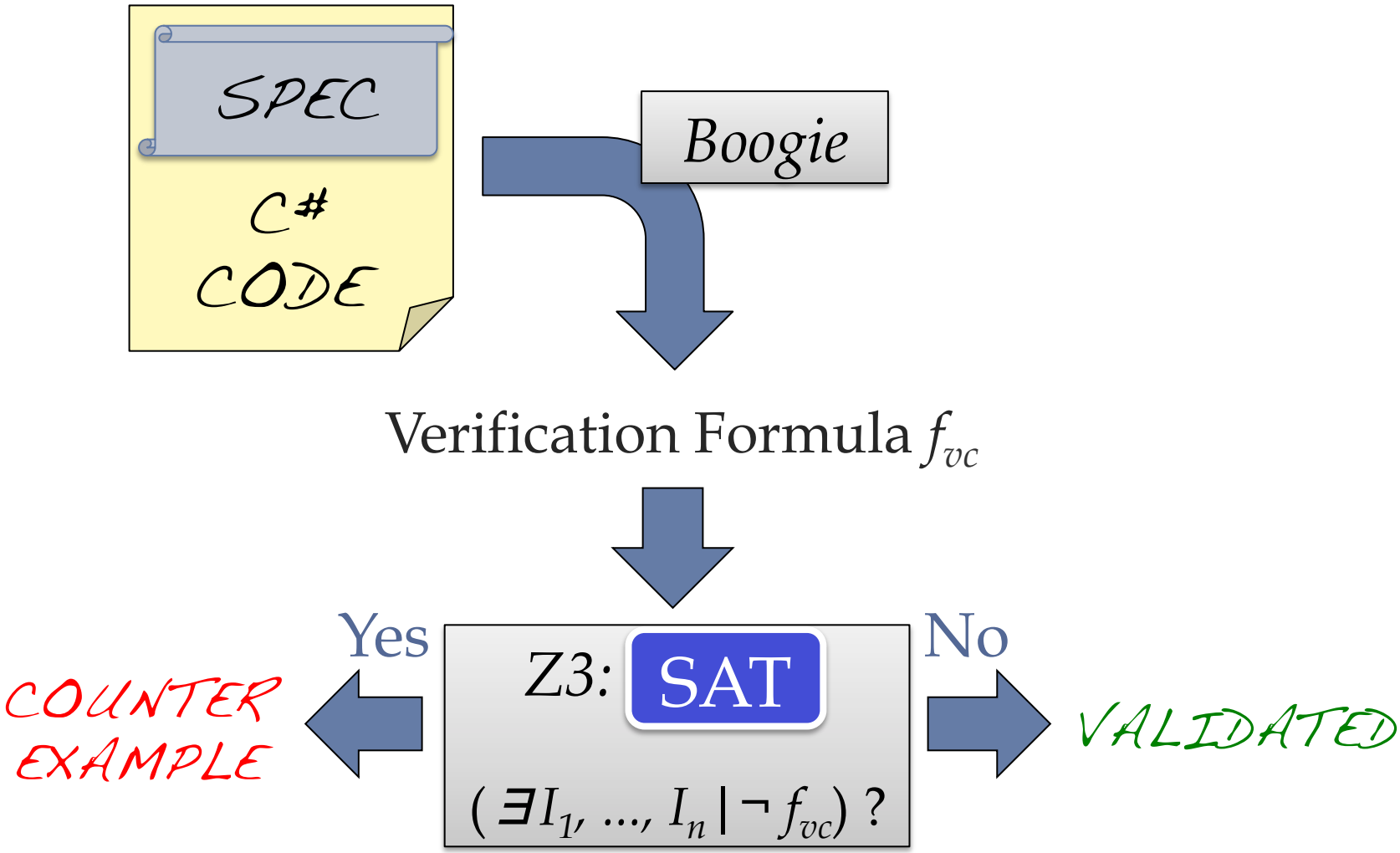
Sketch# Syntax

Option	Semantics
`?{branches: n}`	Upper-bound on # if branches
`?{calls: n}`	Upper-bound on # calls to ops
`?{ops: {…}}`	Possible ops
`?{vars: {…}}`	Possible vars to use in exprs
`?{conjuncts: n}`	Upper-bound on conds # $\wedge$
`?{disjuncts: n}`	Upper-bound on conds # $\vee$

This Talk

- Motivation
- Sketching
- Spec#
- Sketch#
  - Implementation
- Evaluation

Spec# Implementation: Static Verification



Sketch# Implementation


```
int intSqrt(int x) {
        int result;
        int v = 1, i = 1;
        while (v <= x) {
                v = ?;
                i = ?;
        }
        resu
        asse                = 0 && result * result <= x
             && (result + 1) * (result + 1) > x;
        return result;
}
```

QBF

SYNTHESIS FORMULA

VERIFICATION CONDITION

$$\exists X_1, \ldots, X_m \, \forall I_1, \ldots, I_n \mid f_{vc}$$

HOLES

INPUTS

Sketch# Implementation


```
int intSqrt(int x) {
        int result;
        int v = 1, i = 1;
        while (v <= x) {
                v = ?;
                i = ?;
        }
        resu
        asse                = 0 && result * result <= x
             && (result + 1) * (result + 1) > x;
        return result;
}
```

SAT

VERIFICATION CONDITION OVER FINITE SET OF INPUTS

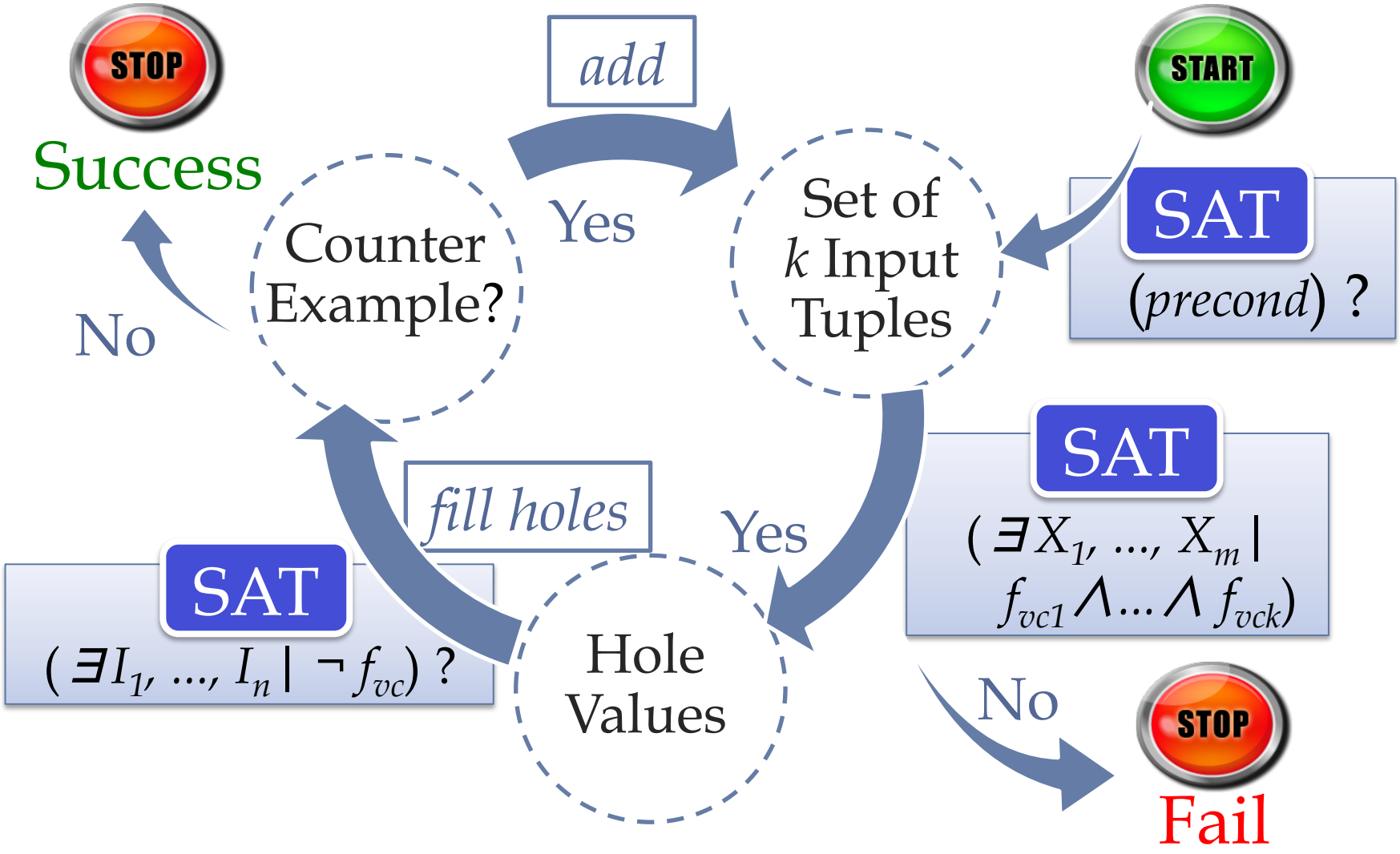
CEGIS FORMULA

$$\exists X_1, \ldots, X_m \mid f_{vc1} \wedge \ldots \wedge f_{vck}$$

HOLES

Sketch# Implementation: CEGIS

Solar-Lezama, Bodik PLDI'05



This Talk

- Motivation
- Sketch#
- Evaluation
  - Method

Sketching Commutativity Condition


```
class … {

    void Com_Cond(Object s1, Object s2, Op op1, Op op2)
        requires s1.Equals(s2);
        ensures Commute_Cond_Sketch(op1, op2) <==>
            s1.Equals(s2); {
        s1.Apply(op1); s1.Apply(op2);
        s2.Apply(op2); s2.Apply(op1);
    }
    boolean Commute_Cond_Sketch(Op op1, Op op2) {
        return ?();
    }
}
```

Sketching Commutativity Condition


```
class Stack {


  void Com_Cond(Stack o1, Stack o2, Push op1, Push op2)
        requires s1.Equals(s2);
        ensures Commute_Cond_Sketch(op1, op2) <==>
            s1.Equals(s2); {
        s1.Apply(op1); s1.Apply(op2);
        s2.Apply(op2); s2.Apply(op1);
    }
    boolean Commute_Cond_Sketch(Push op1, Push op2) {
        return ?();
    }
}
```

Start

Sketching Commutativity Condition


```
class Stack {

  void Com_Cond(Stack o1, Stack o2, Push op1, Push op2)
       requires s1.Equals(s2);
       ensures Commute_Cond_Sketch(op1, op2) <==>
           s1.Equals(s2); {
       s1.Apply(op1); s1.Apply(op2);
       s2.Apply(op2); s2.Apply(op1);
    }
    boolean Commute_Cond_Sketch(Push op1, Push op2) {
       return op1.val == op2.val;
    }
}
```

Sketching OT


```
class … {
    void OT(Object s1, Object s2, Op op)
      requires s1.Equals(s2);
      ensures s1.Equals(s2); {
      s1.Apply(op1); s1.Apply(OT_Sketch(op2, op1));
      s2.Apply(op2); s2.Apply(OT_Sketch(op1, op2));
    }
    Op[] OT_Sketch(Op op1, Op op2) {
        return ?{};
    }
}
```

Sketching OT


```
class Stack {
    void OT(Stack s1, Stack s2, Push op1, Push op2)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Apply(op1); s1.Apply(OT_Sketch(op2, op1));
        s2.Apply(op2); s2.Apply(OT_Sketch(op1, op2));
    }
    Op[] OT_Sketch(Push op1, Push op2) {
        return ?{};
    }
}
```

Start

Sketching OT


```
class Stack {
    void OT(Stack s1, Stack s2, Push op1, Push op2)
        requires s1.Equals(s2);
        ensures s1.Equals(s2); {
        s1.Apply(op1); s1.Apply(OT_Sketch(op2, op1));
        s2.Apply(op2); s2.Apply(OT_Sketch(op1, op2));
    }
    Op[] OT_Sketch(Push op1, Push op2) {
        if (op1.val >= op2.val)
            return {Push(op1.val)};
        else
            return {Pop();Push(op1.val);Push(op2.val)};
```

This Talk

- Motivation
- Sketch#
- Evaluation
  - Results

Inverse / ArrayList

Op	Sketch	Synthesized	Seconds
`Insert(i,v)`	`?{}`	`Delete(i)`	1
`Delete()`	`?{}`	*UNSAT*	2

Commutativity Condition / Stack

Op	Sketch	Synthesized	Seconds
Push(v_1) / Push(v_2)	?{conjuncts: 1}	$v_1 = v_2$	70
Push(v_1) / Pop()	?{conjuncts: 1}	*UNSAT*	2

Commutativity Condition / XMLNode

Op	Sketch	Synthesized	Sec.
InsertAfter(id_1,n_1) / InsertAfter(id_2,n_2)	?{disjuncts: 1, vars: {id_1, n_1.id, id_2, n_2.id}}	$id_1 \neq id_2$	67

Operational Transforms / Stack

Op	Sketch	Synthesized	Sec.

Push(v_1)
/
Push(v_2)
	?{branches:
	2,
	calls:
	3}
		if ($v_1 \geq v_1$)
		Push(v_1)
		else {
		Pop();
		Push(v_1);
		Push(v_2);
		}
			10

Operational Transforms / ArrayList

Op	Sketch	Synthesized	Sec.

Insert(i_1,v_1)
/
Insert(i_2,v_2)
	?{branches:3,
	ops:{Insert},
	args: {0},
	conjuncts: 2}
		if ($i_1 > i_2 \vee$
		($i_1 = i_2 \wedge v_1 \geq v_2$))
		Insert($i_1 + 1$, v_1)
		else
		Insert(i_1, v_1)
			240

Related Work

- Sketch [*Solar, Bodik PLDI'05-'07*]
  - + Specs / Modularity
- Spec# [*Barnett, Leino CASIS'04, ASE'07*]
  - + Sketching
- Formal Verification of
  - Inverse/Commuting Cond [*Kim, Rinard PLDI'11*]
  - Operational Transforms [*Imine '06*]
- Synthesizing Inverse
  - PINS [*Srivastava, Gulwani PLDI'11*]

In Conclusion…

- Sketch#: Synthesis in rich, modular specification & verification framework
- Synthesized Inv, Commutativity Conditions, & OT
- Highlighted need for automatic synthesis template refinement / expansion
  - Future work?
- SMT Solvers work magic now at *reasonable* time. Use them!

Specification-based Sketching in Sketch#

Try Sketch#:

http://www.cs.ucla.edu/~hesam/sketchsharp

Thank You!