Sample-Guided Automated Synthesis for CCSL Specifications

Ming Hu[1], Tongquan Wei[1], Min Zhang[1], Frédéric Mallet[2], and Mingsong Chen[1]

[1]Shanghai Key Lab of Trustworthy Computing, East China Normal University, China

[2]Université Côte d'Azur, CNRS, Inria, I3S, France

華東師範大學
EAST CHINA NORMAL UNIVERSITY

UNIVERSITÉ CÔTE D'AZUR

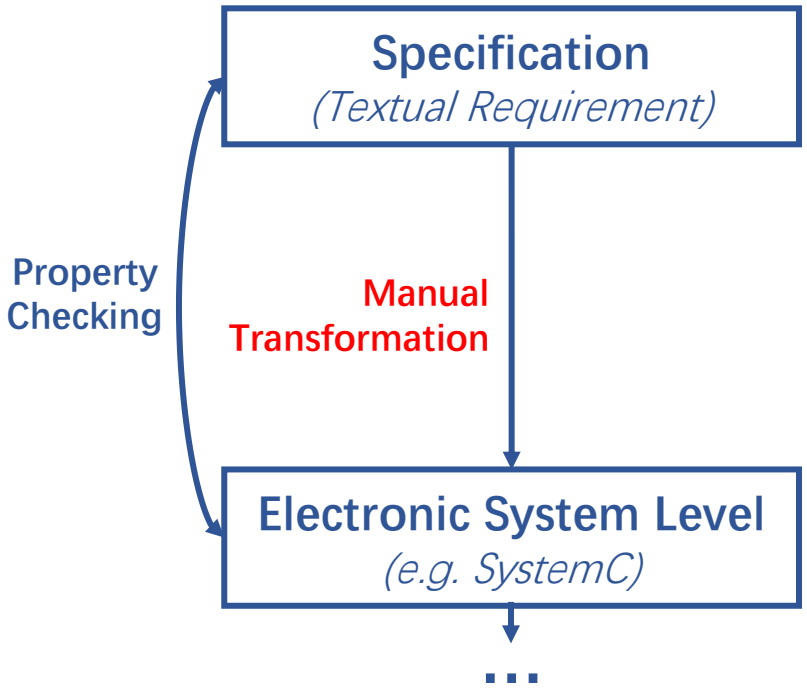
Synchron 2021

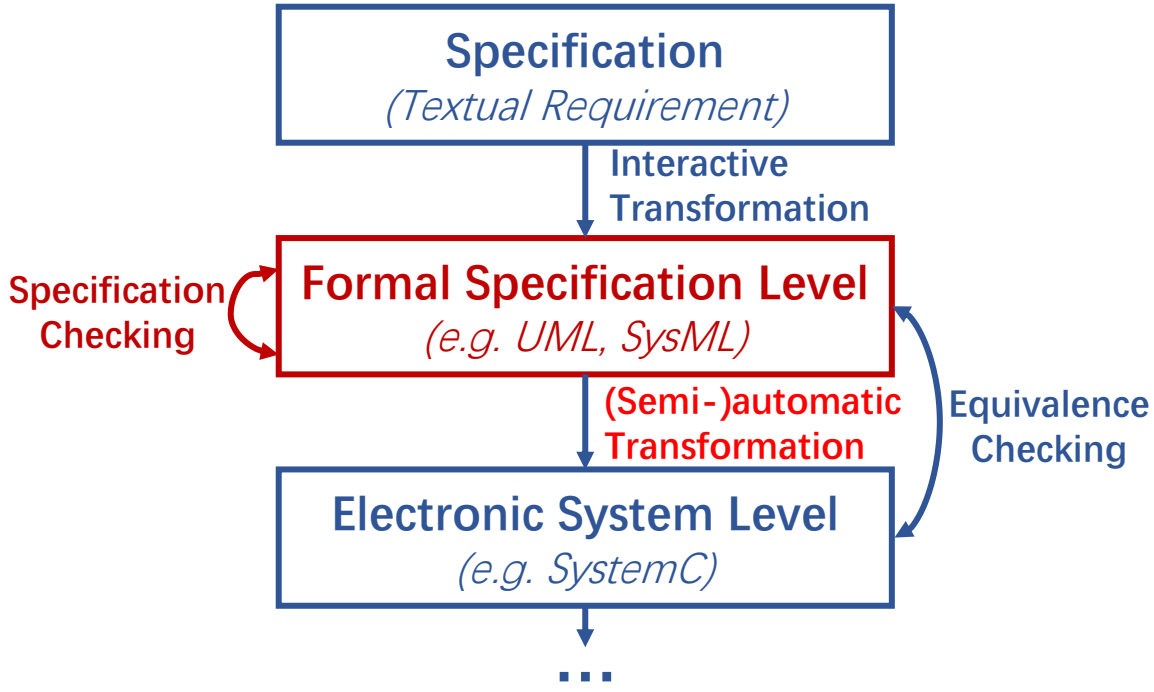
Outline

- Motivation
- Preliminary Knowledge of CCSL and SKETCH
- SKETCH-Based CCSL Synthesis Framework
  - Templates Generation for CCSL Sketching
  - Encoding for Incomplete Relations
  - Encoding for Incomplete Expressions
- Case Studies
- Conclusion

Motivation

- **FSL bridges the gap between specifications and ESL**
  - **The transformation from specifications to ESL designs can be automated**
  - **The correctness of ESL designs can be guaranteed**



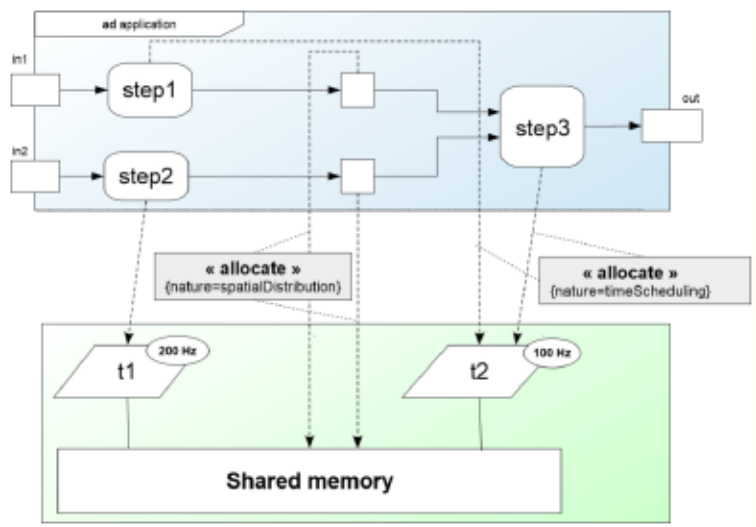


R. Drechsler, *et al*. ***Formal Specification Level: Towards verification-driven design based on natural language processing***. FDL, 53–58, 2012.

Clock Constraint Specification Language (CCSL)

- **CCSL is a formal specification language that**
  - **supports the formal modeling of timing behaviors**
  - **is a companion language for MARTE to handle logical clocks**

A CCSL example [SCP15]

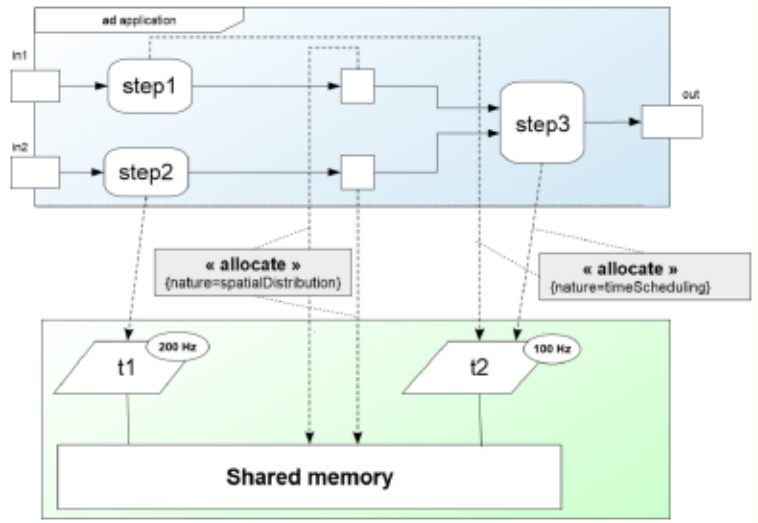


An example of spatial allocation and time scheduling

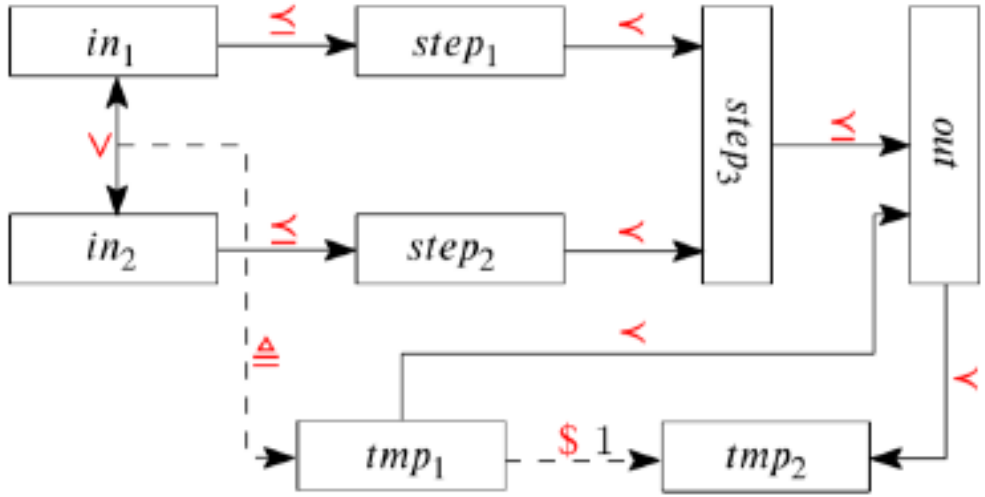
Clock Constraint Specification Language (CCSL)

- **CCSL is a formal specification language that**
  - **supports the formal modeling of timing behaviors**
  - **is a companion language for MARTE to handle logical clocks**

A CCSL example [SCP15]



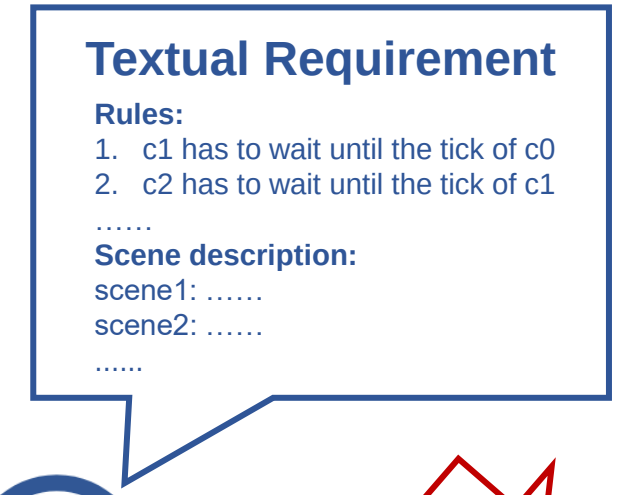
An example of spatial allocation and time scheduling

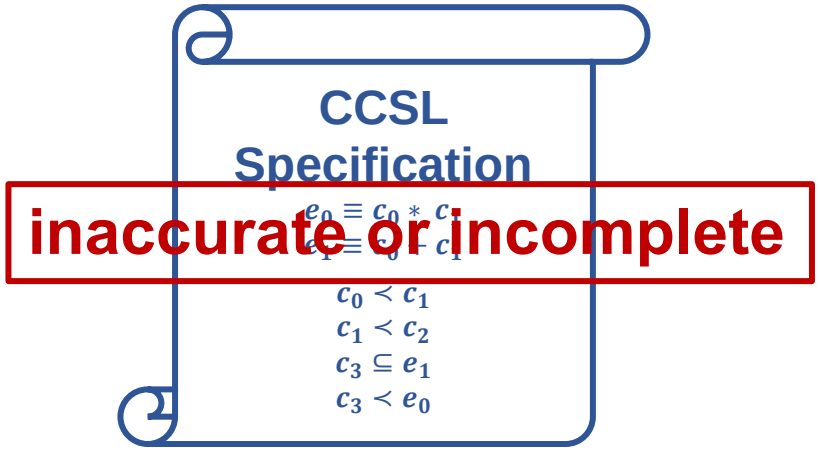


Constraints on events abstracted from the example

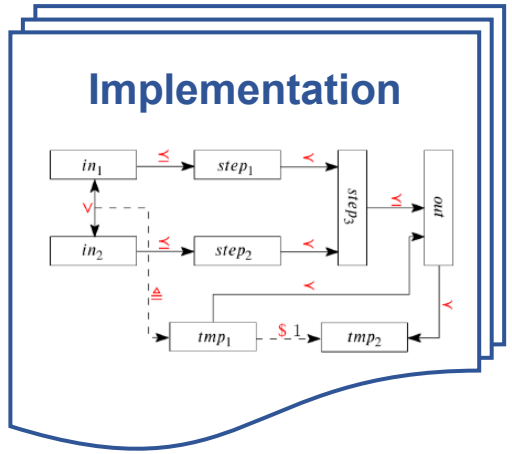
Motivation

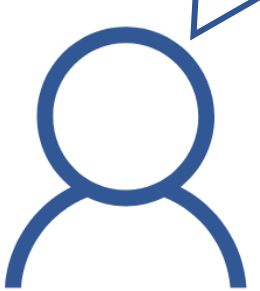
- **Challenges during the derivation of CCSL specifications**
  - **Limited expertise in formal timing modeling (inaccuracy)**
  - **Difficult to explore all the possible timing behaviors (incompleteness)**





inaccurate or incomplete





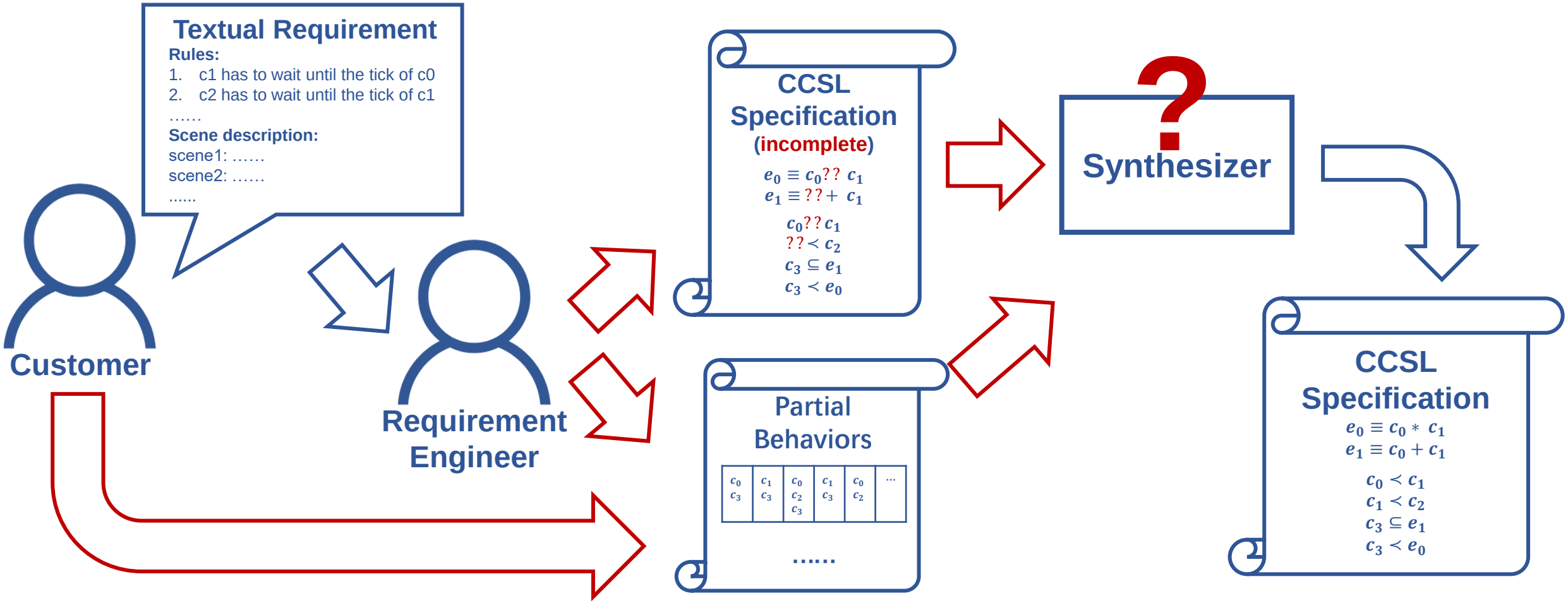
Customer

Requirement Engineer

Development Engineer

Motivation

- **Can we find an approach that automates the generation of accurate CCSL specifications?**



Syntax and semantics of CCSL

	Operators	Φ	$\rho \models \Phi$
Relation	Coincidence	$c_1 = c_2$	$\forall n \in \mathrm{N}^+. \chi_{c_1}^n = \chi_{c_2}^n$
	Precedence	$c_1 \prec c_2$	$\forall n \in \mathrm{N}^+. \chi_{c_1}^n = \chi_{c_2}^n \rightarrow c_2 \notin \rho^n$
	Causality	$c_1 \preccurlyeq c_2$	$\forall n \in \mathrm{N}^+. \chi_{c_1}^n \geq \chi_{c_2}^n$
	Subclock	$c_1 \subseteq c_2$	$\forall n \in \mathrm{N}^+. c_1 \in \rho^n \rightarrow c_2 \in \rho^n$
	Exclusion	$c_1 \ \# \ c_2$	$\forall n \in \mathrm{N}^+. c_1 \notin \rho^n \vee c_2 \notin \rho^n$
Expression	Union	$e = c_1 + c_2$	$\forall n \in \mathrm{N}^+. e \in \rho^n \leftrightarrow c_1 \in \rho^n \vee c_2 \in \rho^n$
	Intersection	$e = c_1 * c_2$	$\forall n \in \mathrm{N}^+. e \in \rho^n \leftrightarrow c_1 \in \rho^n \wedge c_2 \in \rho^n$
	Infimum	$e = c_1 \wedge c_2$	$\forall n \in \mathrm{N}^+. \chi_e^n = max\left(\chi_{c_1}^n, \chi_{c_2}^n\right)$
	Supremum	$e = c_1 \vee c_2$	$\forall n \in \mathrm{N}^+. \chi_e^n = min\left(\chi_{c_1}^n, \chi_{c_2}^n\right)$
	Delay	$e = c_1 \$ d$	$\forall n \in \mathrm{N}^+. \chi_e^n = max\left(\chi_{c_1}^n - d, 0\right)$
	Periodicity	$e = c_1 \propto p$	$\forall n \in \mathrm{N}^+. e \in \rho^n \leftrightarrow c_1 \in \rho^n \wedge \exists m \in \mathrm{N}^+. \chi_{c_1}^n = m \times p - 1$

Preliminary Knowledge of SKETCH

- **SKETCH is a language for finite programs with linguistic support for sketching (a software synthesis approach)**

An example for SKETCH-based Synthesis

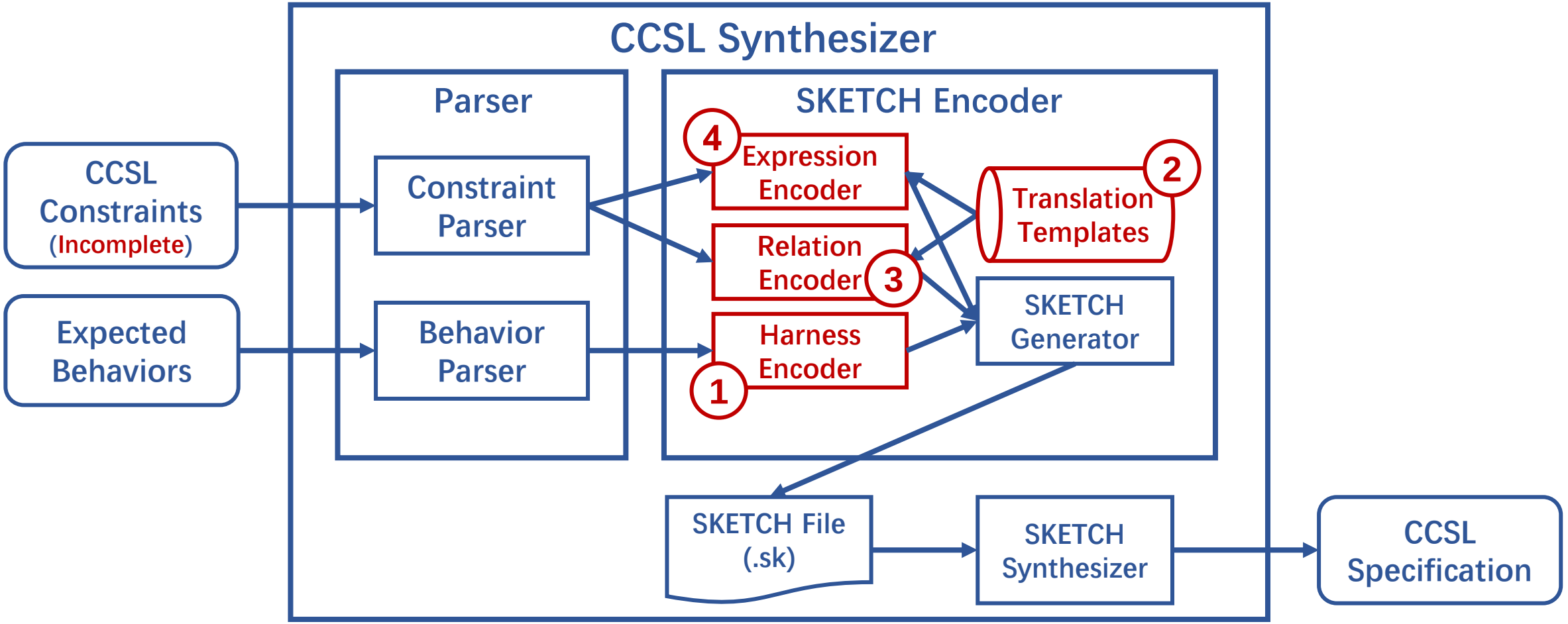
```
harness void main( int x ){
    int t = x * ??;
    assert t == x + x;
}
```

Sketching

```
void main( int x ){
    int t = x * 2;
}
```

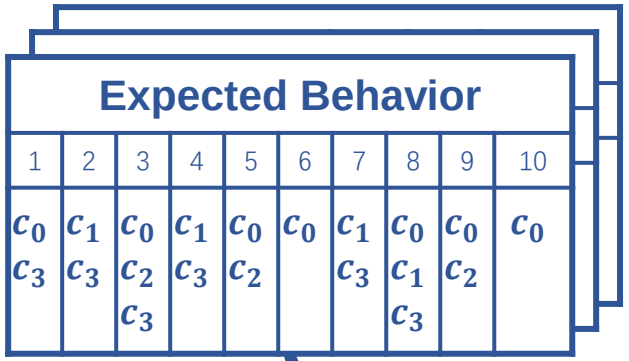
The keyword harness requires the SKETCH synthesizer to find a value for '??' (i.e., hole) that satisfies the assertion for all possible inputs x.

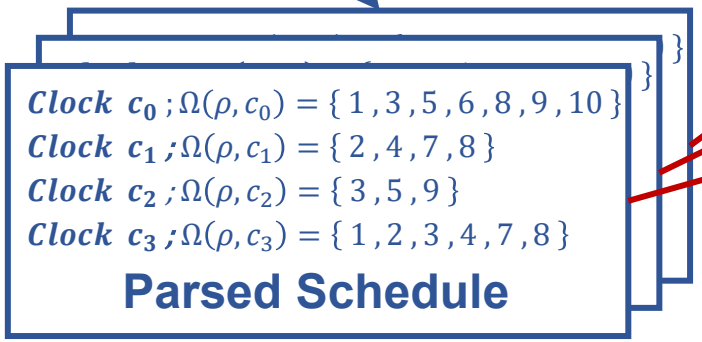
Our SKETCH-Based CCSL Synthesis Framework



SKETCH Implementation for CCSL Generation– An Overview

- **Harness function are used to synthesize incomplete CCSL specifications**
  - **Expected behaviors are encoded as the inputs for the Check Function**
  - **Check Function checks a behavior against all the CCSL specification rules**






```
harness void main{
    int c0_cnt, … , c3_cnt; int[10] c0, … , c3;
    ……// initialize cx_cnt, cx (0<=x<=n) for the 1st sample
    assert check(c0_cnt, … , c3_cnt, c0, … , c3) ==1;
    ……// initialize cx_cnt, cx (0<=x<=n) for the 2nd sample
    assert check(c0_cnt, … , c3_cnt, c0, … , c3) ==1;
    ……// initialize cx_cnt, cx (0<=x<=n) for the 3rd sample
    assert check(c0_cnt, … , c3_cnt, c0, … , c3) ==1;
    ……
}
```

The skeleton of a harness function

Check Function Implementation for Complete CCSL Spec

$Clock\ c_0; \Omega(\rho, c_0) = \{1, 3, 5, 6, 8, 9, 10\}$
$Clock\ c_1; \Omega(\rho, c_1) = \{2, 4, 7, 8\}$
$Clock\ c_2; \Omega(\rho, c_2) = \{3, 5, 9\}$
$Clock\ c_3; \Omega(\rho, c_3) = \{1, 2, 3, 4, 7, 8\}$

Parsed Schedule

Timing Behavior

1	2	3	4	5	6	7	8	9	10
c_0 c_3	c_1 c_3	c_0 c_2 c_3	c_1 c_3	c_0 c_2	c_0	c_1 c_3	c_0 c_1 c_3	c_0 c_2	c_0

```
1  int check( int c0_cnt, … ,int[10] c0,… ){
2    int e0_cnt, e1_cnt;
3    int[10] e0, e1;
5    e0_cnt= IntersectionCnt(c0_cnt,c1_cnt,c0,c1);
6    e0= Intersection(c0_cnt,c1_cnt,c0,c1);
7    e1_cnt= UnionCnt(c0_cnt,c1_cnt,c0,c1);
8    e1= Union(c0_cnt,c1_cnt,c0,c1);
9    ……
10   if(checkPrecedence(c0_cnt,c1_cnt,c0,c1)==0){
11     return 0;
12   }
13   if(checkPrecedence(c1_cnt,c2_cnt,c1,c2)==0){
14     return 0;
15   }
16   if(checkSubClock(c3_cnt,e1_cnt,c3,e1)==0){
17     return 0;
18   }
19   if(checkPrecedence(c3_cnt,e0_cnt,c3,e0)==0){
20     return 0;
21   }
22   return 1;
23 }
```

Check Function

CCSL Specification

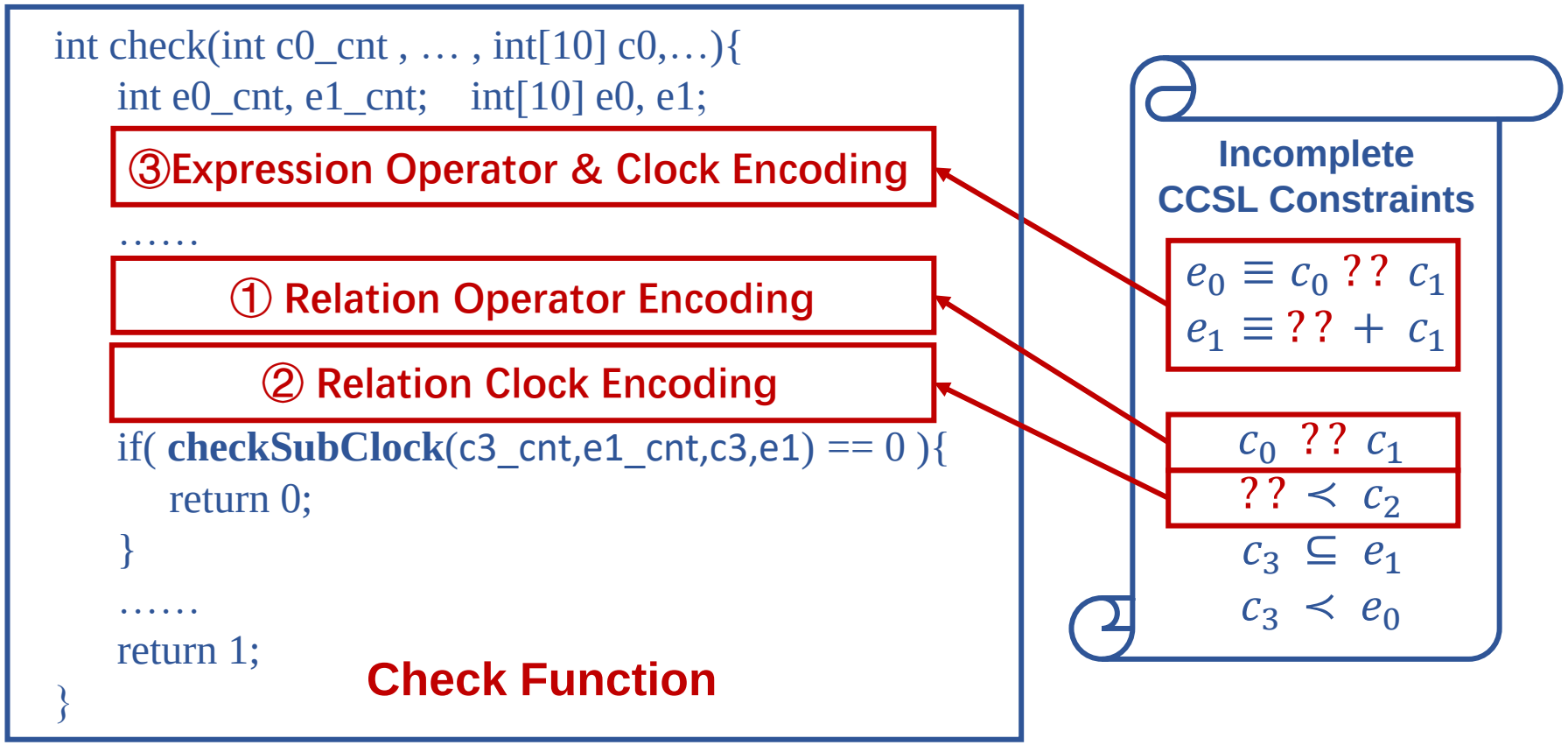
$e_0 \equiv c_0 * c_1$
$e_1 \equiv c_0 + c_1$

Expressions

$c_0 \prec c_1$
$c_1 \prec c_2$
$c_3 \subseteq e_1$
$c_3 \prec e_0$

Relations

Check Function Implementation for Incomplete CCSL Spec



Encoding Relation Operators

- **How to select a relation operator to fill a given hole $c_i ? ? c_j$?**
  - **The tightest one based on behavior refinement relations between operators**

$$c_i \prec c_j \rightarrow c_i \preccurlyeq c_j$$
$$c_i \subseteq c_j \wedge \chi^0_{c_i} \neq \chi^0_{c_j} \rightarrow c_j \prec c_i$$
$$c_i \subseteq c_j \rightarrow c_j \preccurlyeq c_i$$
$$c_i = c_j \rightarrow c_i \subseteq c_j$$

Refinement

$$' \prec' \sqsupseteq ' \preccurlyeq '$$
$$' \subseteq' \sqsupseteq ' \prec ' (\chi^0_{c_i} \neq \chi^0_{c_j})$$
$$' \subseteq' \sqsupseteq ' \preccurlyeq '$$
$$' =' \sqsupseteq ' \subseteq '$$

Operator	Coincidence $=$	Exclusion #	SubClock $\subseteq$	Precedence $\prec$	Causality $\preccurlyeq$
Priority	0	0	1	2	3

We choose the operator with the lowest Priority value

An Example of Relation Operators Encoding

c_0 ?? c_1

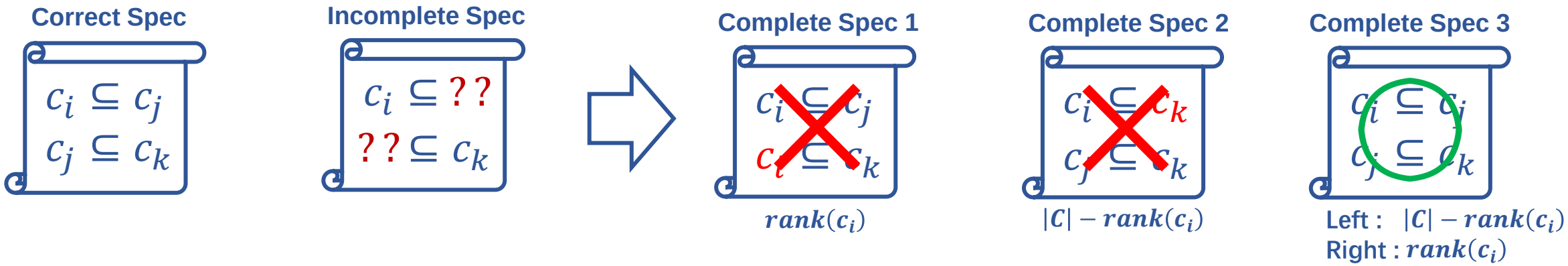
Relation	cond_0	priority_0
$c_0 = c_1$	0	0
$c_0 \prec c_1$	1	2
$c_0 \succ c_1$	2	2
$c_0 \preccurlyeq c_1$	3	3
$c_0 \succcurlyeq c_1$	4	3
$c_0 \subseteq c_1$	5	1
$c_0 \supseteq c_1$	6	1
$c_0 \# c_1$	7	0

```
1   int priority_0 = 0;
2   int cond_0 = ??;
3   assert cond_0 >= 0 && cond_0 < 8;
4   if ( cond_0 == 0 ){
5         priority_0 = 0;
6         if (checkCoincidence(c0_cnt, c1_cnt, c0, c1) == 0){ return 0; }
7   }
8   else if ( cond_0 == 1 ){
9         priority_0 = 2;
10        if (checkPrecedence(c0_cnt, c1_cnt, c0, c1) == 0){ return 0; }
11  }else …….
12  minimize(priority_0);
```

Choose the highest priority relation

Encoding Relation Clocks

- **How to select a clock to fill a given hole $??\ R\ c_j$?**
  - **Choose the clock with the least occurrence cardinality difference from the given clock while satisfying the corresponding operator relation**



$rank(c_i)$: the index of c_i in the sorted clock list (occurrence cardinalities in an ascending order)

Operator	Coincidence $=$	Exclusion #	SubClock $\subseteq$	Precedence $\prec$	Causality $\preccurlyeq$
LPriority	0	$\|C\| - rank(c_i)$	$\|C\| - rank(c_i)$	$rank(c_i)$	$rank(c_i)$
RPriority	0	$\|C\| - rank(c_i)$	$rank(c_i)$	$\|C\| - rank(c_i)$	$\|C\| - rank(c_i)$

We choose the clock with lowest LPriority/RPriority for Left/Right hole

An Example of Relation Clocks Encoding

$?? \prec c_2$

Clock	cond_1	priority_1
c_0	0	$6 - rank(c_0)$
c_1	1	$6 - rank(c_1)$
c_3	2	$6 - rank(c_3)$
e_0	3	$6 - rank(e_0)$
e_1	4	$6 - rank(e_1)$

```
1   ....//sort clocks based on occurrence and slack information
2   int priority_1 = 0;
3   int cond_1 = ??;
4   assert cond_1>=0 && cond_1<5;
5   if(cond_1 == 0){
6       priority_1 = rank[0];
7       if(checkPrecedence(c0_cnt, c2_cnt, c0, c2) ==0){return 0;}
8   }
9   else if(cond_1 == 1){
10      priority_1 = rank[1];
11      if(checkPrecedence(c1_cnt, c2_cnt, c1, c2) ==0){return 0;}
12  } else ......
13  minimize(priority_1);
```

Choose the clock with the lowest LPriority/RPriority

Encoding Expression Operators

- **How to select an expression operator to fill hole** $\begin{matrix} e \equiv c_i \,?\,?\, c_j \\ c_k \; R \;\; e \end{matrix}$ **?**
  - **Choose the operator that makes the clocks e and c_k have the least occurrence cardinality difference**

$$c_i + c_j \preccurlyeq c_i \vee c_j \preccurlyeq c_i \wedge c_j \preccurlyeq c_i * c_j$$

Operator	Intersection $*$	Supremum $\wedge$	Infimum $\vee$	Union $+$
$rank(e)$	0	1	2	3

Operator	Coincidence $=$	Exclusion $\#$	SubClock $\subseteq$	Precedence $\prec$	Causality $\preccurlyeq$
LPriority	0	$3 - rank(e)$	$3 - rank(e)$	$rank(e)$	$rank(e)$
RPriority	0	$3 - rank(e)$	$rank(e)$	$3 - rank(e)$	$3 - rank(e)$

We choose the operator that makes e with the lowest LPriority/RPriority

An Example of Expression Operators Encoding

$$e_0 \equiv c_0 \; ?? \; c_1$$

Expression	cond_e0	e0_op_priority
$e_0 \equiv c_0 + c_1$	0	3
$e_0 \equiv c_0 * c_1$	1	0
$e_0 \equiv c_0 \vee c_1$	2	2
$e_0 \equiv c_0 \wedge c_1$	3	1

$$c_3 \prec e_0$$

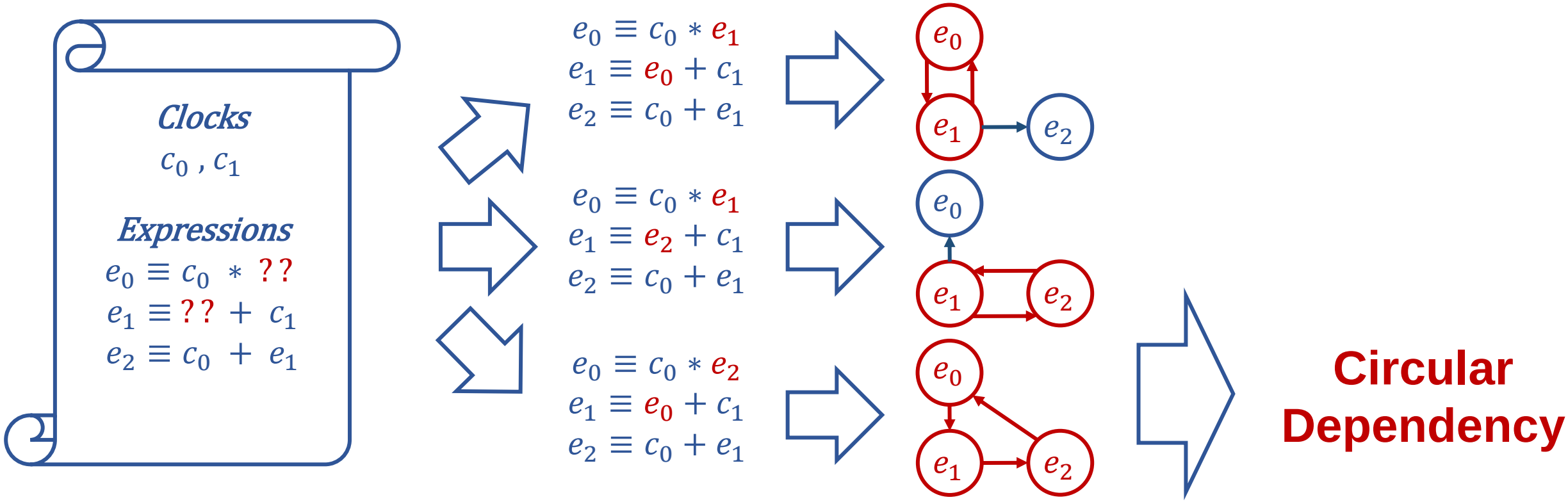
Operator	Precedence $\prec$
LPriority	$rank(e)$
RPriority	$3 - rank(e)$

```
1   int cond_e0 = ??;
2   assert cond_e0>=0 && cond_e0<4;
3   if(cond_e0 == 0){
4       e0_cnt = UnionCnt(c0_cnt, c1_cnt, c0, c1);
5       e0 = Union(c0_cnt, c1_cnt, c0, c1);
6       e0_op_priority = 3;
7   } else if(cond_e0 == 1){
8       e0_cnt = IntersectionCnt(c0_cnt, c1_cnt, c0, c1);
9       e0 = Intersection(c0_cnt, c1_cnt, c0, c1);
10      e0_op_priority = 0;
11  } else ......
12  int priority_3 = 3 - e0_op_priority;
13  if(checkPrecedence(c3_cnt, e0_cnt, c3, e0) ==0){return 0;}
14  minimize(priority_3);
```

Choose the operator that makes e_0 with the lowest LPriority/RPriority

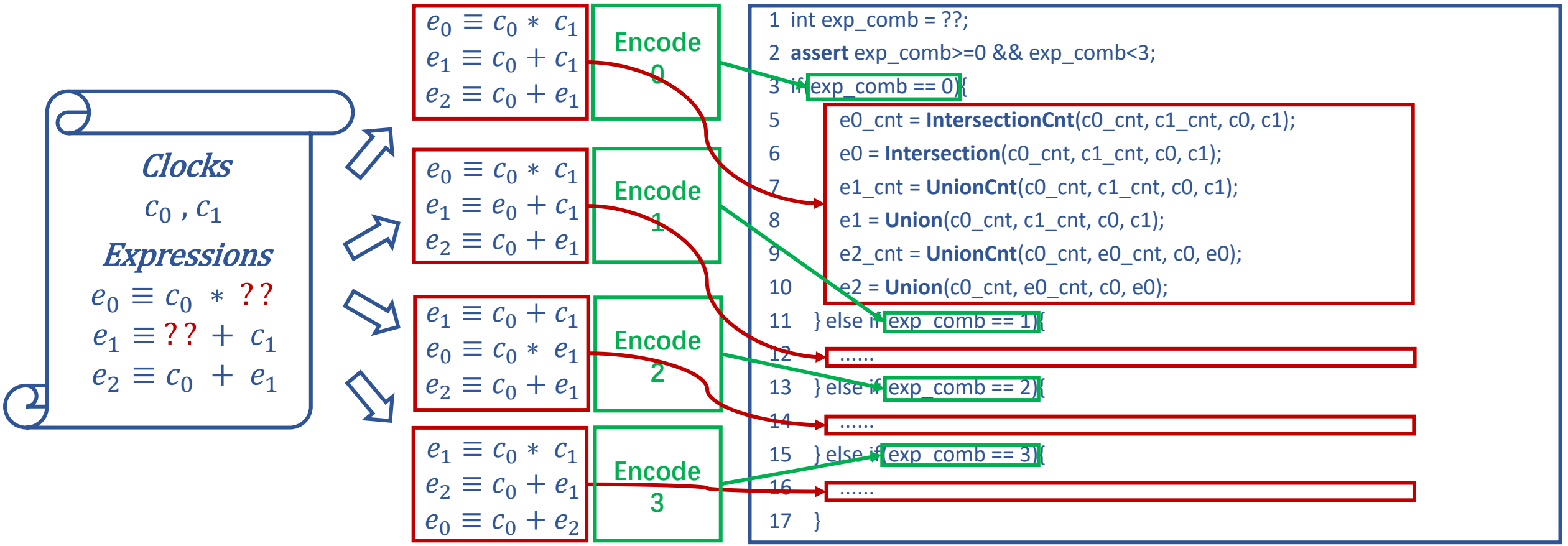
Encoding Expression Clocks

- **Challenges to encode expression clocks**
  - **Generate a large set of possibilities for expression combinations**
  - **Circular dependency (must be removed)**



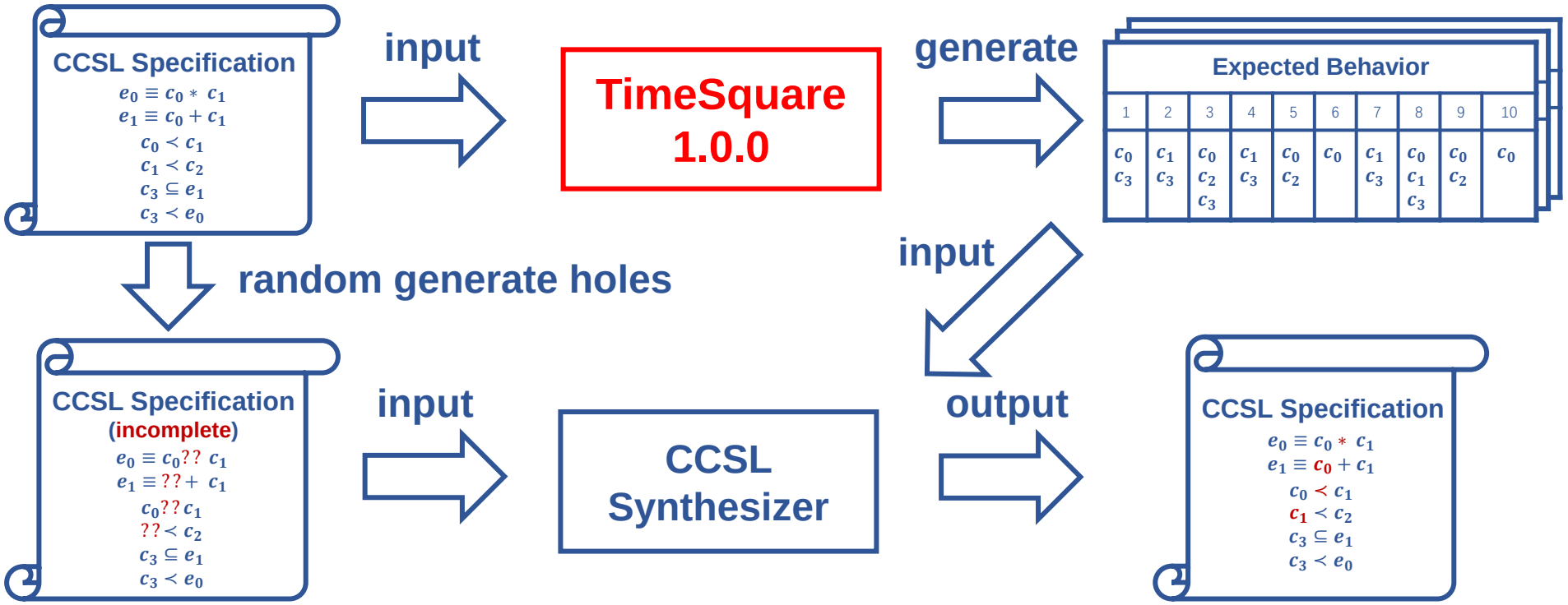
Example of Expression Clocks Encoding

$$e = c_i \; E \; ??$$



Experimental Settings

- Generates expected behaviors using TimeSquare (CCSL simulator)
- All the experiments were obtained on a Mac laptop with 2.5GHz Intel CPU and 16GB RAM



Case Studies - Synthesis Results

Source	Spec	Operator Type	Specification Statistics			Hole Settings			Time (ms)	Accuracy (%)	Same
			Clock	Expression	Relation	Clock	Expression	Relation			
TimeSquare # of Samples 5 Samples Length 50	s	Coincidence	2	0	1	0	0	1	82	100	Yes
	s_1	Precedence	2	0	1	0	0	1	510	100	Yes
	s_2	Causality	2	0	1	0	0	1	512	100	Yes
	s_3	Subclock	2	0	1	0	0	1	5019	100	Yes
	s_4	Exclusion	2	0	1	0	0	1	4420	100	Yes
	s_5	Union	3	1	1	0	1	0	1554	100	Yes
	s_6	Intersection	3	1	1	0	1	0	2515	100	Yes
	s_7	Infimum	3	1	1	0	1	0	2672	100	Yes
	s_8	Supremum	3	1	1	0	1	0	2584	100	Yes
	s_9	Delay	2	1	1	0	1	0	707	100	Yes
	s_{10}	Periodicity	2	1	1	0	1	0	713	100	Yes
Synthetic # of Samples 200 Samples Length 200	$Spec_1$	Composite	4	1	3	3	0	0	6907	100	Yes
						0	1	2	30961	100	Yes
	$Spec_2$	Composite	10	0	10	0	0	10	24608	100	Yes
						8	0	2	21468	100	Yes
	$Spec_3$	Composite	10	5	10	10	0	0	20038	100	No

Bisimulation with $Spec_3$

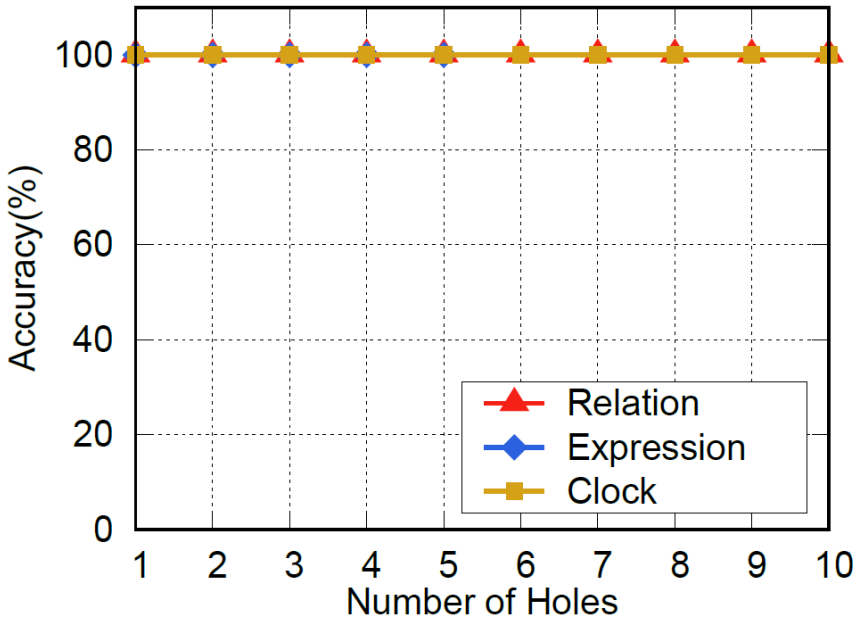
Our approach can achieve the highest accuracy!

Case Studies – Impact Factors of Synthesis Accuracy & Time

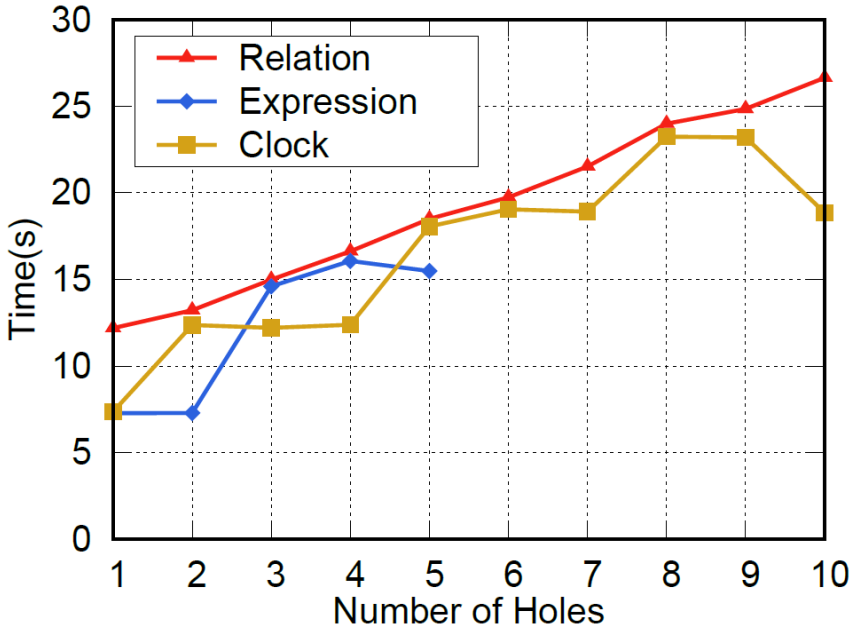
- **Impacts of the types and number of holes**
  - **Generally more holes require more synthesis time**

Specification	Operator Type	Specification Statistics		
		Clock	Expression	Relation
$Spec_3$	Composite	10	5	10

# of Samples: 5
Length of Samples : 50



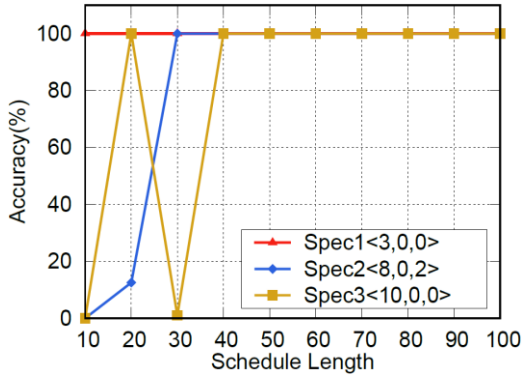
(a) Accuracy information



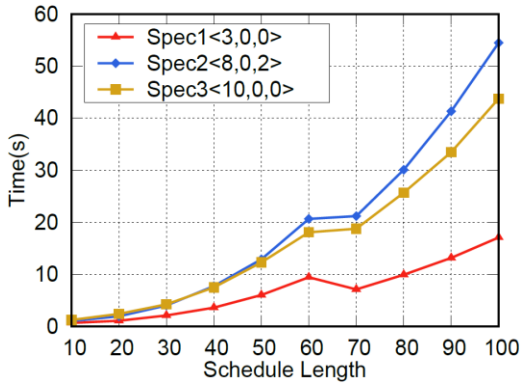
(b) Time information

Case Studies – Impact Factors of Synthesis Accuracy & Time

$spec\langle x, y, z\rangle$ indicate holes settings for <clocks, expression operators, relation operators>



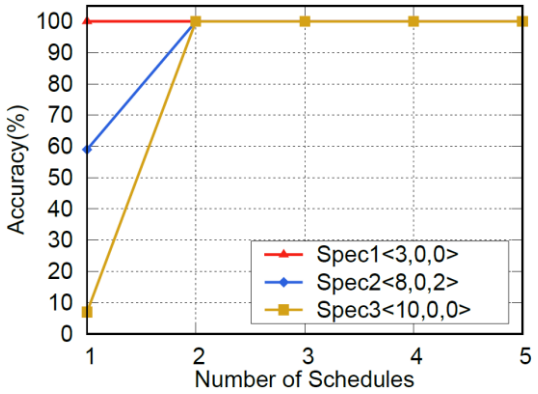
(a) Accuracy information



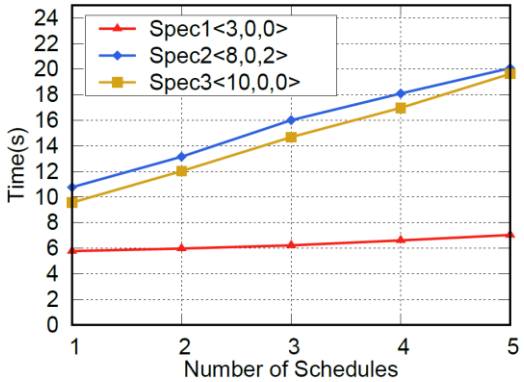
(b) Time information

# of Sampled Behaviors Number : 2

- **Impacts of schedule lengths**
  - Our approach can achieve the highest synthesis accuracy
  - Longer samples will lead to better accuracy



(c) Accuracy information



(d) Time information

Length of Sampled Behaviors : 50

- **Impacts of the number of samples**
  - Our approach can achieve the highest synthesis accuracy
  - More samples will lead to better accuracy

Conclusion

- **Automated CCSL specification synthesis is hard**
  - Limited expertise in formal timing modeling
  - Difficult to explore all the possible timing behaviors of systems
- **Contribution of this paper**
  - A SKETCH-based framework that automates CCSL synthesis
  - Effective priority policies to generate the tightest CCSL constraints
- **Experimental results**
  - Results on benchmarks collected from TimeSquare and complex synthetic examples show the effectiveness of our approach

Thank You !