UC Santa Barbara
Computer Science Department

VIEW: An Incremental Approach to Verify Evolving Workflows

Mohammad Javad Amiri, Divyakant Agrawal

Workflows and Workflow Verification

- A *workflow* consists of a set of activities performed in coordination in an organizational Environment to accomplish a business goal
- Workflow Verification: Determining whether a workflow exhibits certain desirable behaviors
  - Complete execution, i.e., Soundness
  - Consistent data
  - User-defined properties, e.g., LTL Constraint
- Workflow verification has high complexity
- In presence of data the general verification problem is unsolvable

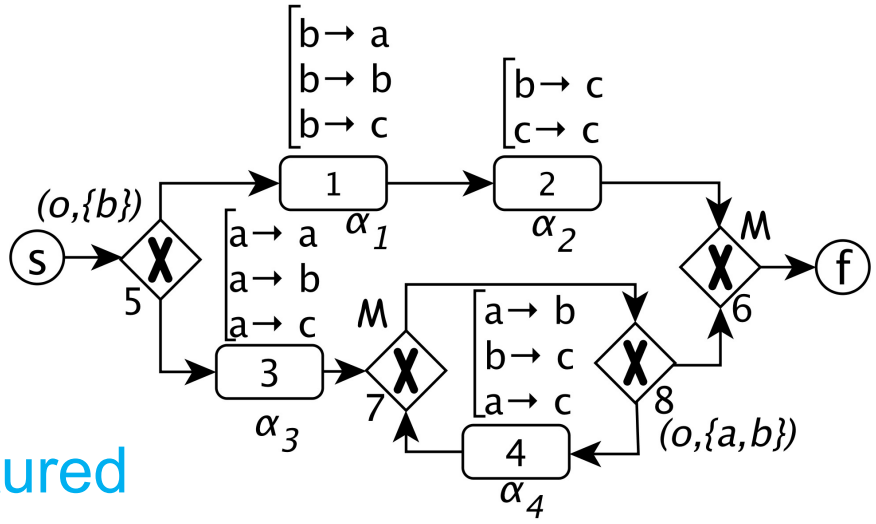
Even worse, Workflows need to be changed

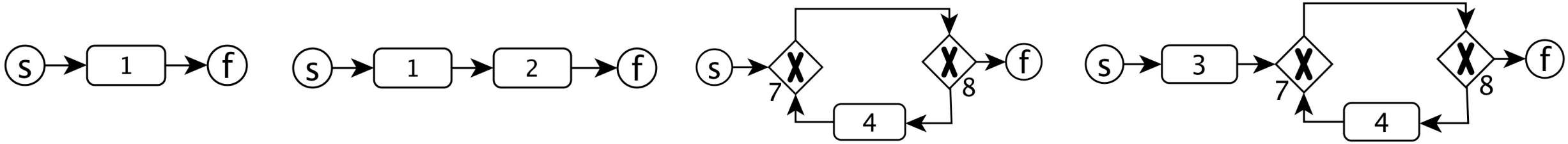
- Workflows need to be changed to react quickly and adequately to events
- Every change could affect the correctness of workflow
- The evolved workflow has to be verified again!
- Do we really need to verify the entire workflow after each change?
  - **No**
- Incremental Verification: Verify an evolved workflow without checking every node

Problem: Given a *workflow schema*, a *change operation*, and a *set of constraints*, do all possible executions of the *evolved workflow* satisfy the constraints?

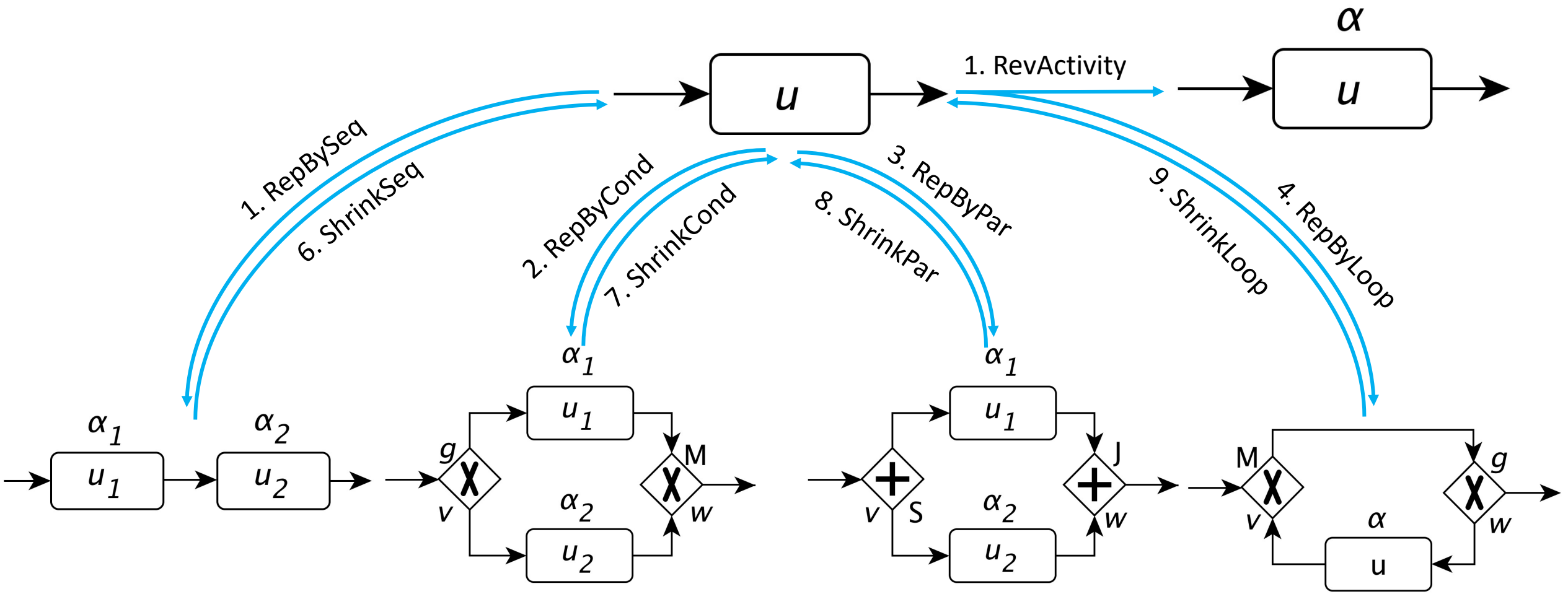
(Object-aware) Workflow Schema

- P = (N, s, f, L, E, O)
  - Each object in O has a set of states
  - Activity: a set of objects and transitions: (α,O,τ)
  - Choice gateway: an object and a set of states: (o,χ)
- Workflow schemas are restricted to be well-structured
  - Either and activity or a composite block
  - A composite block: two *sequential*, two *conditional*, two *parallel*, or a *loop* sub-block
- Workflow schemas are constructed recursively





Workflow Change Operations

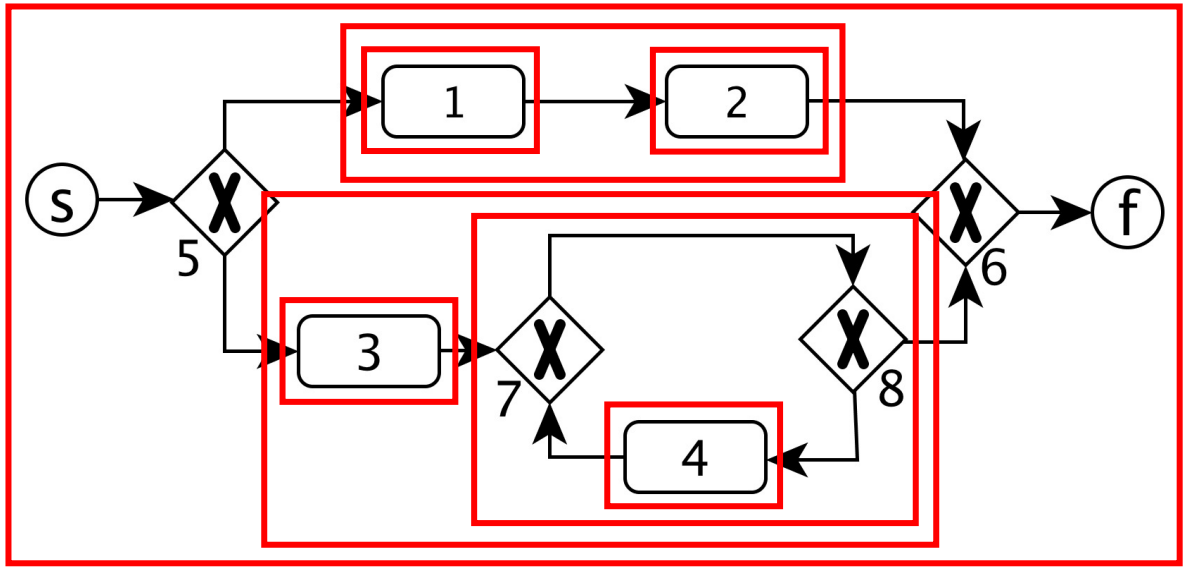


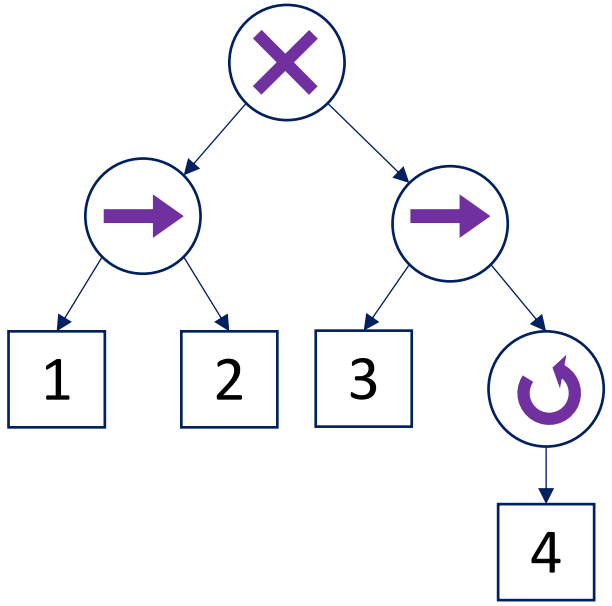
DecSerFlow Constraints

1. Cardinality	Lb(α): minimum number of occurrences of activity α Ub(α): maximum number of occurrences of activity α Fix(α): exact number of occurrences of activity α Rng(α): range of occurrences of activity α		
2. Existence	Ex(α,ß): each occurrence of activity α implies an occurrence of activity ß coEx(α,ß): either both α, ß or none of them are occurred		
	Response	Precedence	Succession
3. Ordering	Res(α, ß): each occurrence of α is followed by an occurrence of ß	Pre(α, ß): each occurrence of ß is preceded by an occurrence of α	Res(α, ß) and Pre(α, ß)
4. Alternating	aRes(α, ß): in addition to Res(α, ß) α and ß alternate	aPre(α, ß): in addition to Pre(α, ß) α and ß alternate	aRes(α, ß) and aPre(α, ß)
5. Chain	cRes(α, ß): each occurrence of α is immediately followed by an occurrence of ß	cRes(α,ß): each occurrence of ß is immediately preceded by an occurrence of α	cRes(α,ß) and cPre(α,ß)

Process Tree

- Each Workflow schema *P* has a corresponding process tree *T(P)*
- Process Tree is constructed recursively
- Each Block in *P* has a corresponding sub-tree in *T(P)*





Incremental Verification

Verify an evolved workflow without checking all the nodes

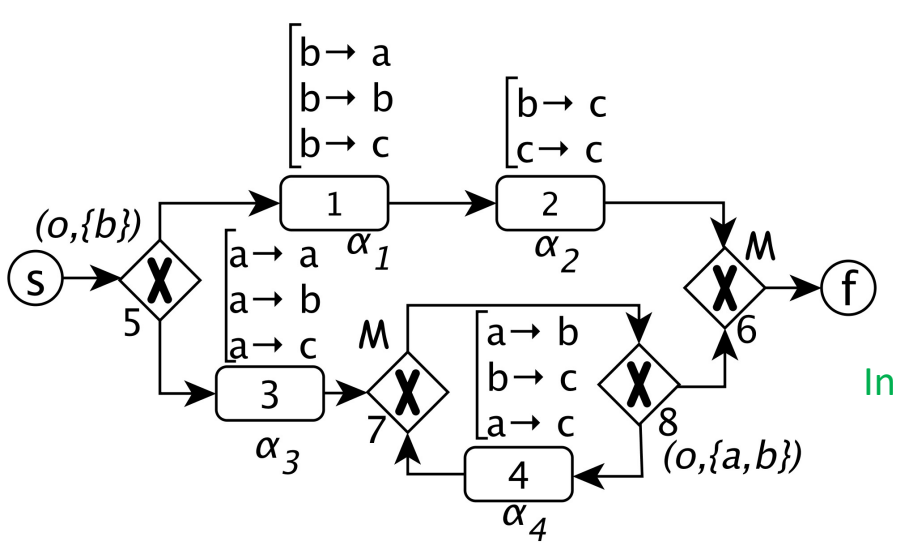
For each node within a process tree an auxiliary data is stored

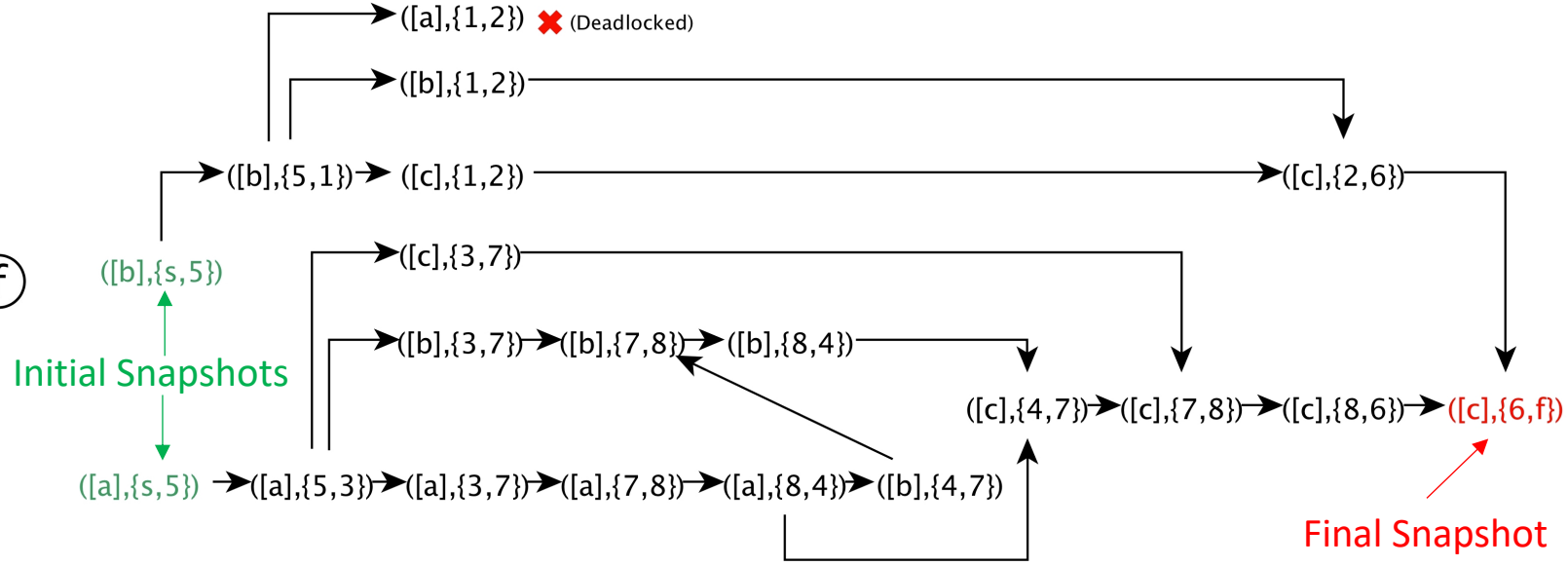
Auxiliary data stores are used during the verification step

Auxiliary data stores keep track of *the execution paths*

Execution Paths

A *snapshot* of schema P: Σ = (D, I) where D : O ⟼ S assigns each object in P a state, and I is a set of edges in P.





- The workflow has totally six complete executions
  - two paths from ([b], {s, 5}) to ([c], {6, f })
  - four paths from ([a], {s, 5}) to ([c], {6, f })

Auxiliary Data Store

- Given a workflow schema P, let r be the root node of the corresponding tree T(P),

An auxiliary data store of node r is a relation D_r consists of tuples (x,y,C)

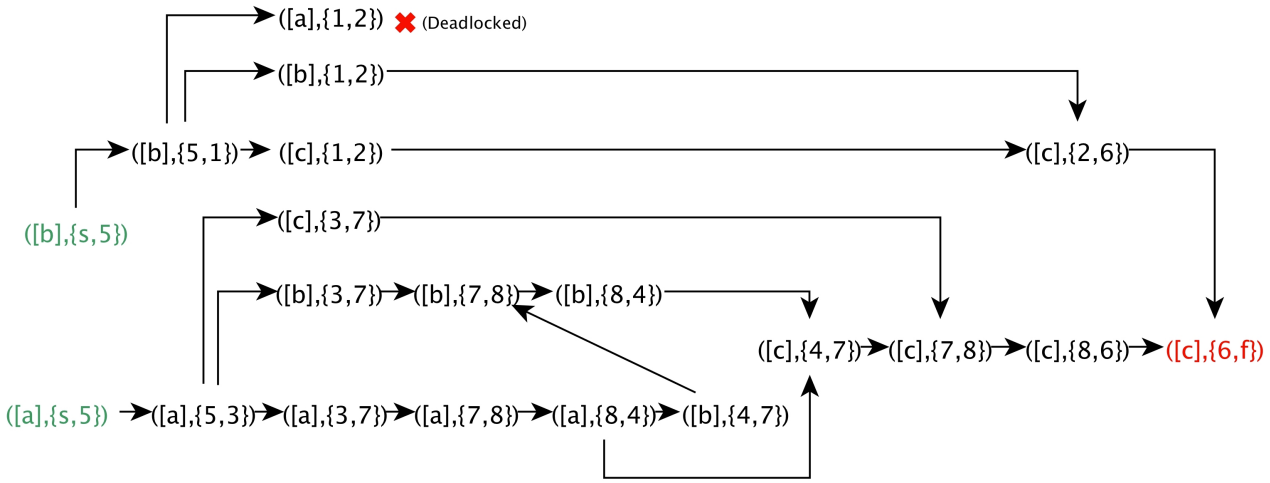
  - x, y are two state (relations)
  - C counts execution paths from an initial snapshot with state x to a final snapshot with state y

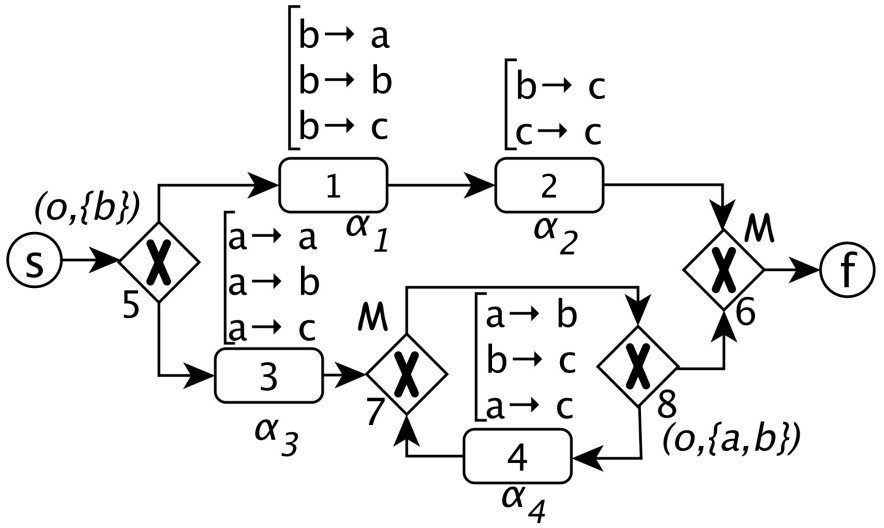
- Element C is defined based on the class of the given constraint

Count C of Auxiliary data Stores

Class	*Example*	C	Explanation
Cardinality	*Lb*(*α*)	$\{c_1,\ldots c_n\}$	each *c* in *C* shows the number of *α*'s in a distinct path
Existence	Ex(α, ß)	C[4]	*C*[0], *C*[1], *C*[2] and *C*[3] are the number of distinct paths consisting *α* but not *ß*, *ß* but not *α*, both *α* and *ß*, and neither *α* nor *ß* respectively
Ordering	Res(α, ß)	C[6]	*C*[0]: there is an *α* without a following *ß*, and each *ß* is preceded by an *α* *C*[1]: there is a *ß* without a preceding *α*, and each *α* is followed by a *ß* *C*[2]: there is an *α* without a following *ß*, and there is a *ß* without a preceding *α* *C*[3]: each *α* is followed by a *ß*, and each *ß* is preceded by an *α* *C*[4]: paths have neither *α* nor *ß* *C*[5]: paths that never satisfy the constraint
Alternating	aRes(α, ß)	C[6]	similar to ordering, except that the following and preceding *α* and *ß* are distinct
Chain	aRes(α, ß)	C[7]	similar to ordering, except that the following and preceding *α* and *ß* are immediate *C*[6]: paths with no activity nodes

Count C of Auxiliary data Stores





Cardinality	*Lb*(α)	$\{c_1,\ldots c_n\}$	each *c* in *C* shows the number of α's in a distinct path

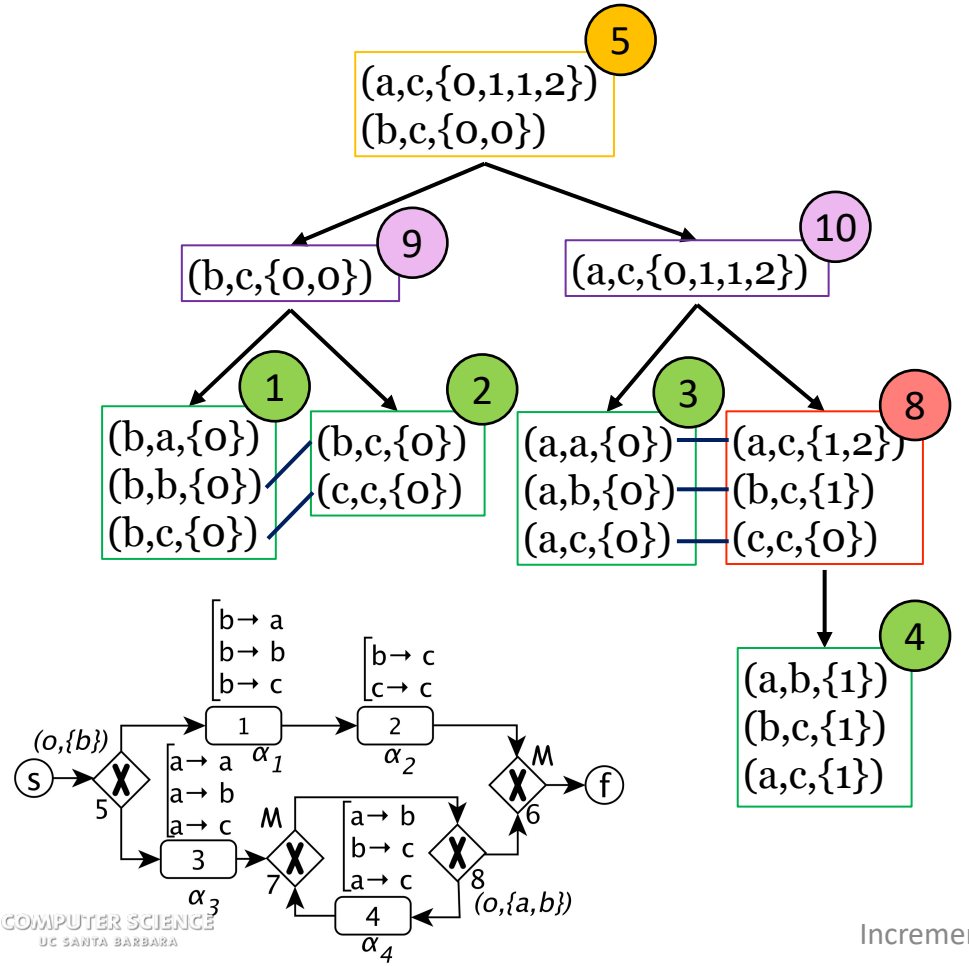
- Cardinality Constraint *Ub(4) = 1* ⇒ D = {(a,c,{0, 1, 1, 2}), (b,c,{0, 0})}

Existence	Ex(α, ß)	C[4]	*C*[0], *C*[1], *C*[2] and *C*[3] are the number of distinct paths consisting α but not ß, ß but not α, both α and ß, and neither α nor ß respectively

- Existence Constraint *Ex(3,4)* ⇒ D = {(a,c,[1,0,3,0]), (b,c,[0,0,0,2])}

Bottom Up Construction of Auxiliary Data

Cardinality Constraint (activity 4)



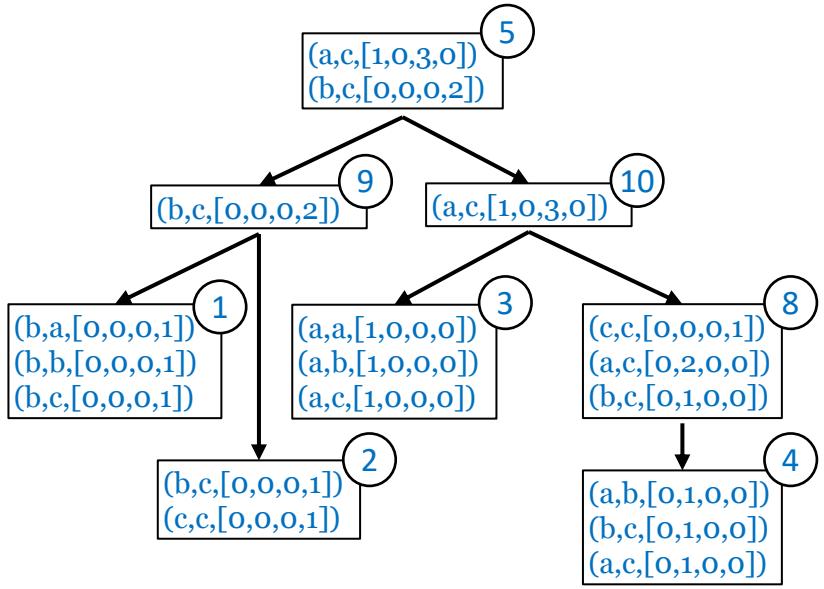
u is an *activity* node where $L(u) = (\alpha, O, \tau)$:
for all *(x,y)* in τ: Add *(x,y,C)* to D_u

u is a *sequence* node with child nodes v_1 and v_2:
$D_u(x,y, f_Q(C_1,C_2,\varphi)) \longleftarrow D_{v1}(x,z,C_1), D_{v2}(z,y,C_2)$

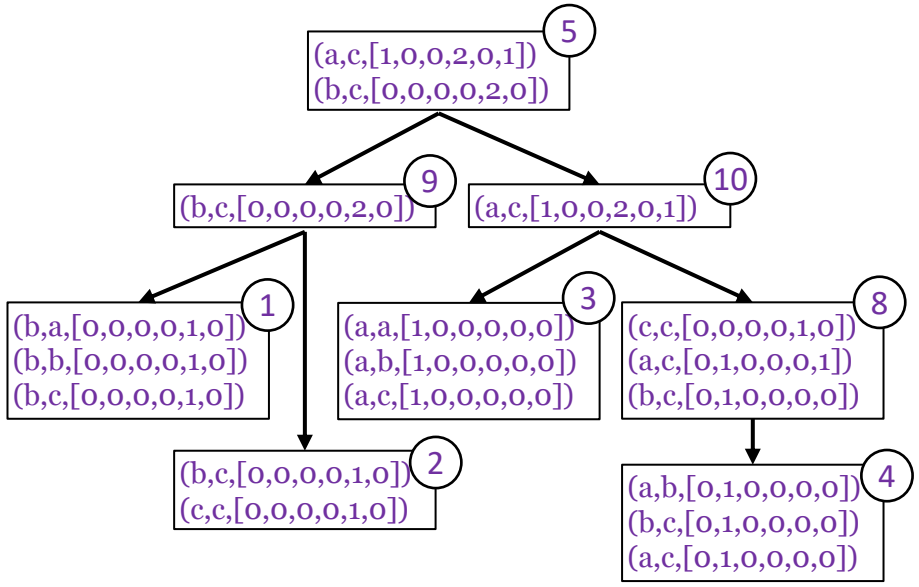
u is a *loop* node with a child v_1 where $L(u) = (x_i, \chi)$:
$T_u(x,y,C) \longleftarrow D_{v1}(x,y,C), \chi_1(x_i)$
$T_u(x,y, f_Q(C_1,C_2,\phi)) \longleftarrow Tu(x,z,C_1), D_{v1}(z,y,C_2), \chi_1(x_i)$
$T_u(x,x,C_0(\phi)) \longleftarrow (x,x), \chi_2(x_i)$
$D_u(x,y,C) \longleftarrow T_u(x,y,C), \chi_2(y_i)$

u is a *conditional* node with child nodes v_1 and v_2:
$D_u(x,y,C) \longleftarrow D_{v1}(x,y,C), \chi_1(x_i)$
$D_u(x,y,C) \longleftarrow D_{v2}(x,y,C), \chi_2(x_i)$

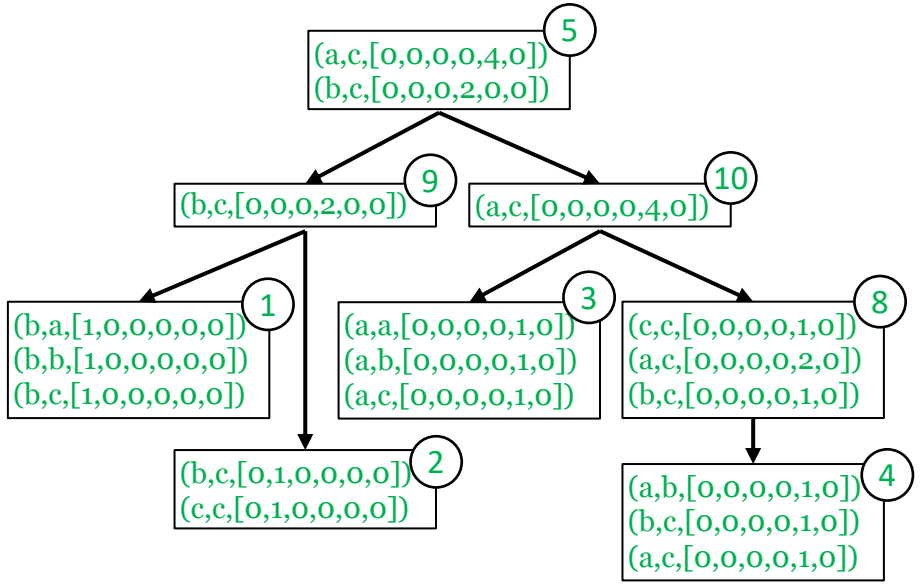
u is a *parallel* node with child nodes v_1 and v_2:
$Du(x,y, f_P(C_1,C_2,\phi)) \longleftarrow D_{v1}(x,z,C_1), D_{v2}(z,y,C_2)$



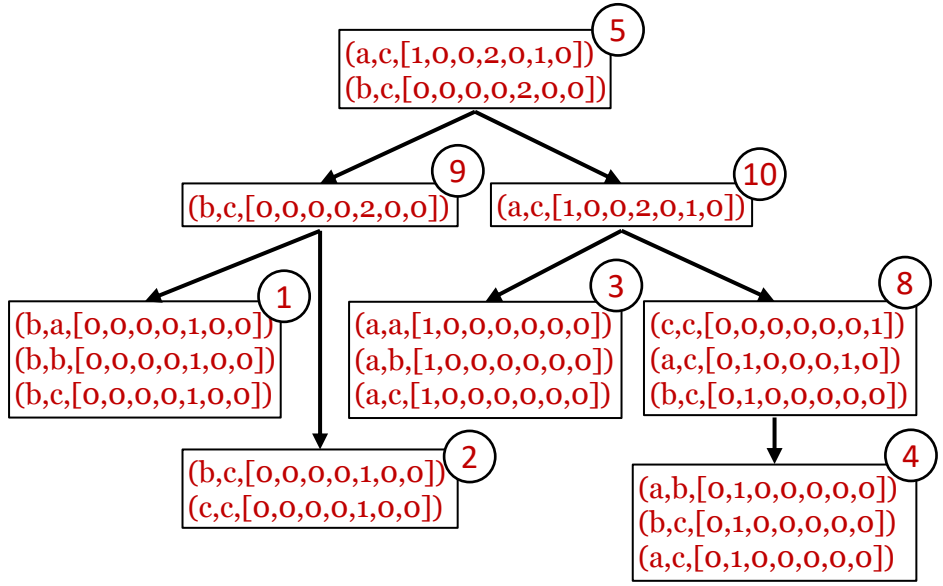
Existence Constraint $Ex(\alpha_3,\alpha_4)$



Alternating Constraint $\alpha Res(\alpha_3,\alpha_4)$



Ordering Constraint $Res(\alpha_1,\alpha_2)$



Chain Constraint $cPre(\alpha_3,\alpha_4)$

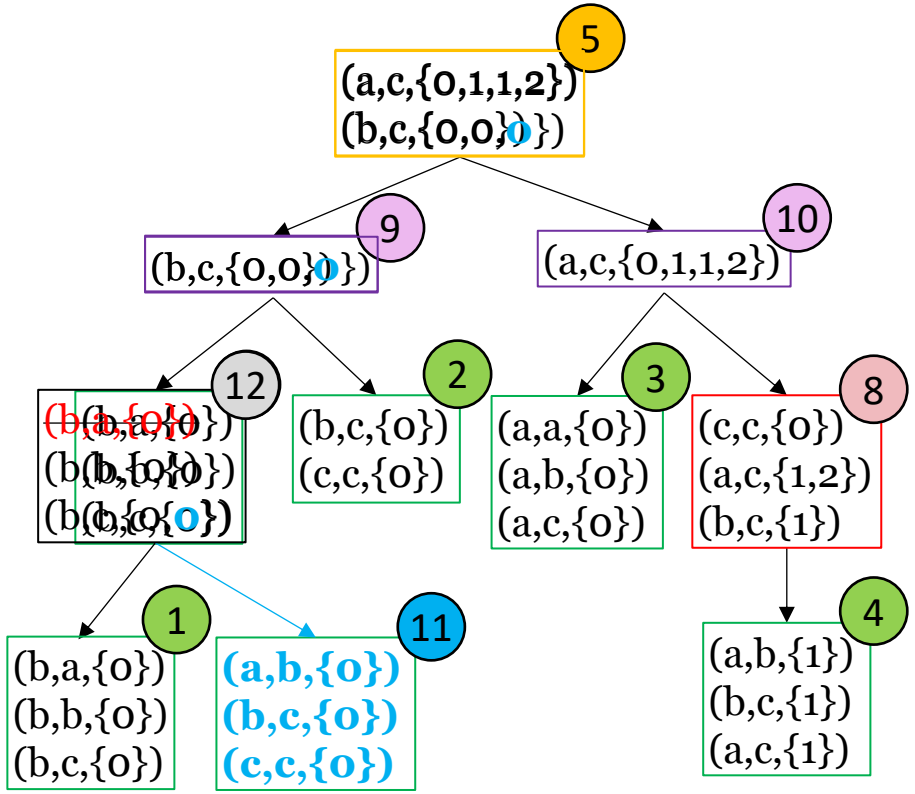
Incremental Construction of Auxiliary Data

- The leaf level (activity) nodes and their auxiliary data are updated
- The changes are propagated incrementally only along the path to the root node
- For each node, two relations D^+ and D^- are defined
  - D^+: elements which are added
  - D^-: elements which are removed
- $D' = D - D^- + D^+$

Incremental Construction of Auxiliary Data

Change Operation: RepBySeq(1,11)

Cardinality Constraint



Verification of Constraints

- Only check the auxiliary data of the *root node* to verify a given constraint

Cardinality	Lb(α)	For all c in C : c ≥ Lb(α)
	Ub(α)	For all c in C : c ≤ Lb(α)
	Fix(α)	For all c in C : c = Fix(α)
	Rng(α)	For all c in C : c ∈ Rng(α)
Existence	Ex(α,ß)	C[0] = 0
	coEx(α,ß)	C[0] = C[1] = 0
Ordering	Res(α,ß)	C[0] = C[2] = 0
	Pre(α,ß)	C[1] = C[2] = 0
	Suc(α,ß)	C[0] = C[1] = C[2] = 0
Alternating	aRes(α,ß)	C[0] = C[2] = C[5] = 0
	aPre(α,ß)	C[1] = C[2] = C[5] = 0
	aSuc(α,ß)	C[0] = C[1] = C[2] = C[5] = 0
Chain	cRes(α,ß)	C[0] = C[2] = C[5] = 0
	cPre(α,ß)	C[1] = C[2] = C[5] = 0
	cSuc(α,ß)	C[0] = C[1] = C[2] = C[5] = 0

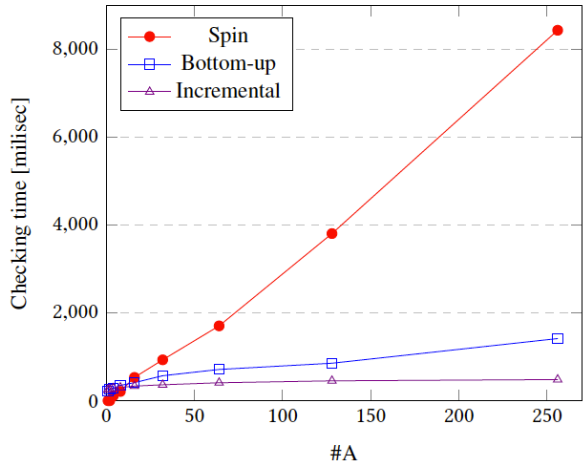
- Cardinality Constraint "*Ub(4) = 1*" is *not verified*
  - D = {(a,c, {0, 1, 1, 2}), (b, c, {0, 0})} ⟹ there is path with more than one occurrences of 4
- Existence Constraint *Ex(3,4)* is *not verified*
  - D = {(a, c, [1, 0, 3, 0]), (b, c,[0, 0, 0, 2])} ⟹ there is path that goes through 3, but not 4

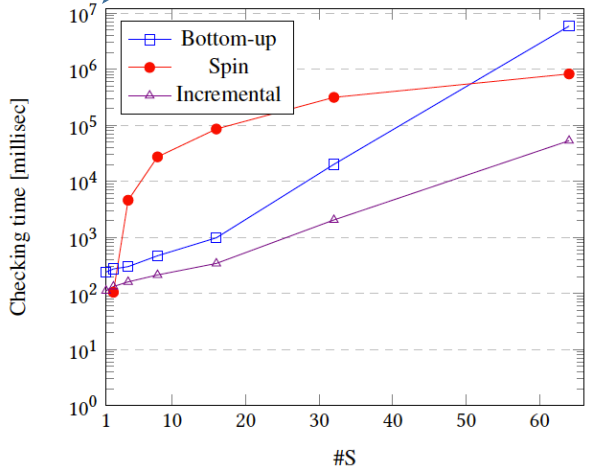
Experiments Setup

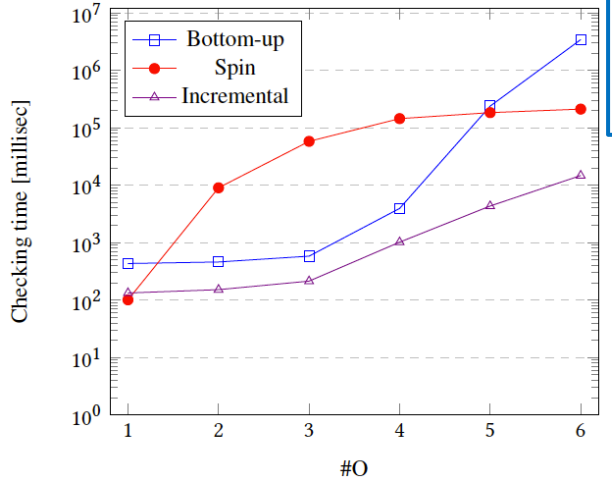
- Algorithms: Spin-based, bottom up, and incremental construction
- Dataset: randomly generated
- Input constraints: a set of 10 constraints, 2 from each class (8 verified, 2 unverified)
- Parameters: number of activities (#A), objects (#O), states per object (#S), transitions per activity (#T), and loops (#L)
- To measure the impact of
  - Workflow size (#A): sequential workflows with #O=3, #S=10, and #T=3 are considered
  - last four parameters: workflows consisting of all fragments with #A=10, #O=3, #S=10, #T=3, and #L=1 is considered (each time one of the #O, #S, #T, or #L is changed)
- Each experiment is performed 9 times, each time with a different change operation

Experiments Results

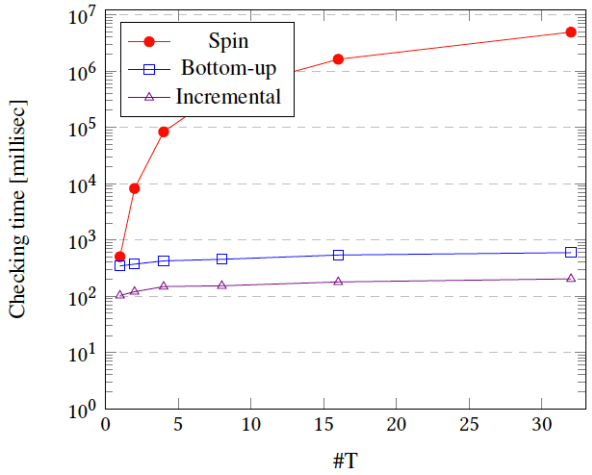
logarithmic

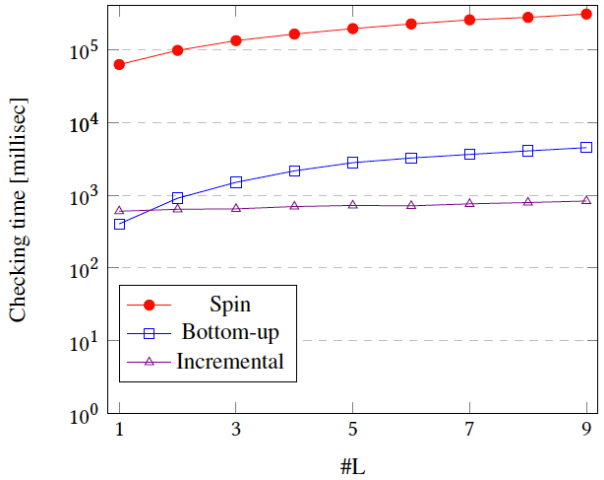




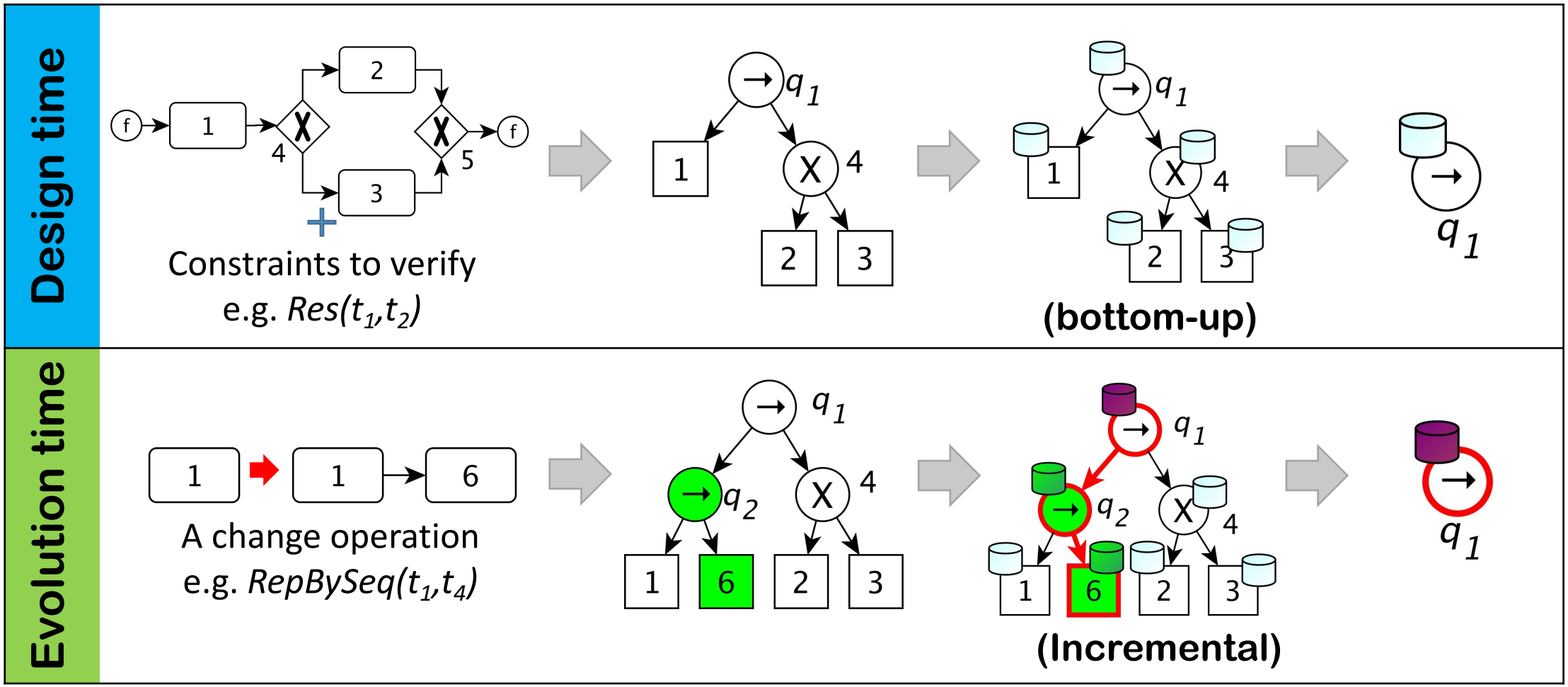


Bottom up: exponential grow rate in terms of the number of objects





Conclusion


Design time
Evolution time
Constraints to verify
e.g. Res(t1,t2)
A change operation
e.g. RepBySeq(t1,t4)
(bottom-up)
(Incremental)
Input Models
Process Tree
Process Tree with Auxiliary Data
Verification

Future Work

Extend the approach to support the same set of constraints regarding the object states (instead of activity nodes)

Support other types of constraints like Soundness (we need to change the structure of our auxiliary data)

Support schemas that do not have the liveness property (where executions might enter infinite loops)

Suggest a set of change operations on a workflow to make the specified constraints satisfiable.

Use the presented technique to solve model counting problems

THANK YOU!

Questions?!

I’m on Skype! (Hopefully ☺)