Lightweight query-based analysis of workflow process dependencies

W. Dai, D. Covvey, P. Alencar, and D. Cowan

The Journal of Systems and Software (2009)

2009. 9. 2

Hyunsik Cho

KAIST SE LAB

KAIST Software Engineering Laboratory

Contents

Introduction (1/4)

- Changes in workflow processes are unavoidable
  - Modification of regulations or laws leads to changes
  - Modified activities or processes impact on other activities or processes

→ Need to identify the entities affected by proposed change
  - Assess the effects of a change on workflow process to reduce the impact

Introduction (2/4)

- Previous works have limitations
  - Adam's work[1]
    - Construct Petri-Net based framework for modeling, simulation, and analysis of workflows
    - Focus on only routing dependency
      - Lead to an incomplete understanding of dependencies
      - Not consider dependency among processes
  - Kim's work[2]
    - Design workflow management system for process execution
      - Not focus on change impact analysis
    - Introduce a dependency analysis framework consisting of four separate dependency

[1] N.R. Adam , V. Atluri , W. Huang, Modeling and Analysis of Workflows Using Petri Nets, Journal of Intelligent Information Systems, 1998
[2] K. Kim, Workflow Dependency Analysis and Its Implications on Distributed Workflow Systems, International Conference on Advanced Information Networking and Applications, 2003

Introduction (3/4)

- Previous works have limitations (cont')
  - Ryder's work[1]
    - Analyze change impact for Object Oriented Programs by query mechanism
    - Not focus on workflow process dependency

[1] Barbara G. Ryder , Frank Tip, Change impact analysis for object-oriented programs, Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, 2001

Introduction (4/4)

- In this paper,
  - Analyze workflow process dependency
    - Focus on routing, data and role dependency model
      - For sufficient dependency analysis between entities
    - Using query mechanism based on logic programming
      - For more efficient and accurate than manual procedure

Background

Prolog

- Logic programming language
  - State properties which are true of system as *facts*
  - State query mechanism as *rules*
  - Infer the new facts from existing knowledge base
- Example

Activity_1 → Activity_2 → Activity_3

- Facts
  - RelatedTo(Act_1, Act_2)
  - RalatedTo(Act_2, Act_3)

- Rule
  - DependOn(A, B) :- RelatedTo(A, B)
  - DependOn(A, B) :- RelatedTo(A, C), DependOn(C, B)

- Query

?- DependOn(Act_1, Result)

=> RelatedTo(Act_1, Result)

=> RelatedTo(Act_1, Act_2) **-> True**

=> RelatedTo(Act_1, Act_3) **-> False**

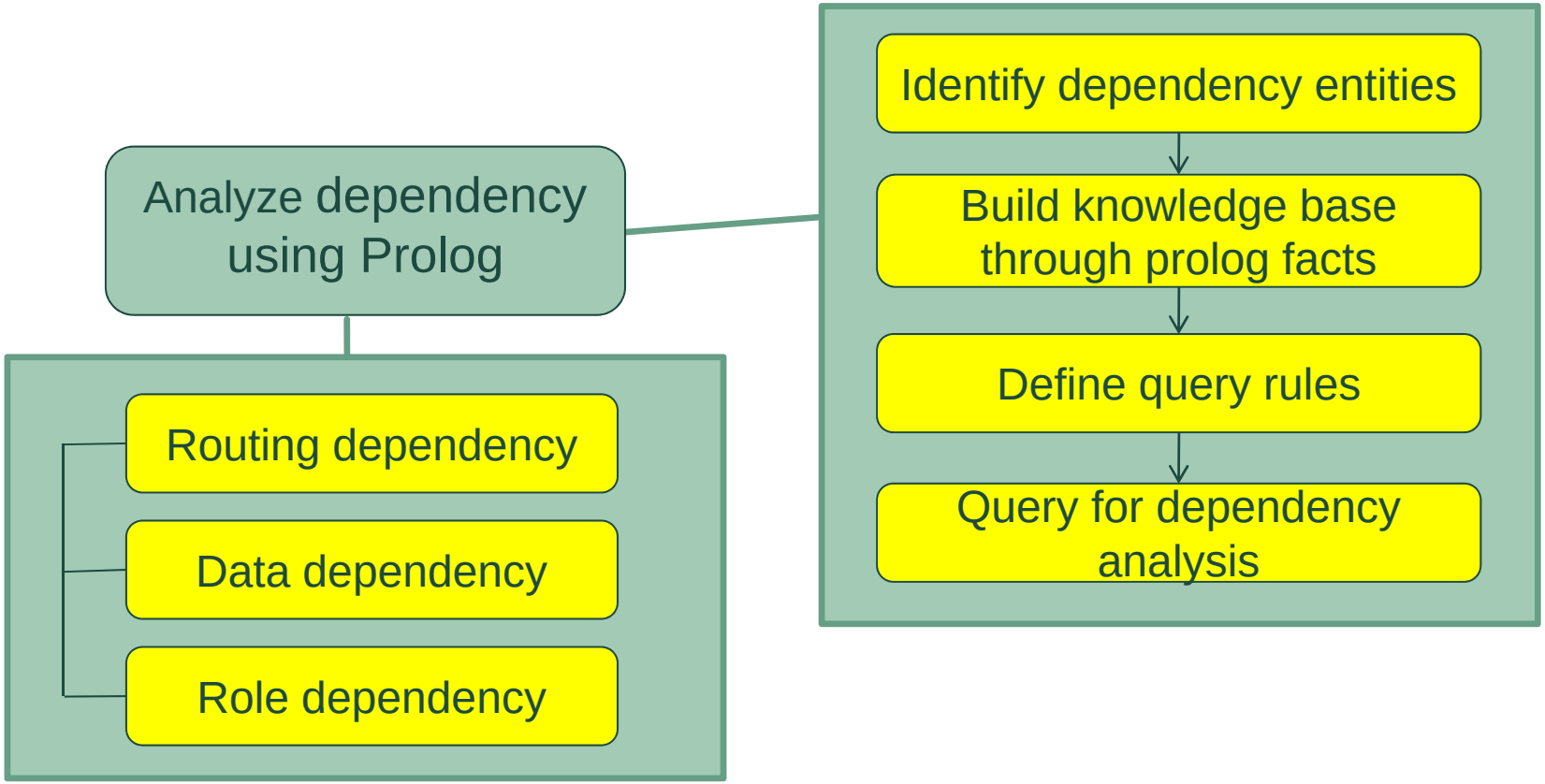
=> RelatedTo(Act_1, C), DependOn(C, Result) => …. =>

=> RelatedTo (Act_1, Act_2), RelatedTo(Act_2, Act_3) **-> True**

- **Output**

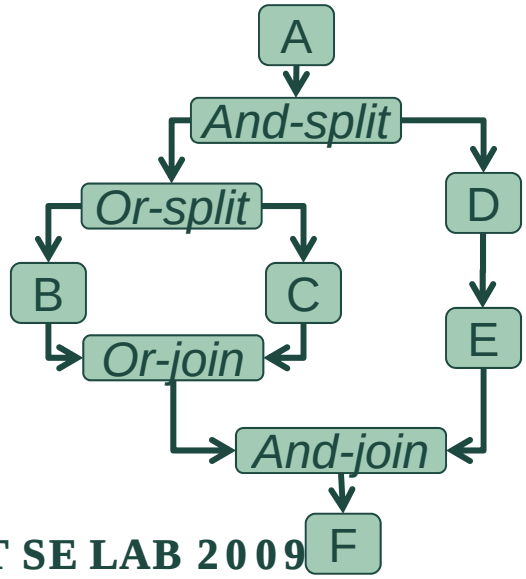
Result = Act_2, Act_3

Overall approach



Dependency model (1/3)

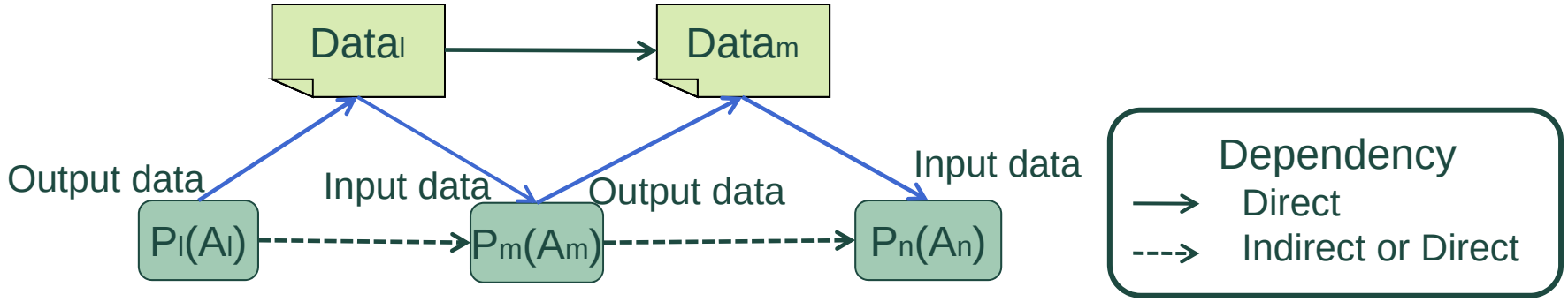
- Routing dependency
  - Define the execution order of activities in a process
    - Order of activities represented by a directed edge corresponds to routing dependency relationship
  - Use 5 routing types
    - Sequential, And-split, And-join, Or-split, Or-join
  - Example of process representation



•Routing dependency
- (A,B), (A,D), (D,E), etc.

Dependency model (2/3)

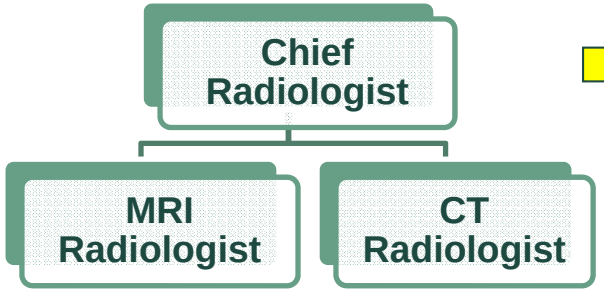
- Data dependency
  - Define the flow of data associated with each activity
  - Example



  - Data$_l$ and Data$_m$ are dependent directly
  - A$_l$ and A$_m$ may not be neighboring

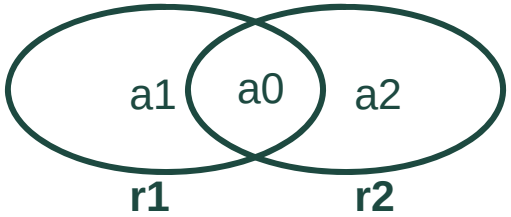
Dependency model (3/3)

- Role replacement dependency
  - Define the changeable roles corresponding to a role associated with an activity
  - Viewed as hierarchical structure
    - Higher role can execute activities executed by lower role
      - Example



Chief radiologist can execute activities which executed by CT radiologist

    - Replacement relationship is activity-based



- r1 is not replaceable to r2
- r1 is only replaceable to r2 executes a0

Logical based dependency representation and query(1/7)

❖ Process entities

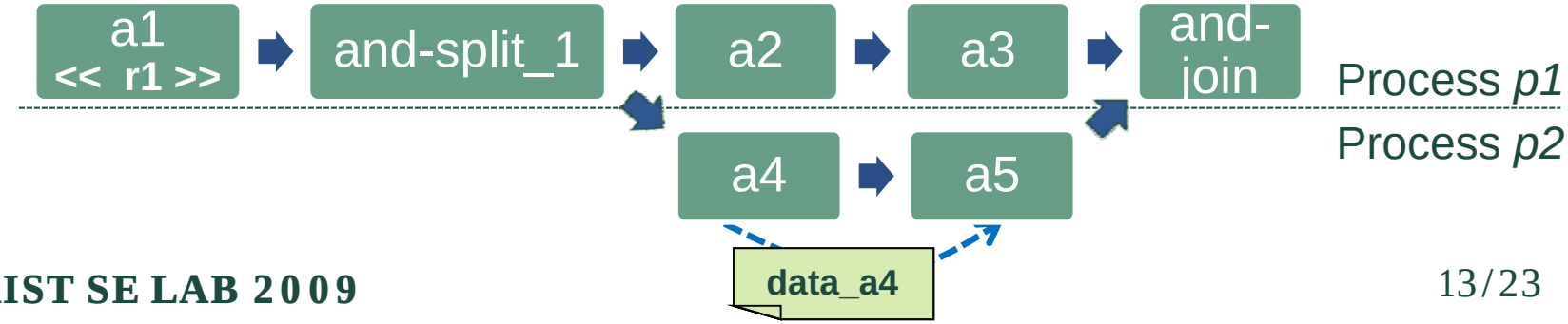
Entity	Definition	Single entity representation
P	Set of processes in analysis domain	p
A	Set of activities in process *p*	a
C	Set of basic control routing in process *p*	c (sequential, or-split, or-join, and-split, and-join)
D	Set of output data generated from activity *a* in process *p*	$(d_1^a, d_2^a, \ldots, d_k^a)$
R	Set of roles assigned to activity *a* in process *p*	r_a

Logical based dependency representation and query(2/7)

- Building knowledge base
  - Define basic facts of process

Fact	Argument types	Example
process	*P*	process(*p1*), process(*p2*), …
activity	*A, P*	activity(*a1*, *p1*), activity(*a4*, *p2*), …
routing	*structure(C,P)*	routing(and-*split*(*and-split_1*, *P1*)), …
data	*D, A, P*	data(*data_a4, a4, p2*), …
role	*R, A, P*	role(*r1, a1, p1*), …

  - Example



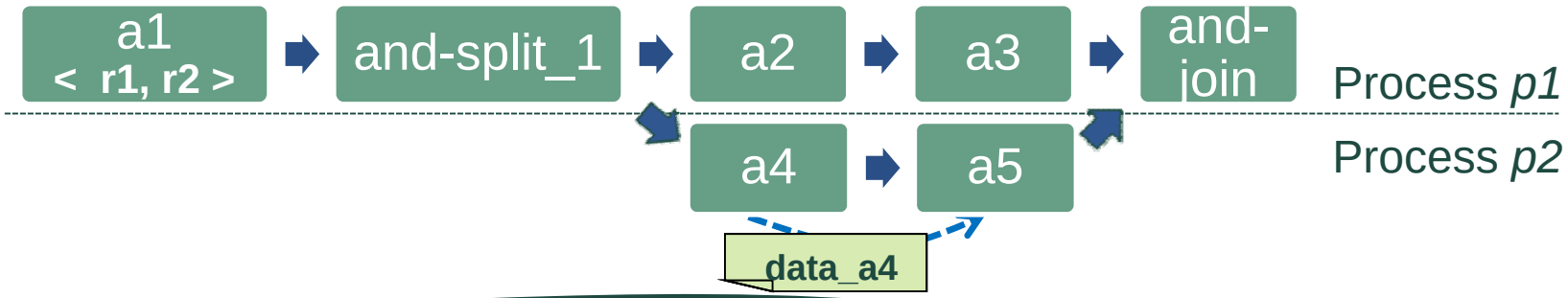
Logical based dependency representation and query(3/7)

❖ Building knowledge base (cont')

- Define dependency facts

Fact	Argument types	Example
routingDep	***A_1, P_1, [routing], A_2, P_2***	**routingDep(a1, p1, [and-split(and-split_1, p1)], a4, p2), ...**
dataDep	*A_1, P_1, structure(D, A_2, P_2)*	dataDep(a5, p2, input(data_a4, a4, p2)), …
roleDep	*R_1, A, P, R_2*	roleDep(r1, a1, p1, r2), ..

- Example



• Inter-dependency among processes is supported
  - All facts include the process information

Logical based dependency representation and query(4/7)

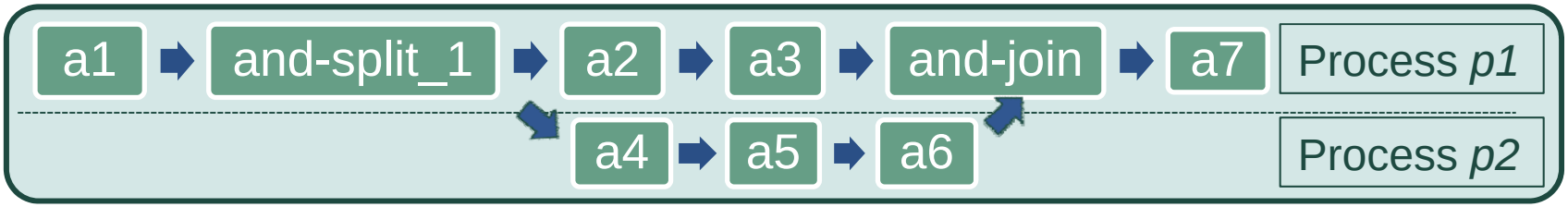
- Query rules
  - Define at different level
    - Activity level
      - Find activities which depend on a activity to be changed
    - Process level
      - Find activities which depend on any activity in a process to be changed
  - Define in different direction
    - Forward query
      - Find succeeding entities to be impacted based on given activity or process
    - Backward query
      - Find preceding entities to impact on given base activity or process

Logical based dependency representation and query(5/7)

❖ Query rules for routing dependency

Query rule	Result	Query rule
postActivityRouting(A, P, Results)	(A, [routings], P)	postProcessRouting(P, Results)
preActivityRouting(A, P, Results)	(A, [routings], P)	preProcessRouting(P, Results)
activityReachPaths(A, P, B, Q)	(A, P) path	

- Example



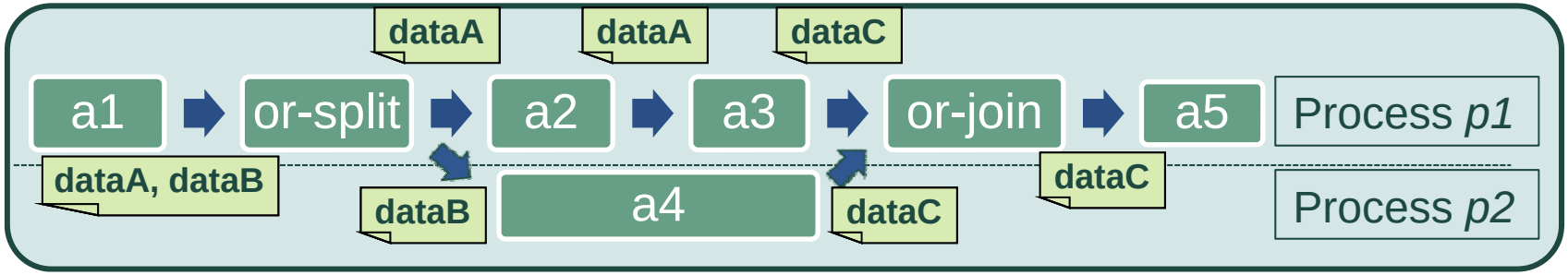
- postActivityRouting(a1, p1, Results)
  - Results = activity(a2, [and-split(and-split_1, p1)], p1),
    activity(a4, [and-split(and-split_1, p1)], p2)
- activityReachPath(a4, p2, a7, p1)
  - Results = activity(a5, p2), activity(a6, p2)

Logical based dependency representation and query(6/7)

Query rules for data dependency

Query rule	Result	Query rule
postActivityData(A, P, [D], Results)	(A, P)	postProcessData(P, Results)
preActivityData(A, P, [D], Results)	(A, P)	preProcessData(P, Results)

- Example



- postActivityData(a1, p1, Results)
  - Results = activity(a2, p1), activity(a4, p2)
- preprocessData(a1, p1, dataA, Results)
  - Results = activity(a2, p1)

Logical based dependency representation and query(7/7)

Query rules for role dependency

Query rule	Result
roleReplaceable(R1, R2, A, P)	Y or N
roleActivityReplace(R, Results)	(R, A, P)
roleReplace(R, A, P, Results)	(R)

- Analysis example

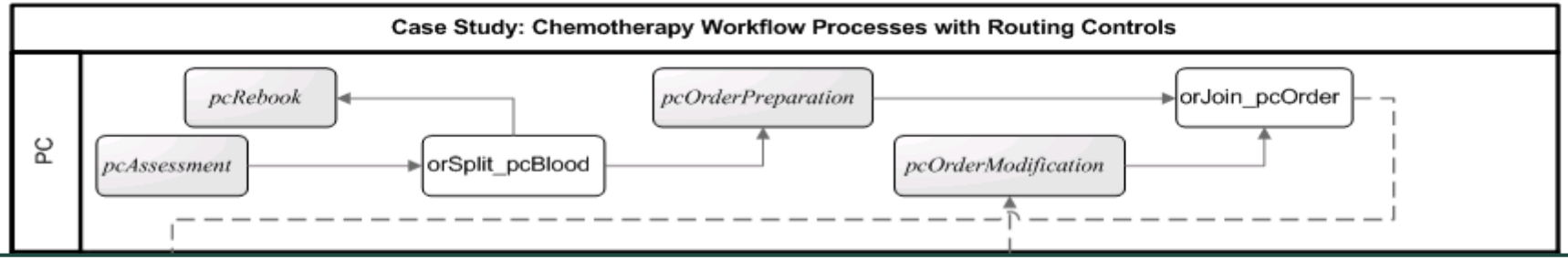
- roleReplaceable(rA, rB, A1, p1) => Results = N
- roleActivityReplace(rA, Results) => Results = role(rB, a2, p1)
- roleReplace(R, A, P, Results) => Results = rB

Case study (1/3)

- Apply this approach to real case
  - Use healthcare workflow process of Grand River Region cancer Center
    - 4 processes, 16 activities and 14 data and 5 roles
  - Use prolog tool XSB

Case study (2/3)

1. Build knowledge base from workflow process diagram and associated data and role table by identifying logic facts



Activity(Process)	Activity Name(ID)	Input/Output Data	Role
Physician a patient base blood test r (PC).			hysician
Rebook pat another ass (PC).			ooking taff
Doctor prep for followin cycle (PC).			hysician

```
% Routing Dependency
routingDep(pcAssessment, pc, [orsplit(orSplit_pcBlood, pc)],
pcRebook, pc).
routingDep(pcAssessment, pc, [orsplit(orSplit_pcBlood, pc)],
pcOrderPreparation, pc).
… …
% Data Dependency
… …
dataDep(pchConfirmation, pch, input(medicine, ctPreparation,
ct)).
```

Case study (3/3)

2. Query for dependency analysis

<postActivityRouting query>

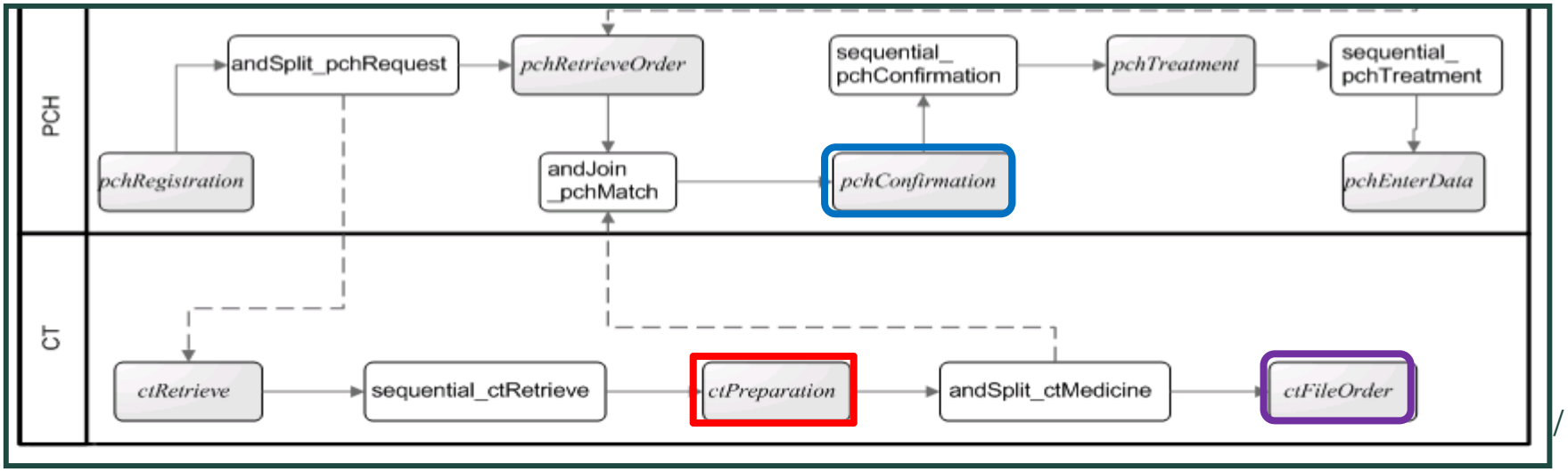
```
C:\WINDOWS\System32\cmd.exe - xsb
| ?- [chemotherapyProcesses].
[chemotherapyProcesses loaded]

yes
| ?- postActivityRouting(ctPreparation, ct, Results).

Results = [activity(pchConfirmation,pch),[andsplit(andSplit_ctMedicine,ct),andjoin(andJoin_pchMatch,p
ch)]),activity(ctFileOrder,ct,[andsplit(andSplit_ctMedicine,ct)])]
```

Return the expected results

<workflow process>



Conclusion

- Analyze the dependency relationship between changed parts and potentially affected parts
  - Analyze 3 types of dependency
    - Routing, data and role
  - Analyze at different levels
    - Activity and process
  - Analyze in different directions
    - Forward and backward
- Analyze dependency using logical programming
  - by query from knowledge base

Discussion

- Contributions
  - Propose solid foundation for other dependency relationship analysis
  - Propose effective query based dependency analysis

- Limitations
  - Identify the facts manually
  - Not support dependency analysis of workflow process in distributed environment

Thank You .

KAIST SE LAB
KAIST Software Engineering Laboratory