Process Mining

Part IV – Clustering-based Process Mining

Discovery of hierarchical process models
Discovery of process taxonomies
Outlier detection

Outline

- **Part I – Introduction to Process Mining**
  - Context, motivation and goal
  - General characteristics of the analyzed processes and logs
  - Classification of Process Mining approaches
- **Part II – Workflow discovery**
  - Induction of basic Control Flow graphs
  - Other techniques (α-algorithm, Heuristic Miner, Fuzzy mining)
- **Part IV – Beyond control-flow mining**
  - Organizational mining
  - Social net discovery
  - Extension algorithms
- **Part III – Evaluation and validation of discovered models**
  - Conformance Check
  - Log-based property verification
- **Part V – Clustering-based Process Mining**
  - Discovery of hierarchical process models
  - Discovery of process taxonomies
  - Outlier detection

Limitations of classical wf-discovery approaches

- Model expressiveness is limited, as only local relationships are considered between tasks
  - real-life processes may follow complex behavioral rules, which cannot easily expressed through precedences and local constraints
  - e.g., there is no actual execution containing both fidelity discount and register new client, even if Order Management schema admits them
- The discovery of variants of a given process is not addressed
- In both cases, the resulting process model can be too loose:
  - several modeled executions will never occur in any actual enactment

Quality of a mined schema

The quality of a schema *W* can be measured w.r.t. the log *L* it was extracted from

- *Soundness*: % of traces of *W* that occur in *L*
- *Completeness*: % of traces in *L* that comply with *W*

High soundness is difficult to be achieved...

How to mine accurate models?

Alternatives

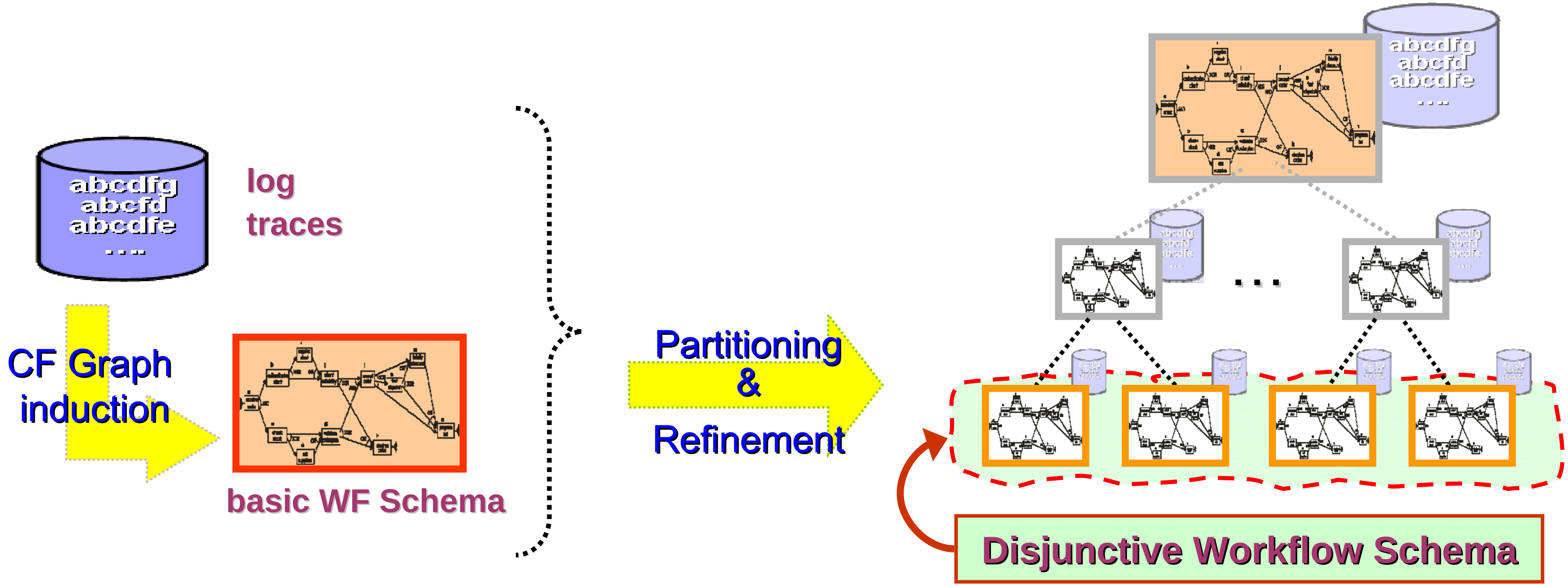
~~Use more expressive languages / meta-models~~

- ~~e.g., control flow graphs could be enriched with additional "global" constraints, relating nodes that are not adjacent to each other~~
- ~~but, explicitly handling such constraints may lead to knotty models and makes harder the process mining task~~

Mine different schemas (usage scenarios)

- Complex behavioral rules can be caught indirectly, by recognizing different unexpected and frequent behavioral patterns
  - unexpected w.r.t. a given control flow graph, but frequent in the log
  - such patterns evidence the existence of constraints (or usage patterns) that are not properly modeled by the graph
- Use a set of workflow schemas
  - more expressive, and accurate, than a single schema
  - but still intuitive and easy to mine

The proposed approach



- Mine a basic schema S_0 modeling all the log traces and put it in W^U
- Iteratively refine a schema S_K (e.g., the least sound) in W^U:
  - cluster its associated traces according to their mutual similarity w.r.t. “unexpected” behavioral patterns (see later) discovered in the log
  - produce a new schema for each cluster of traces

… till the soundness of W^U is not satisfactory and its size is less than M

The Process Mining Algorithm in detail

INPUT: log ***L***, two natural numbers ***M*** and ***k***, a soundness threshold γ

OUTPUT: a hierarchy ***H*** of workflow schemas

1. ***W₀***=*mineWFschema*(*L*) *// a preliminary schema is built for L, essentially*
   // modeling precedences and local constraints
2. set W_0 as the root of ***H*** and assign all the traces in *L* to it
3. **WHILE** *soundness*(***H***,***L***)< γ **AND** ***H*** contains less than ***M*** nodes **AND** there are leaf schemas that have not been examined yet
   i. Let ***W**** be the least sound leaf schema not considered yet
   ii. Partition the traces associated with ***W**** into at most ***k*** clusters
   iii. For each cluster obtained, mine a workflow schema (using again method *mineWFschema*) and add it to ***H*** as a child of ***W****
4. **END WHILE**
5. **RETURN** ***H***

- **The algorithm converges in at most *M* steps**
- **After each step the soundness of *H* increases**

Top-down node refinement

- Given a node *N*, with schema *S* and trace set *T*
  - A set of nodes is obtained which corresponds to a partition of *T* and to a set of schemata more specific than { *S* }
- Clustering (partitioning) of *T*
  1. Find a set of features which capture different patterns of behavior exhibited by traces in *T*
     - unexpected w.r.t the schema *S*
  2. Select an optimal subset of features (greedily)
  3. Project the traces in *T* in the feature space
  4. Apply a distance-based clustering algorithm (e.g., *k-means*) to the traces of *T*
  5. Mine a refined schema for each cluster

Properties of the algorithm and issues

- Properties of the algorithm:
  - The algorithm converges in almost *M* steps of the main loop
  - After each step (refine the selected schema) the soundness of the disjunctive schema *W** cannot decrease (and usually gets higher)

- Issues related to the features:
  - What a kind of features?
  - How to select them?

Features: discriminating rules

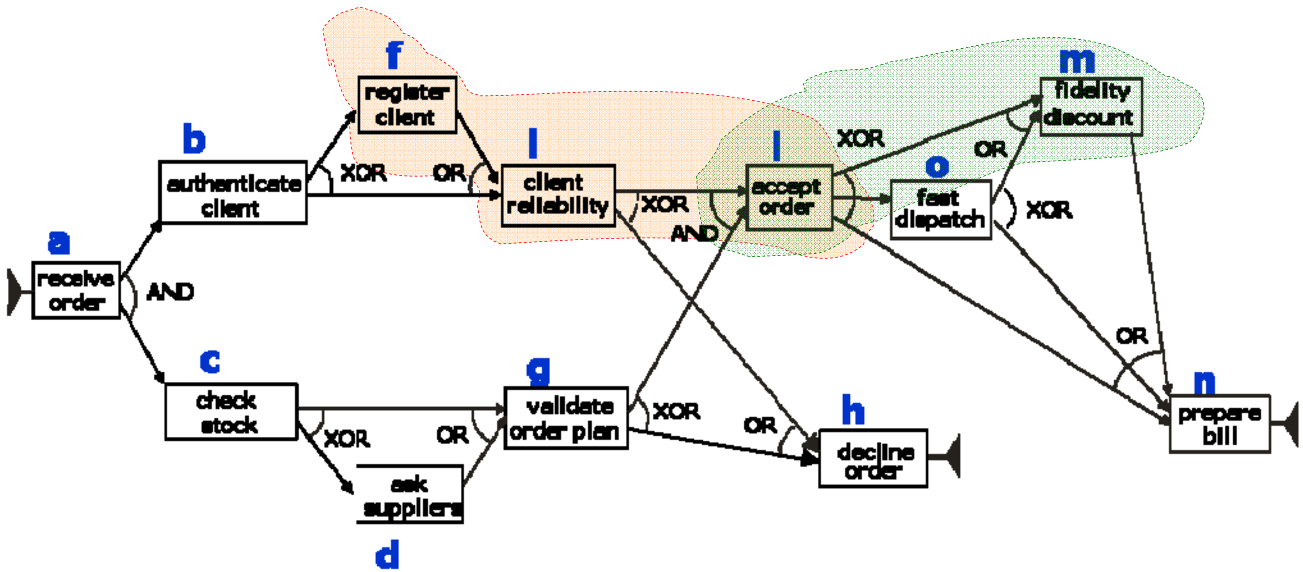
- *A discriminating rule* is an expression $\phi : [a_1 \ldots a_h] \nrightarrow a$, s. t.:
  - $[a_1 \ldots a_h]$ and $[a_h a]$ are both "highly" frequent in *L*
  - but $[a_1 \ldots a_h\ a]$ is "lowly" frequent in *L*

  … according to some given frequency thresholds

  - evidence for hidden constraints or unexpected patterns of behavior

- Example:

$f\ i\ l \nrightarrow m$



- In the log of *OrderManagement* both sequences `fil` and `lm` are frequent, but their combination `film` never occurs in the log
  - due to the global constraint disallowing `m` whenever `f` is executed

Mining Discriminant Rules

Input: A log $\mathcal{L}_P$, a schema $\mathcal{WS} = \langle A, E, a_0, A_F, \mathit{Fork}, \mathit{Join} \rangle$, thresholds σ and γ, natural number ℓ and $maxFeatures$.

Output: A set of minimal discriminant rules.

Method: Perform the following steps:

$L_2^\sigma := \{\mathtt{ab} \mid \mathtt{ab}$ is σ-frequent in $\mathcal{L}_P\}$;
$len := 3$; $\mathcal{F} := \emptyset$;
while $len \leq \ell$ and $L_{len}^\sigma \neq \emptyset$ **do** //*iterations on the length of the features*
  $Cand_{len} := \emptyset$; $F_{len} := \emptyset$;
  for each sequence $\mathtt{a_1 \ldots a_j} \in L_{len-1}^\sigma$ **do** //*construction of the candidates*
    for each $\mathtt{a_j a} \in L_2^\sigma$ **do**
      $Cand_{len} := Cand_{len} \cup \{\mathtt{a_i \ldots a_j a}\}$;
  $L_{len}^\sigma := \{s \mid s \in Cand_{len} \wedge s$ is σ-frequent in $\mathcal{L}_P\}$;
  $L_{len}^\gamma := \{s \mid s \in Cand_{len} \wedge s$ is γ-frequent in $\mathcal{L}_P\}$;
  for each sequence $\mathtt{a_1 \ldots a_j a} \in (Cand_{len} - L_{len}^\gamma)$ **do** //*update features*
    if $\nexists \mathtt{a_1 \ldots a_j b} \in (Cand_{len} - L_{len}^\gamma)$ such that $\mathtt{ab} \in L_2^\sigma$ and
      $\nexists [\mathtt{c_1 \ldots c_k}] \not\dashrightarrow_{\langle \sigma, \gamma \rangle} \mathtt{a}$ in $\mathcal{F}$, such that $tasks(\mathtt{c_1 \ldots c_k}) \subseteq tasks(\mathtt{a_1 \ldots a_j})$ **then**
    $F_{len} := F_{len} \cup \{[\mathtt{a_1 \ldots a_j}] \not\dashrightarrow_{\langle \sigma, \gamma \rangle} \mathtt{a}\}$;
  end for
  $\mathcal{F} := \mathcal{F} \cup F_{len}$; $len := len + 1$;
end while
return $mostDiscriminantFeatures(\mathcal{F}, maxFeatures)$;

Initialization: L_2^σ contains all the *σ-frequent* sequences of length 2

Generate all possible *σ-frequent* sequences whose length is *len*, based on *σ-frequent* sequences with length *len−1*, and store them in $Cand_{len}$

Scan the log to spot the sequences in $Cand_{len}$ that are *σ-frequent* and *γ-frequent* in Λ_P

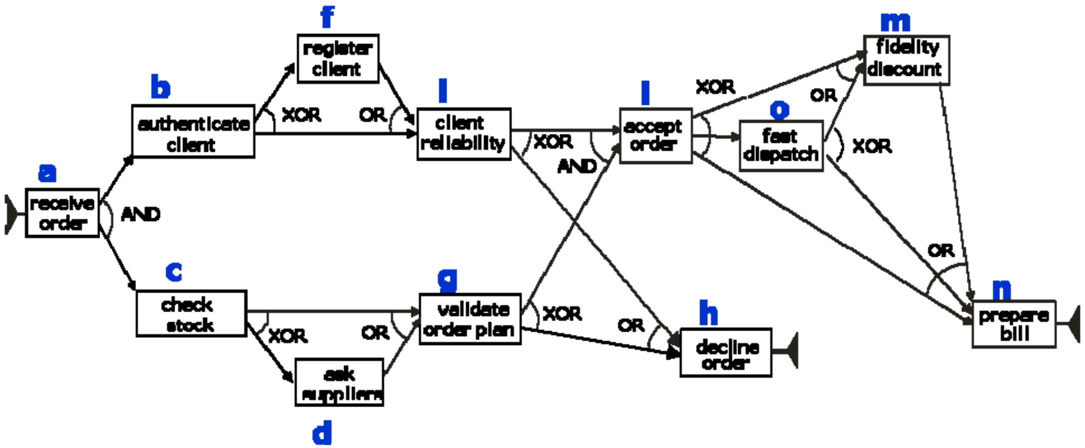
Identify the features consisting of *len* nodes

Insert the discovered minimal features into Φ

Select the *maxFeatures* most frequent in Φ, in order to reduce the dimensionality of feature space: the features with the lowest values of γ are chosen.

Selecting a good set of features

- *Minimal* discriminating rule
  - Introduced to prune redundant rules, e.g.: `abfil` $\not\rightarrow$ `m`



  - we defined a level-wise method for singling out all of them

- *Most discriminating features*
  - An optimality criterion for select a subset of features, which allow to split the traces "at best" (significant clusters)
  - We defined a greedy heuristics for finding an approximate solution

The approach in action: mined clusters

Log traces:

s_1: *acdbfgih*	s_5: *abicglmn*	s_9: *abficgln*	s_{13}: *abcidglmn*
s_2: *abficdgh*	s_6: *acbiglon*	s_{10}: *acgbfilon*	s_{14}: *acdbiglmn*
s_3: *acgbfih*	s_7: *acbgilomn*	s_{11}: *abcfdigln*	s_{15}: *abcdgilmn*
s_4: *abcgiln*	s_8: *abcfgilon*	s_{12}: *acdbfigln*	s_{16}: *acbidgln*

Discovered Features:

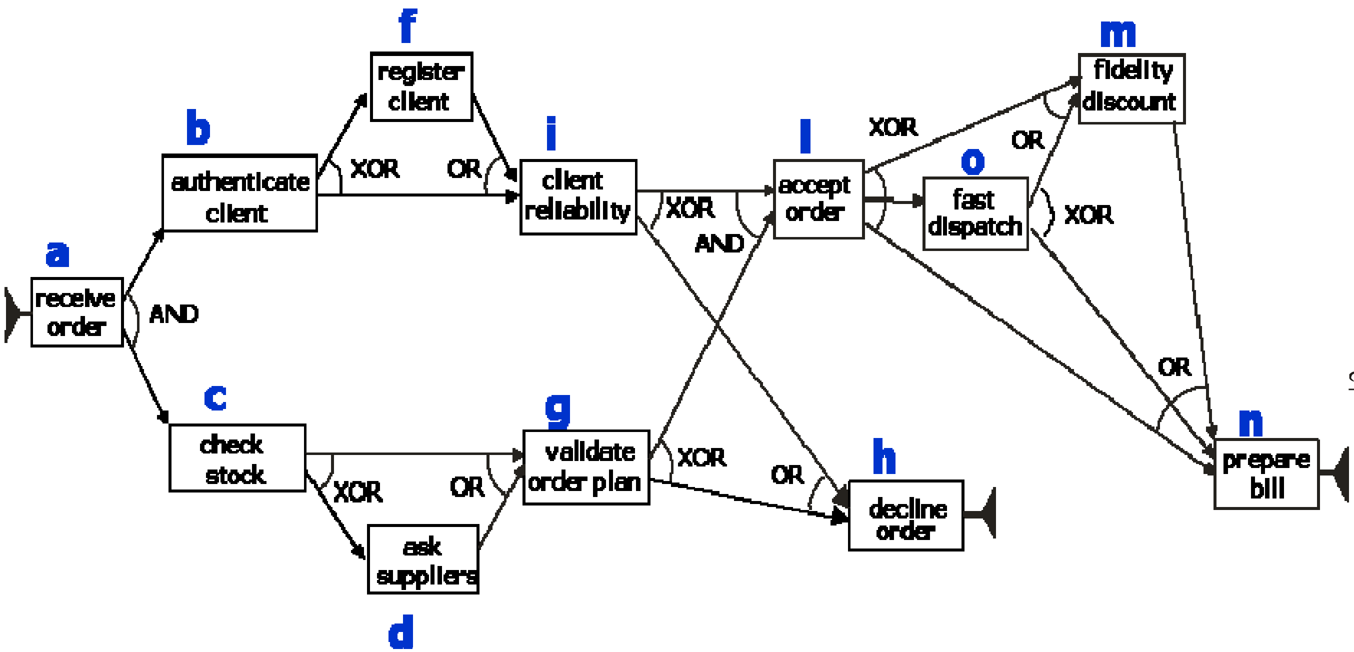
ϕ_1 : [*f i l*] $\not\rightarrow$ *m*

Fidelity discounts are never applied on new (just registered) clients

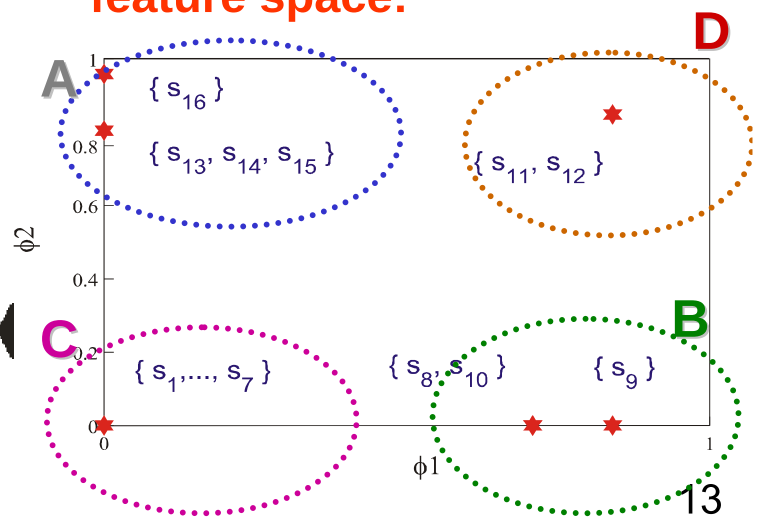
ϕ_2: [*d g l*] $\not\rightarrow$ *o*

If external supplies have been checked, no fast dispatch occurs

Basic (first-level) schema induced:

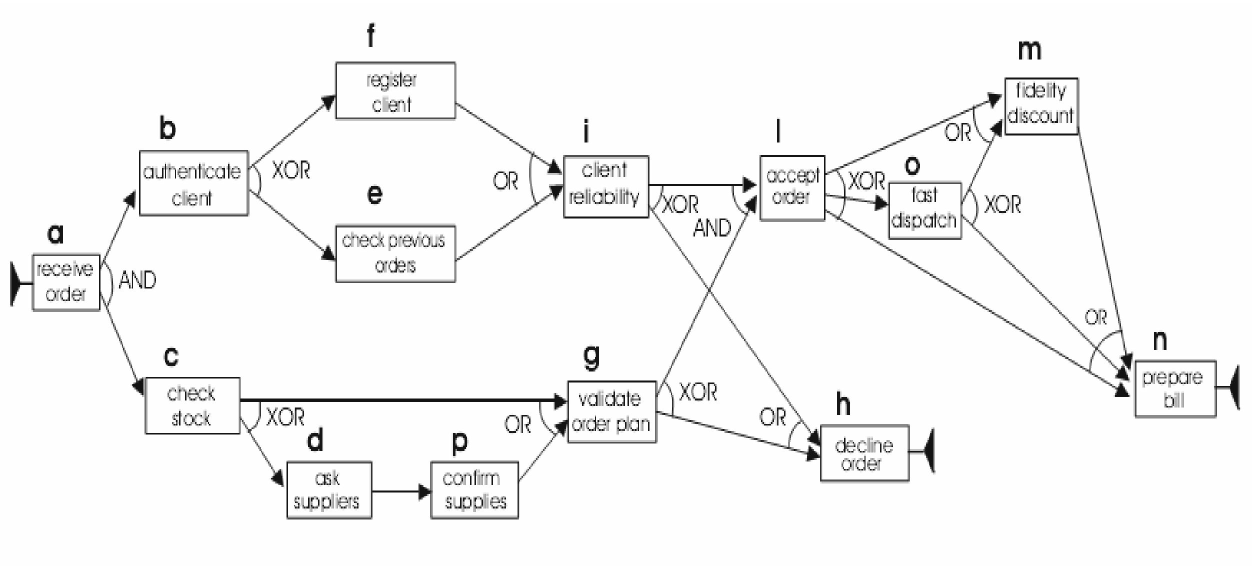


Clusters of traces in the feature space:



The approach in action:

The first schema induced



- W_0 coincides with the original schema
  - it does not model the additional constraints
- W_0 hence admits “extraneous” traces
  - e.g., **`acgbfilmn`**

- In order to get higher soundness, W_0 we search for clusters of traces that correspond to different usage scenarios
- To this aim a set of discriminating features is extracted:
  - ϕ_1 : [***f i l***] $\not\rightarrow$ ***m***

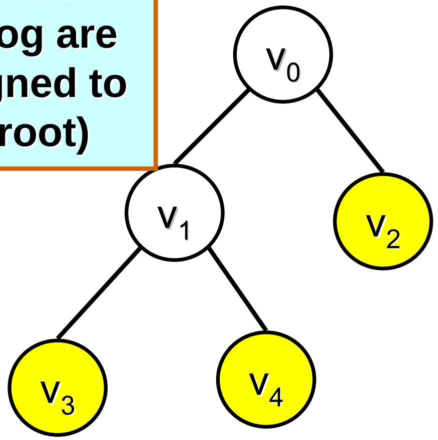
    Fidelity discounts are never applied on new (just registered) clients
  - ϕ_2: [***d g l***] $\not\rightarrow$ ***o***

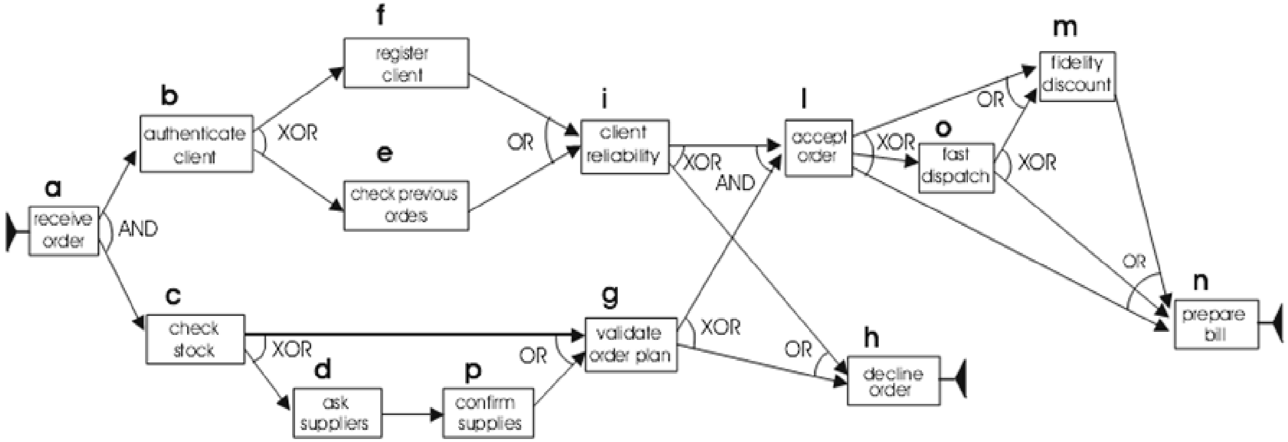
    If external supplies have been checked, no fast dispatch occurs

The approach in action:

The discovered hierarchy of schemas

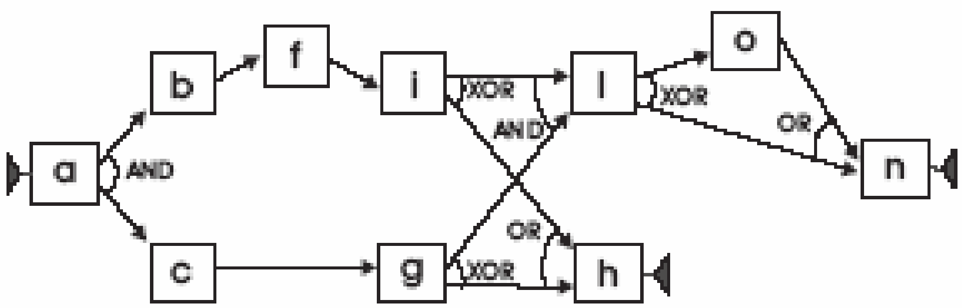
all traces in the log are assigned to v_0 (root)



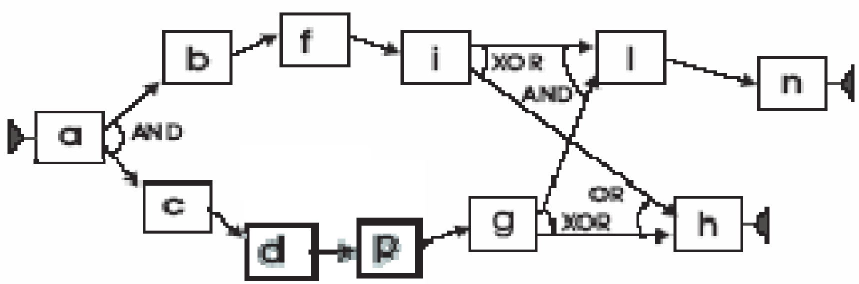


Workflow schema W_0 for node v_0

W_0 must be refined because its soundness is not high enough



Workflow schema W_3 for node v_3



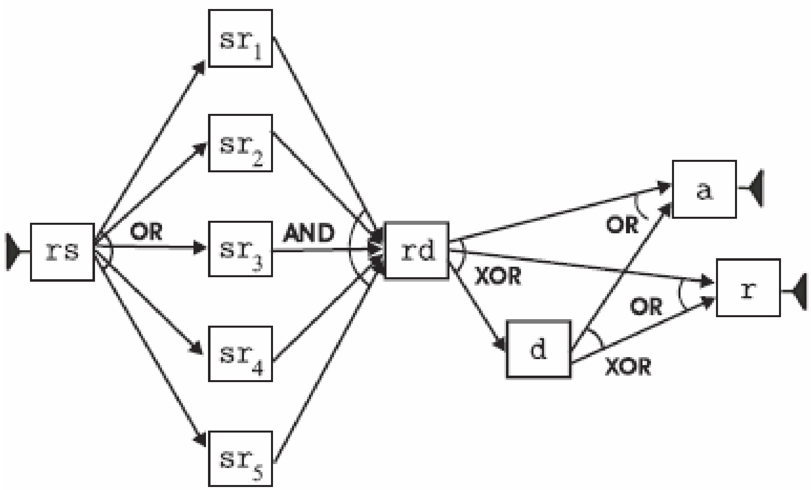
Workflow schema W_4 for node v_4

the leaf schemas (the only ones shown here) constitute, as a whole, a maximally sound and complete disjunctive scheme

Example 2

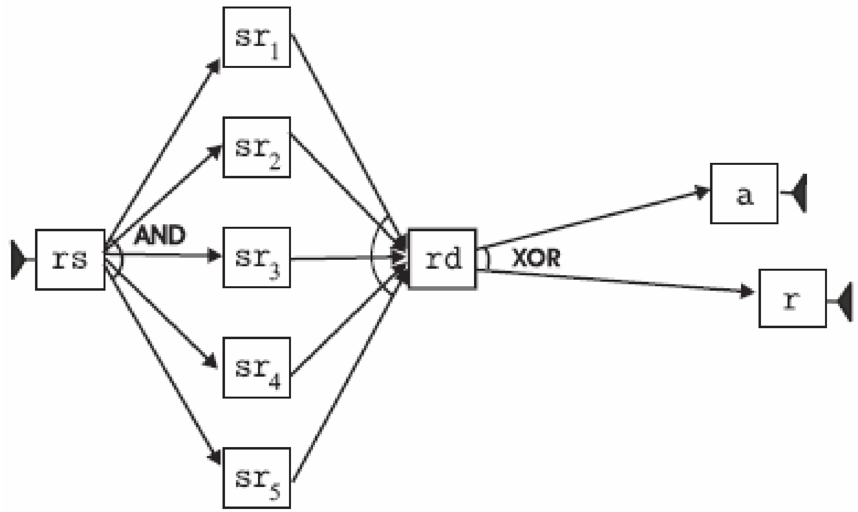
- process *ReviewPaper:*
  - (**rs**) receiving the submission
  - ($\mathbf{sr}_i$) ($1 \leq i \leq 5$) sending the paper to the reviewers,
  - (**rd**) receiving the revisions and take a decision,
  - (**d**) discussing on the paper in the case revisions are not uniform,
  - (**a**) accepting the paper, and
  - (**r**) rejecting the paper.
- Constraints:
  - if the paper is authored by a program committee member, it has to be reviewed by 5 reviewers and it is immediately rejected in the case some reviewer does not want it to be accepted for publication.
  - Otherwise, only 3 reviewers are assigned to the paper.

A single workflow model for the process:

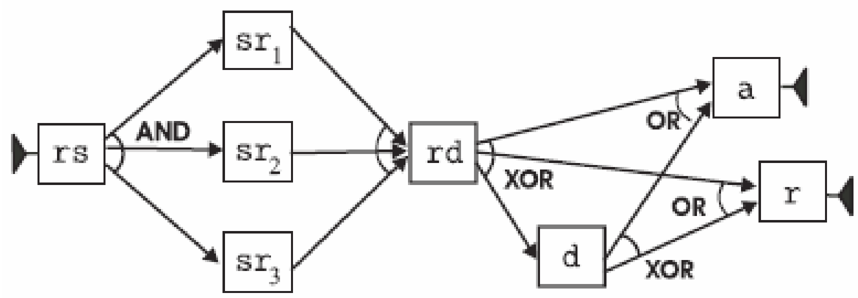


Clustering

…refined workflow schemas

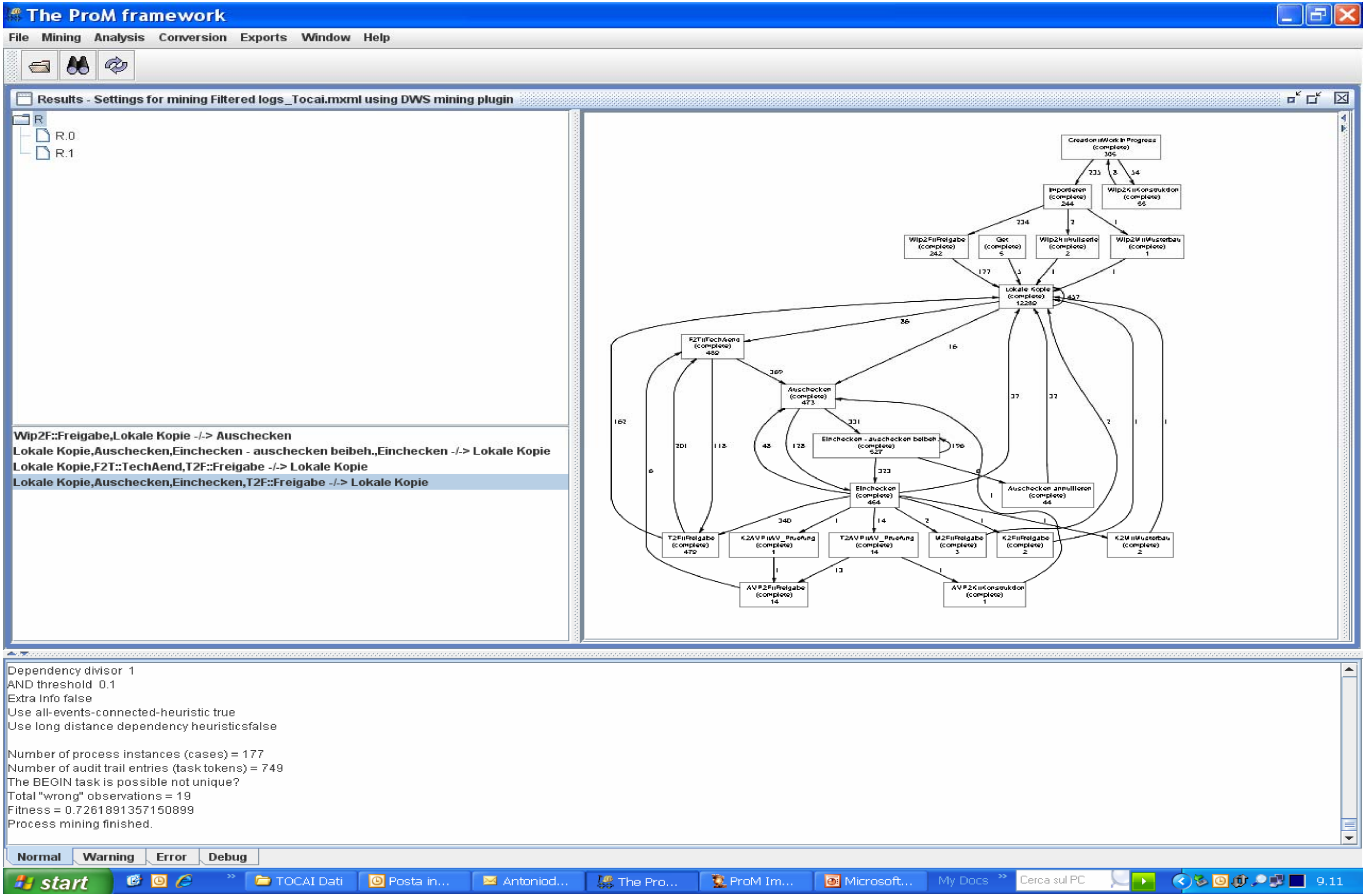


This schema is a *1-sound* model for the handling the revision of a paper written by a program committee member



This schema is a *1-sound* model for handling the revision of all the other papers

Plugin DWS


The ProM framework
File Mining Analysis Conversion Exports Window Help
Results - Settings for mining Filtered logs_Tocai.mxml using DWS mining plugin
R
R.0
R.1
Wip2F::Freigabe,Lokale Kopie -/-> Auschecken
Lokale Kopie,Auschecken,Einchecken - auschecken beibeh.,Einchecken -/-> Lokale Kopie
Lokale Kopie,F2T::TechAend,T2F::Freigabe -/-> Lokale Kopie
Lokale Kopie,Auschecken,Einchecken,T2F::Freigabe -/-> Lokale Kopie
Dependency divisor 1
AND threshold 0.1
Extra Info false
Use all-events-connected-heuristic true
Use long distance dependency heuristicsfalse
Number of process instances (cases) = 177
Number of audit trail entries (task tokens) = 749
The BEGIN task is possible not unique?
Total "wrong" observations = 19
Fitness = 0.7261891357150899
Process mining finished.
Normal
Warning
Error
Debug
start
TOCAI Dati
Posta in...
Antoniod...
The Pro...
ProM Im...
Microsoft...
My Docs
Cerca sul PC
9.11

Outline

- **Part I – Introduction to Process Mining**
  - Context, motivation and goal
  - General characteristics of the analyzed processes and logs
  - Classification of Process Mining approaches
- **Part II – Workflow discovery**
  - Induction of basic Control Flow graphs
  - Other techniques (α-algorithm, Heuristic Miner, Fuzzy mining)
- **Part III – Beyond control-flow mining**
  - Organizational mining
  - Social net discovery
  - Extension of workflow models
- **Part IV – Evaluation and validation of discovered workflow models**
  - Conformance Check
  - Log-based property verification
- **Part V – Clustering-based Process Mining**
  - Discovery of hierarchical process models
  - Discovery of process taxonomies
  - Outlier detection

Motivation: mining complex processes

- *Problem*: real processes may involve lots of activities, and complex behavioral rules for combining them
  - the discovered model may fail in representing the process with enough accuracy
  - … and may be too complex for business users who want to monitor and analyze process executions at an appropriate abstraction level

Desiderata

Execution Classification

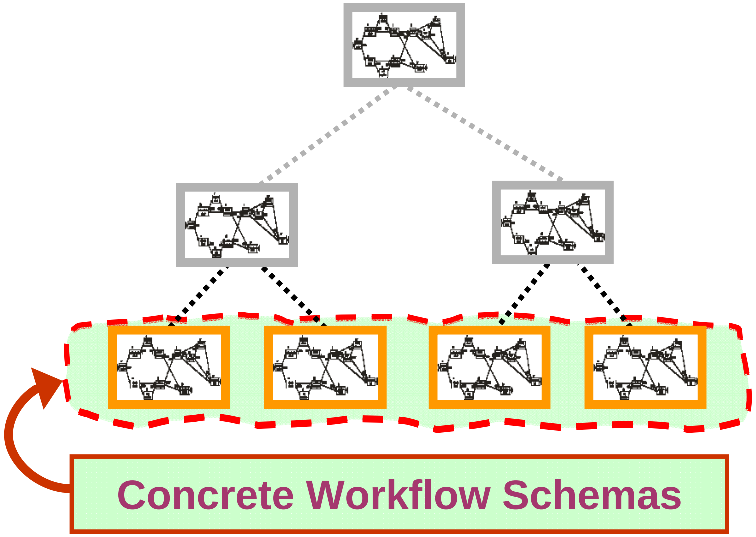
This allows to gain in accuracy,modularity, and understandability, w.r.t. a single workflow schema mixing all executions

Abstraction

BPA platforms (e.g, iBOM by HP) allow to manually define abstract views over a workflow, by mainly aggregating groups of activities

Taxonomical process models

- An expressive and easy to understand process model, consisting of a taxonomy of workflow schemas



- The tree describes the process behavior at different level of details
- At the highest level of detail (leaves of the tree), the schemas could be used to support the design of concrete workflow models
- At lower levels, the schemas are abstract views over heterogeneous behaviors, which could support analysis and monitoring tasks

- A two-phase discovery approach:
  - First, mine a tree of workflow schemas, by using a hierarchical, top-down, clustering algorithm
  - Then, restructure the mined model at several levels of abstraction, in a bottom-up way (i.e., from the leaves to the root)

Generalization of workflow schemas

- Given two workflow schemas *W* and *W'* (with activity set *A* and *A'*, resp.), it is said that *W generalizes W'*, denoted by $W' \prec W$, if :
  1. for any activity *x* in *A* either *A*' contains *x* or there exists at least one activity *y* in *A'* such that *x* "*abstracts*" *y*, and
  2. there is no activity in *A'* that "*abstracts*" *x*
- Schema taxonomies are defined according to this notion

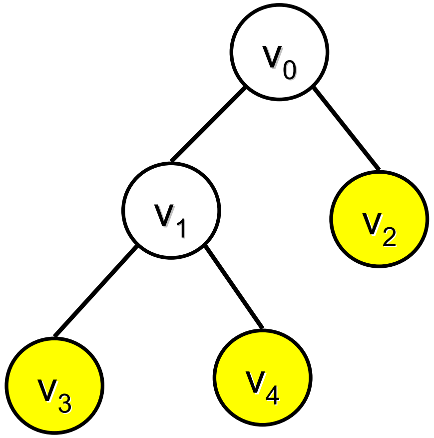
  A schema hierarchy *H* for *P* is a *schema taxonomy* if *Schema*(v) $\prec$ *Schema*(v') for any *v, v*' such that *v*' is a child of *v*

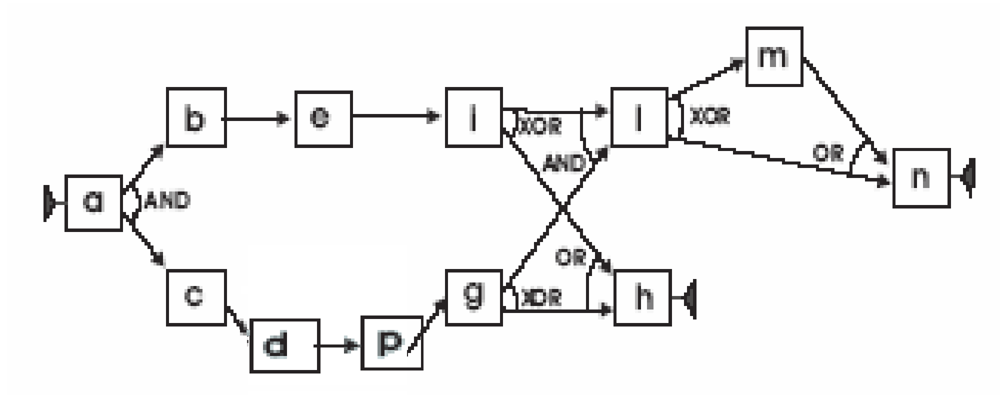
Abstraction relationships among activities

- Basic relationships: abstraction dictionary *D=<Isa,PartOf>*
  - (*b, a*) in *Isa* means that *b* is a refinement of *a*
  - (*b, a*) in *PartOf* means that *b* is a component of *a*
- Derived relationships
  - *a* ***implies*** *a'* w.r.t. *D*, denoted by $a \rightarrow^{D} a'$, if
    - (*a', a*) in *D.Isa*, or
    - (*a', a*) in *D.PartOf*, or
    - (recursively) there exists an activity *x* such that $a \rightarrow^{D} x$ and $x \rightarrow^{D} a$
  - The set of activities implied by *a* w.r.t. *D* is referred to as $impl^{D}(a)$
- Complex activities
  - An activity *a* is *complex* if $impl^{D}(a)$ is not empty
  - It is a higher level concept defined over the (basic) activities that actually occur in the executions

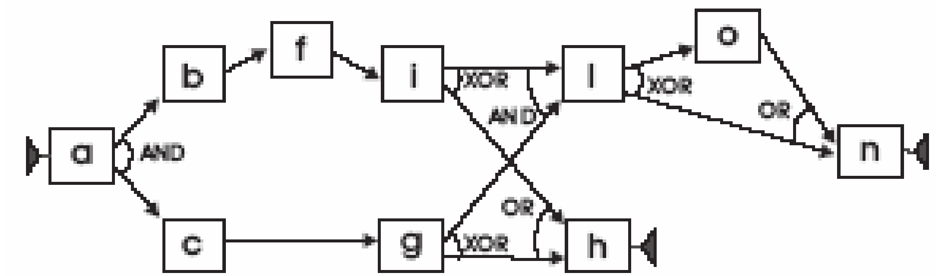
Example: The mined schema hierarchy

- The hierarchy of workflow schemas extracted so before

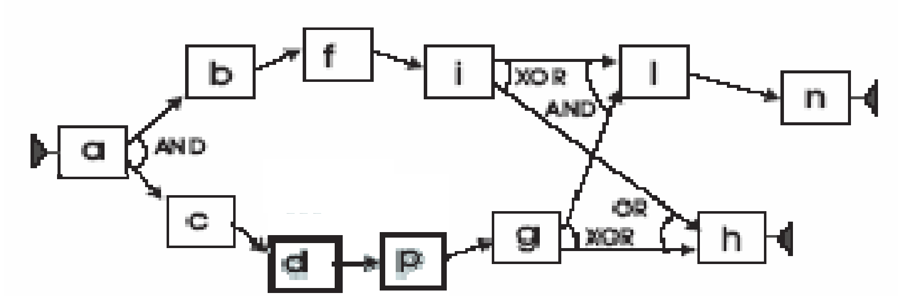




Workflow schema W_2 for node v_2



Workflow schema W_3 for node v_3

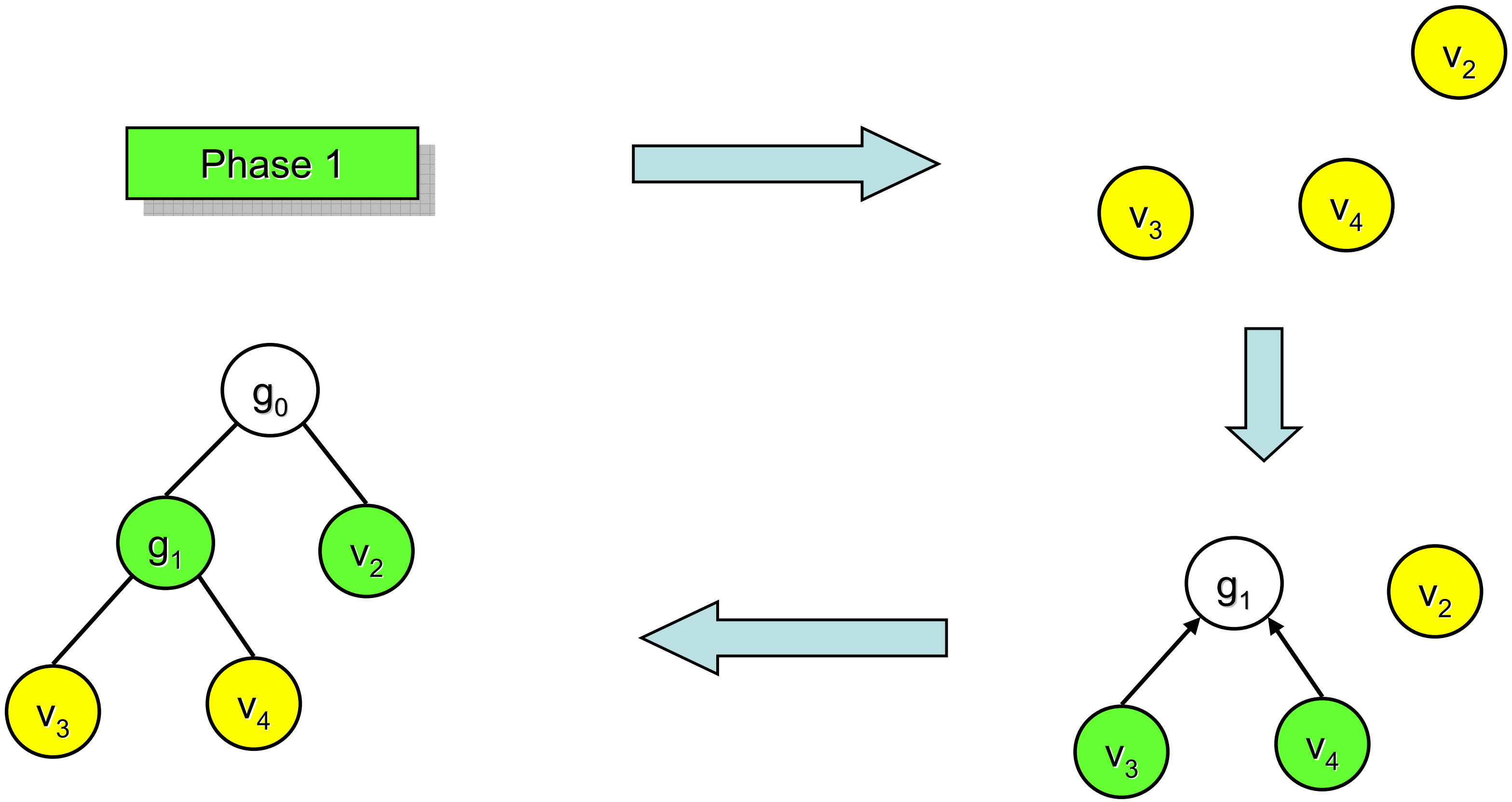


Workflow schema W_4 for node v_4

- … can be transformed into a taxonomy, by restructuring the schemas of all non-leaf nodes, $\mathbf{v_1}$ and $\mathbf{v_0}$, in a bottom-up fashion

Restructuring a schema hierarchy

- Every non-leaf schema in the hierarchy is replaced with an abstract schema that generalizes those of its children
  - The process is applied in a bottom-up way, i.e., form the leaves to the root of the hierarchy



g_0 is computed that abstracts v_2 and g_1

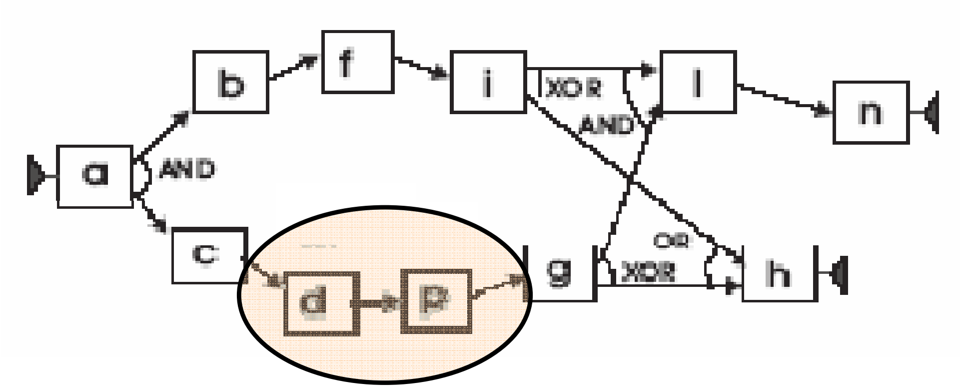
g_1 is computed that abstracts v_3 and v_4

How two schemas are generalized?

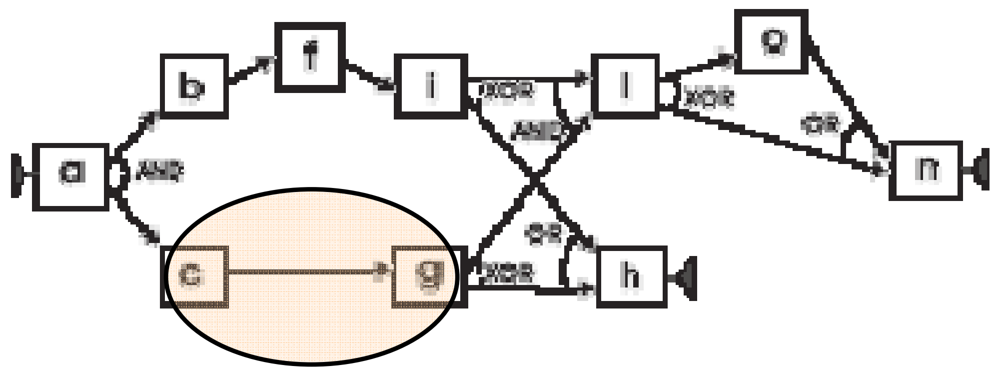
Computation of the generalized schema for a non-leaf node

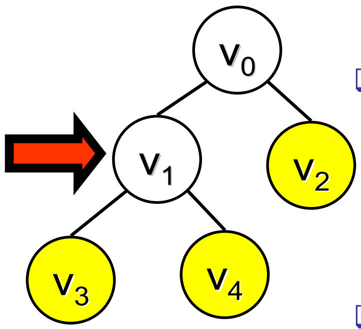
1. For each child schema *abstract* “specific” activities (activities that do not occurring in all children)
2. Merge all the children schemas into a single one
   - compute the union of the graphs, and adjust all constraints
3. *Abstract* “specific” activities appearing in the merged schema

Schema of v_3



Schema of v_4





- Only activities appearing in all children are surely kept in the generalized schema, while remaining ones, are abstracted
  - A group of “specific” activities is replaced with a complex activity that implies them all via IS-A or PART-OF relationships
- We need a strategy to recognize groups of “specific” activities that can be abstracted by the same higher-level activity ….

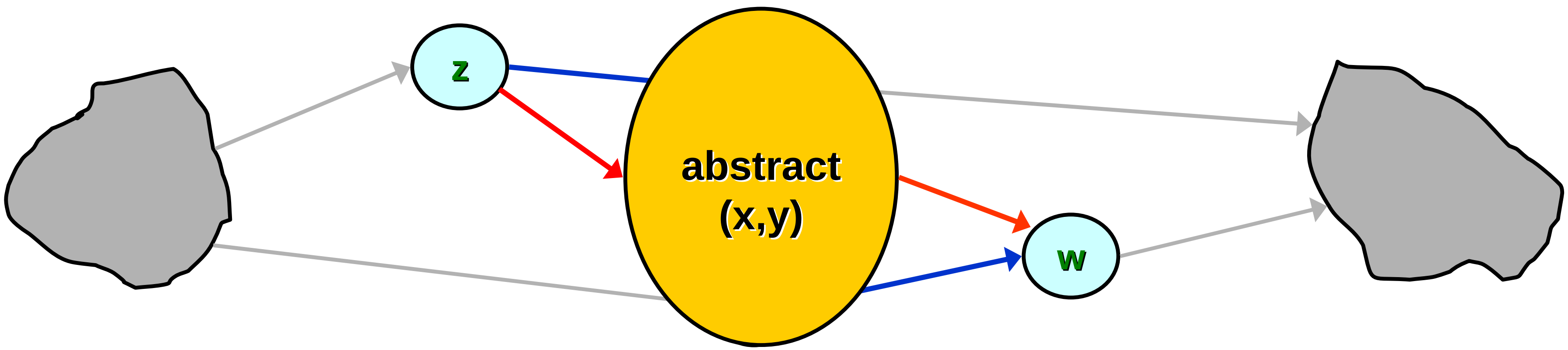
Merging activities to be abstracted

- Pair-wise approach
  - A pair of “specific” activities is greedily chosen for being abstracted together into a single higher-level activity
- A notion of safety w.r.t. merge for pairs of activities
  - for preventing the creation of “spurious” dependencies among not abstracted activities, in the generalized schema
- A series of affinity measures assessing how much two any “specific” activities are suitable to be merged
  - A “topological” affinity measureTopological $sim^E(x,y)$
    - how similar the neighborhoods of *x* and *y* are w.r.t. the flow graph
  - Two “semantical” affinity measures, $sim^D_P(x,y)$ and $sim^D_G(x,y)$
    - how similar *x* and *y* are w.r.t. the generalization/aggregation relationships stored in an abstraction dictionary *D*
  - Combined into an overall ranking function:

$$score^{\mathcal{D},E}(x,y) = \begin{cases} 0, \text{ if } (x,y) \text{ is not a merge-safe pair of activities} \\ max\{sim^E(x,y), sim^{\mathcal{D}}_P(x,y), sim^{\mathcal{D}}_G(x,y)\}, \text{ otherwise} \end{cases}$$

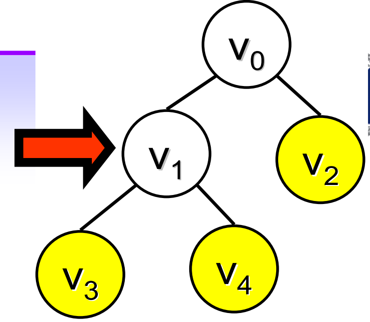
Merge-safe activities

- A couple of activities (*x, y*) is *merge-safe* w.r.t. a given an edge set *E*, if one of the following conditions holds:
  - *x* and *y* are directly linked by some edges in *E* and after removing these edges no other path exists between them
  - there is no path in *E* connecting *x* and *y*
- Only in the second case spurious dependencies may be introduced among other activities, whenever there are two activities *z* and *w* such that:
  - (*z,w*) *not in E**, and
  - { (*z, x*), (*y,w*) } *in E*

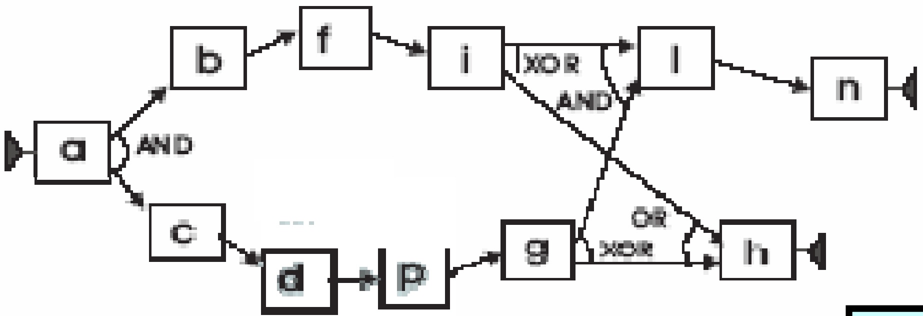


The approach in action:

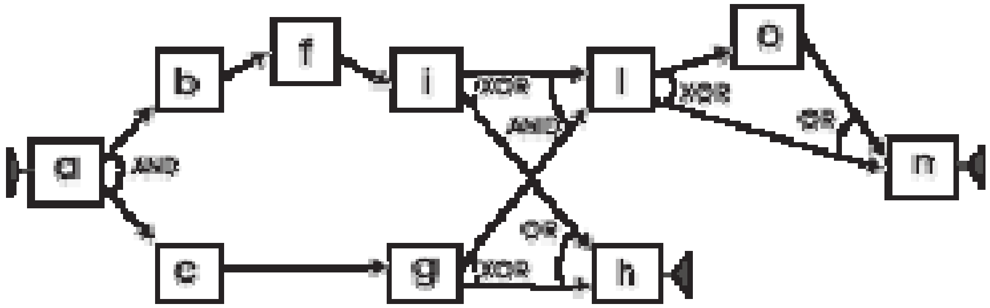
Restructuring a schema hierarchy



Schema of v_3

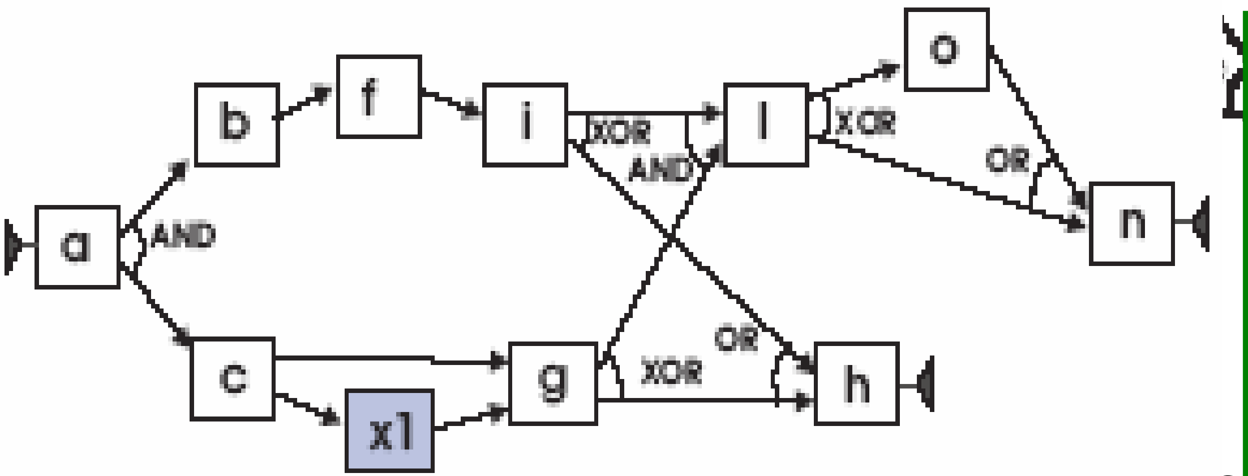


Schema of v_4



union

Generalized schema for v_1



Abstraction Disctionary

(assumed initially empty)

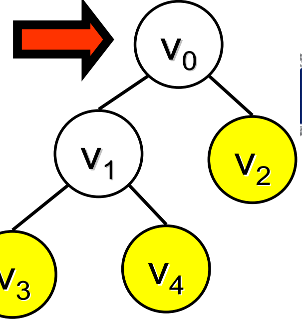
PART-OF = **{(d,x1), (p,x1) }**

ISA = **{ }**

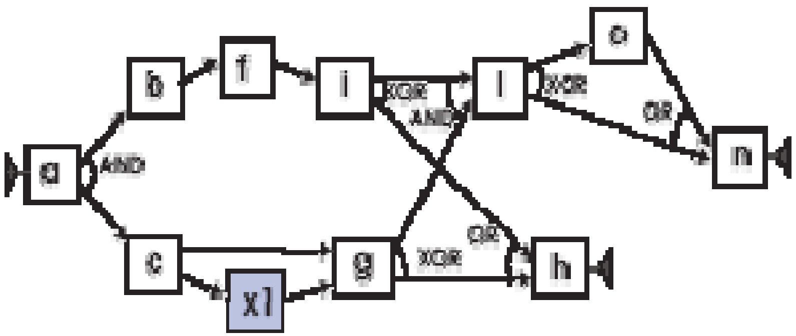
which are abstracted into activity **x1**, via PART-OF

The approach in action:

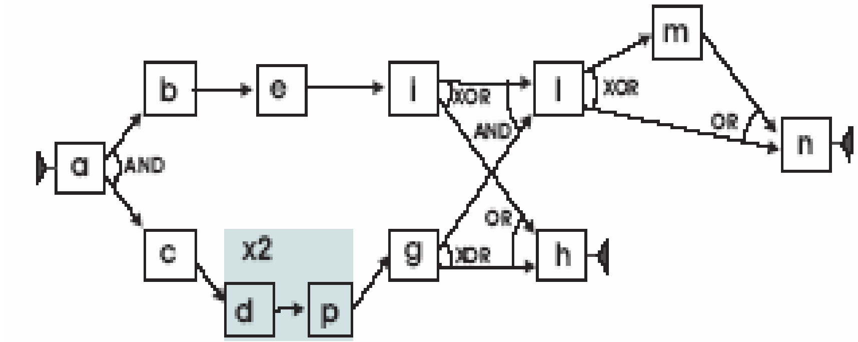
Restructuring a schema hierarchy



generalized schema of node v_1



schema of node v_2

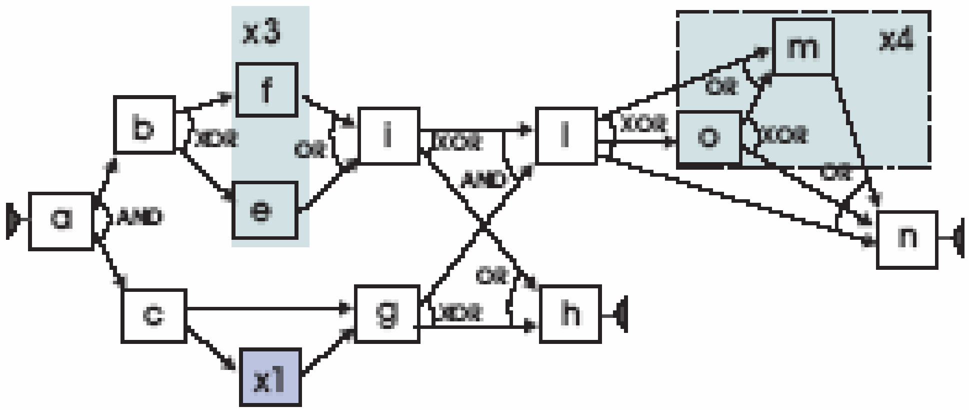


PART-OF = {(d,x1), (p,x1)}
ISA = { }

x2 contains the same basic activities as x1 (according to the dictionary)

therefore it is merged into x1 (no new activity is created)

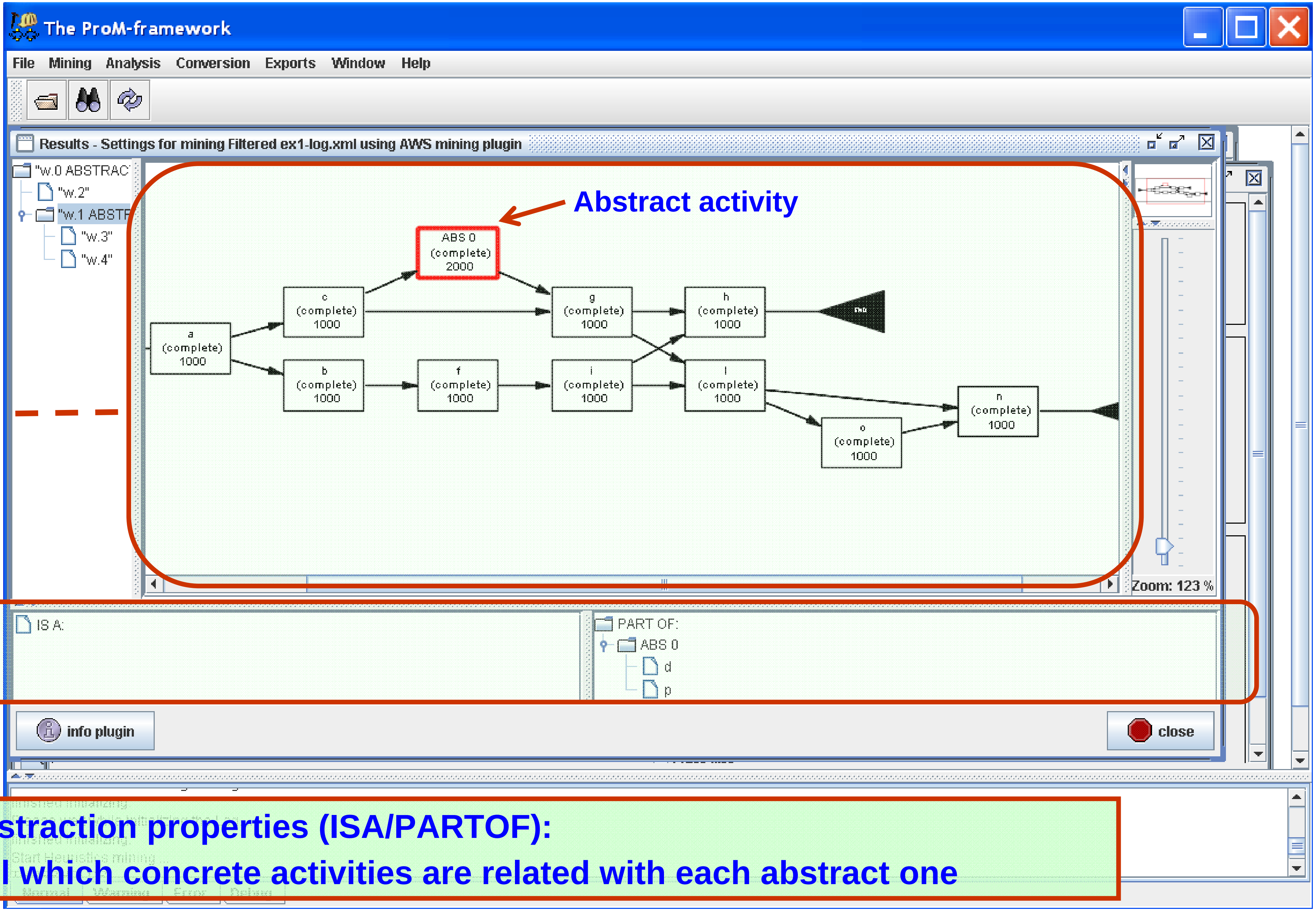
generalized schema of root v_0



PART-OF =
{ (d,x1), (p,x1),
(f,x3), (e,x3),
(o,x4), (m,x4) }

ISA = { }

Plugin AWS



Abstract Workflow

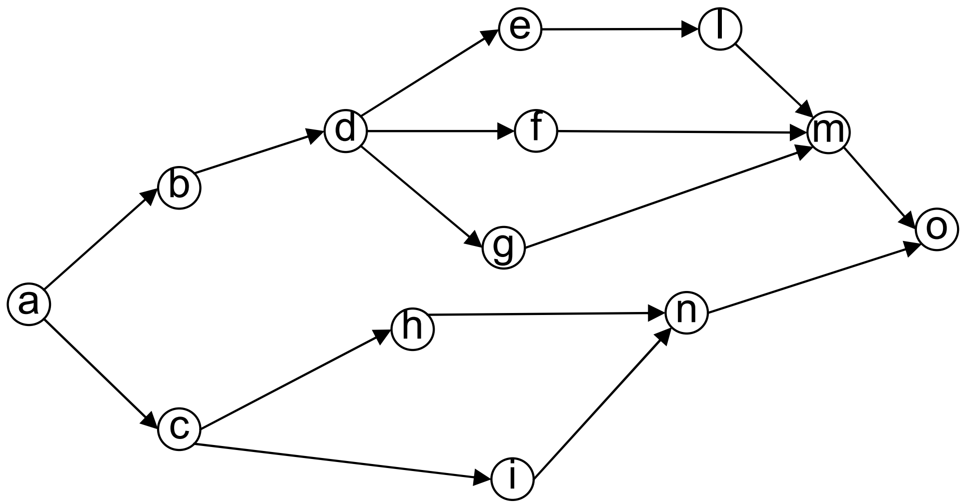
Abstraction properties (ISA/PARTOF):

Tell which concrete activities are related with each abstract one

Outline

- **Part I – Introduction to Process Mining**
  - Context, motivation and goal
  - General characteristics of the analyzed processes and logs
  - Classification of Process Mining approaches
- **Part II – Workflow discovery**
  - Induction of basic Control Flow graphs
  - Other techniques (α-algorithm, Heuristic Miner, Fuzzy mining)
- **Part III – Beyond control-flow mining**
  - Organizational mining
  - Social net discovery
  - Extension of workflow models
- **Part IV – Evaluation and validation of discovered workflow models**
  - Conformance Check
  - Log-based property verification
- **Part V – Clustering-based Process Mining**
  - Discovery of hierarchical process models
  - Discovery of process taxonomies
  - Outlier detection

Outlier Detection Challanges in process Mining

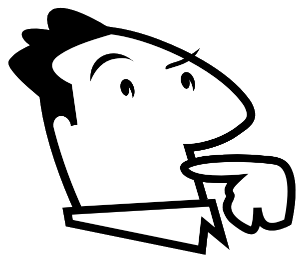


The application of traditional sequential outlier techniques may be misleading

- a lot of traces that only differ in the ordering between parallel tasks may be interpreted as anomalous (***false positive***)

Considering the compliance with an ideal schema may fails too

- some trace might well be supported by a model, yet representing anomalous behavoiurs (***false negative)***



An approach to outlier detection for process logs

- Core Idea
  - Find out homogenous clusters of traces sharing the same behaviour in executing tasks
  - Outliers as those individuals that hardly belong to any of the computed clusters or that belong to clusters whose size is definitively smaller than the average cluster size.
- Two phase computation approach
  - Extraction of *structural patterns* describing "normal" process behaviour
  - Co-Clustering of log traces and associated patterns