Process Oriented Data Science

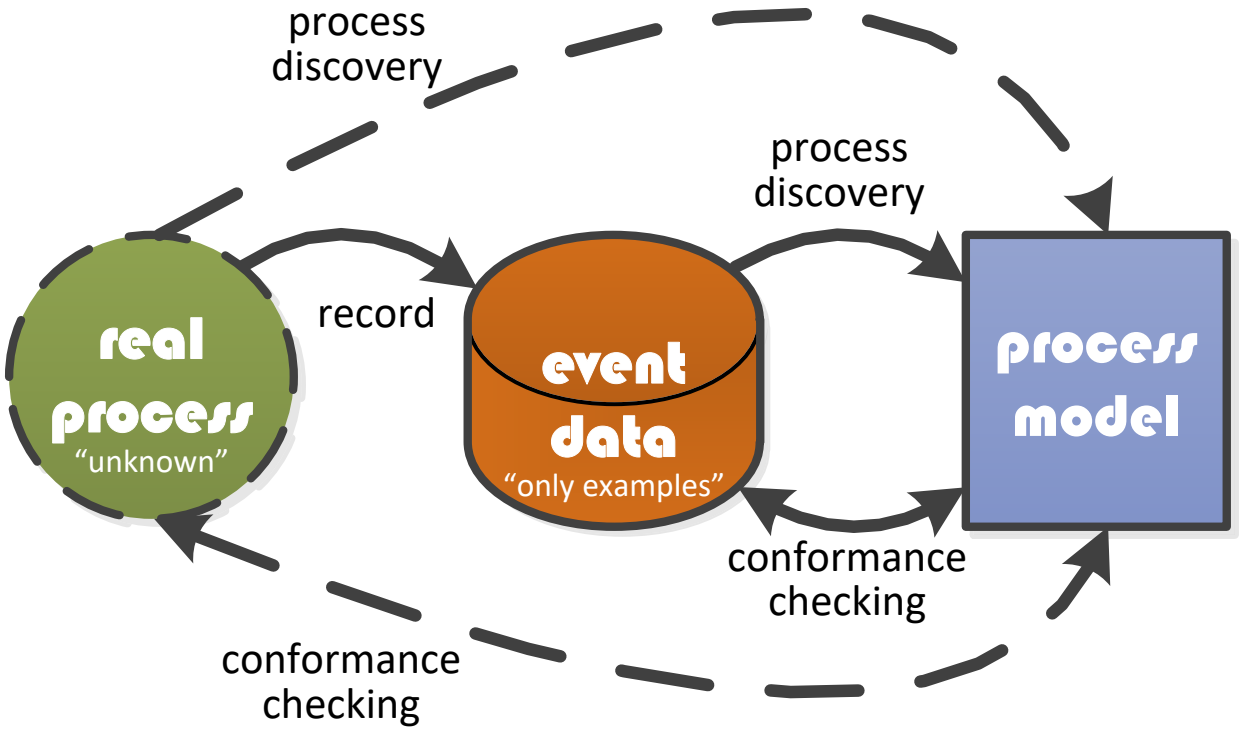
UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Campus d'Excel·lència Internacional

Josep Carmona
Computer Science Department

Outline

- M1: Process Mining Overview, Positioning & Preliminaries (Event data & Process Models)
- M2: Process Discovery
- **M3: Conformance Checking**
- M4: Process Enhancement



[Source: Wil van der Aalst]

Is the process model a correct reflection of the real process?

Recent Report: MarketsAndMarkets

"Process Analytics Market worth 1,421.7 Million USD by 2023"

"The process conformance segment is expected to be the fastest-growing segment during the forecast period."

"… the process mining software converts the event logs into a process model and checks them against the ideal and pre-defined processes. Hence, deviations between the derived business process model and ideal processes can be diagnosed, and non-conformance can be highlighted and visualized."

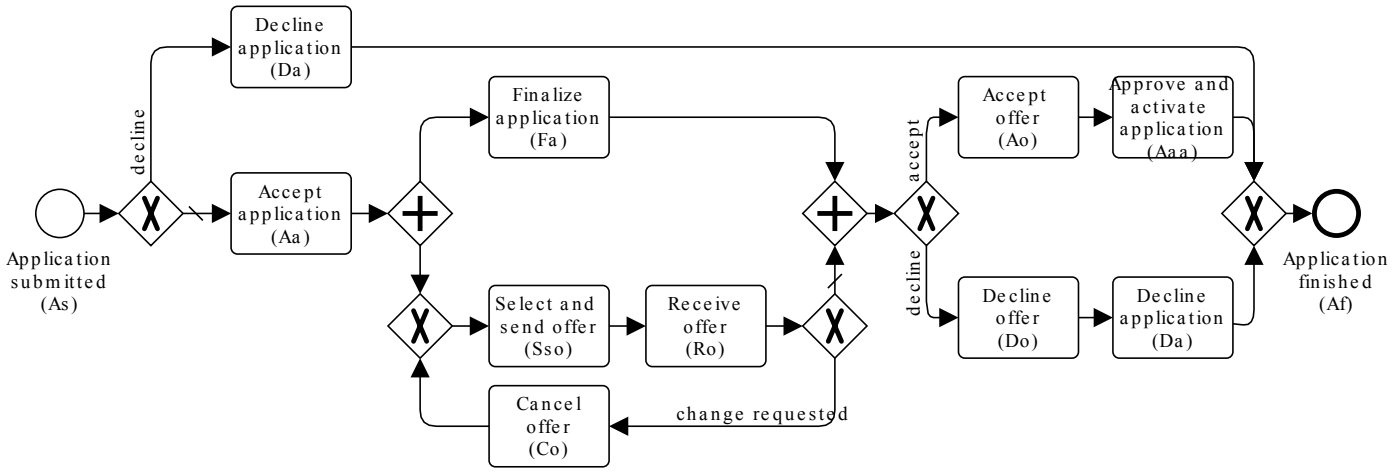
Report TC 6371: Process Analytics Market by Process Mining Type (Process Discovery, Process Conformance & Process Enhancement), - Global Forecast to 2023

The Loan Application Process

A process defines how a loan application is handled once it has been submitted, with the outcome being its positive or negative assessment. A first activity is a check for eligibility of the respective applicant. Subsequently, a decision is taken, which determines whether an application is declined or accepted. If accepted, the application is finalised and an order is selected for further processing.

…

Process Models: BPMN



Control-flow Data Organizational Other: Flexibility, …

Covered Partially Covered Not Covered

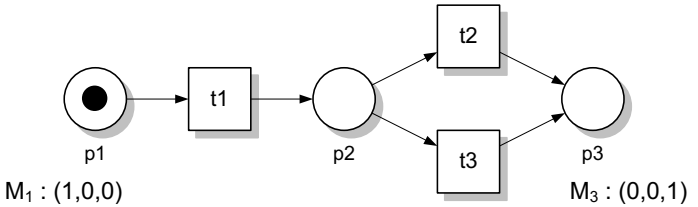
The Spectrum of Conformance Checking

- Conformance checking is about detecting and analyzing deviations between observed/recorded and modelled behavior.
- Modelled behaviour: Process models
- Observed behaviour: Event data (e.g., event logs)
- Conformance Artefact Construction:
  - Replaying event data on process models allows to discover deviations.
  - Two replay methods:
    - Token-Replay (intuitive, heuristic)
    - Alignments (sound, optimal).

Petri Nets as a Formal Foundation

Petri net syntax

- Directed graph consisting of places, transitions, and arcs between them
- Places = {p1,p2,p3}
- Transitions = {t1,t2,t3}
- Arcs {(p1,t1), (t1,p2), (p2,t2), (t2,p3), (p2,t3), (t3,p3)}



Petri net semantics

- State of a Petri net (the *marking*) is a distribution of tokens over places
  - Initial Marking = (1,0,0)
- A transition is enabled in a marking if all its input places carry at least one token
  - t1 is enabled in the initial marking
- Firing a transition in a marking leads to a new marking, such that:
  - A token is removed from all input places that are not output places
  - A token is added to all output places that are not input places
  - The number of tokens does not change for all other places

Notes on Petri Nets

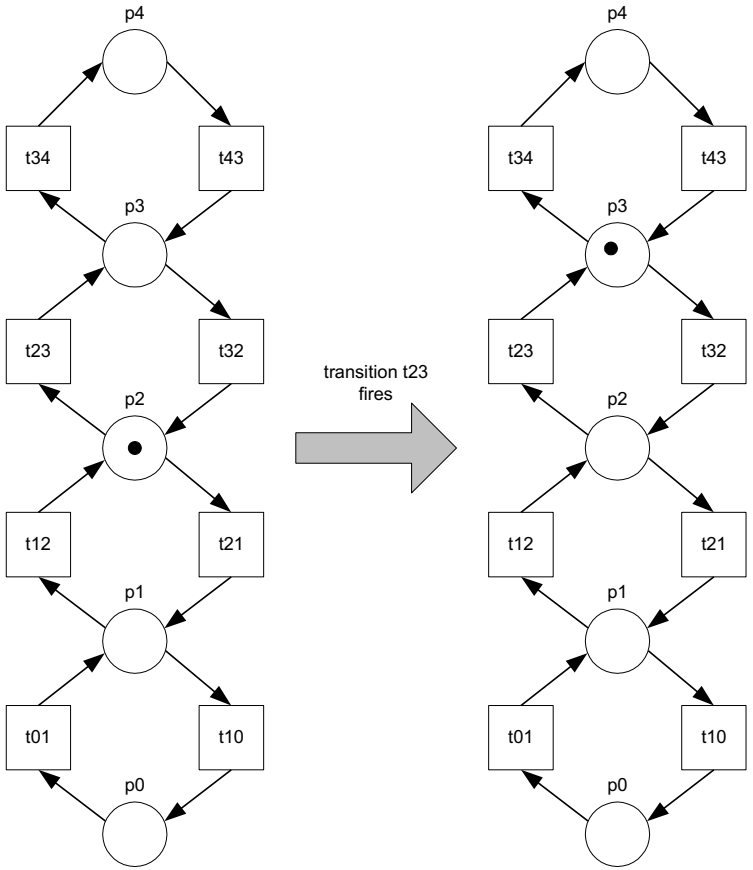
Firing of a transition is atomic

Multiple transitions may be enabled,
but only one fires as part of a state change

The above leads to non-determinism

The number of tokens in a net may vary

The network, the net structure, is static



from: Wil van der Aalst, Petri net refresher

Workflow Nets

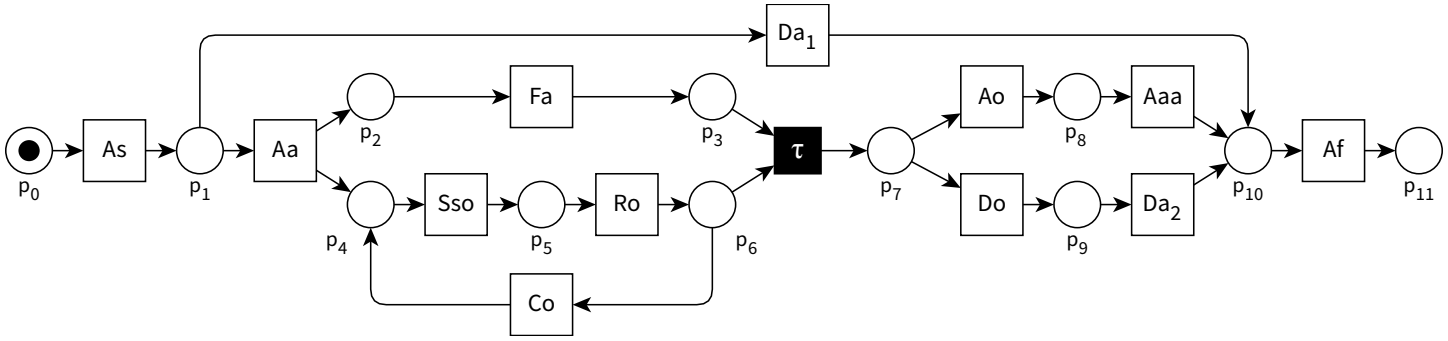
Use Petri nets to model business processes

- Transitions represent activities; places represent (partial) states of the business process
- Arcs represent control flow dependencies; tokens represent instances

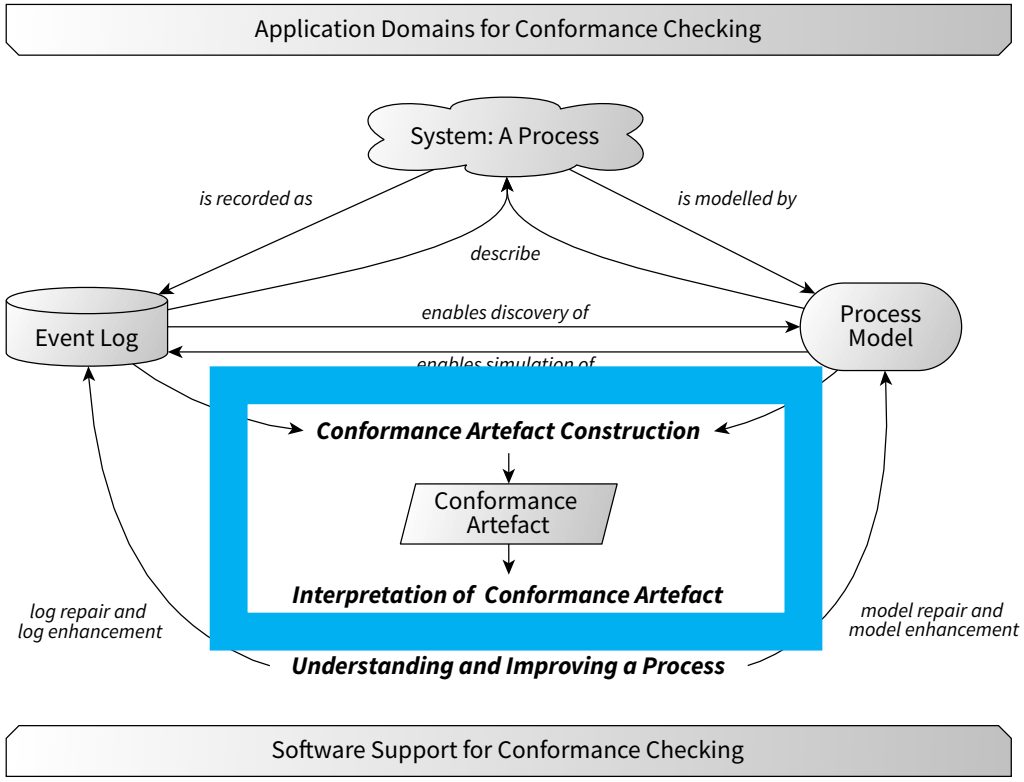
In fact: Modelling languages such as BPMN are rooted in Petri net systems

Workflow net: Petri net with:

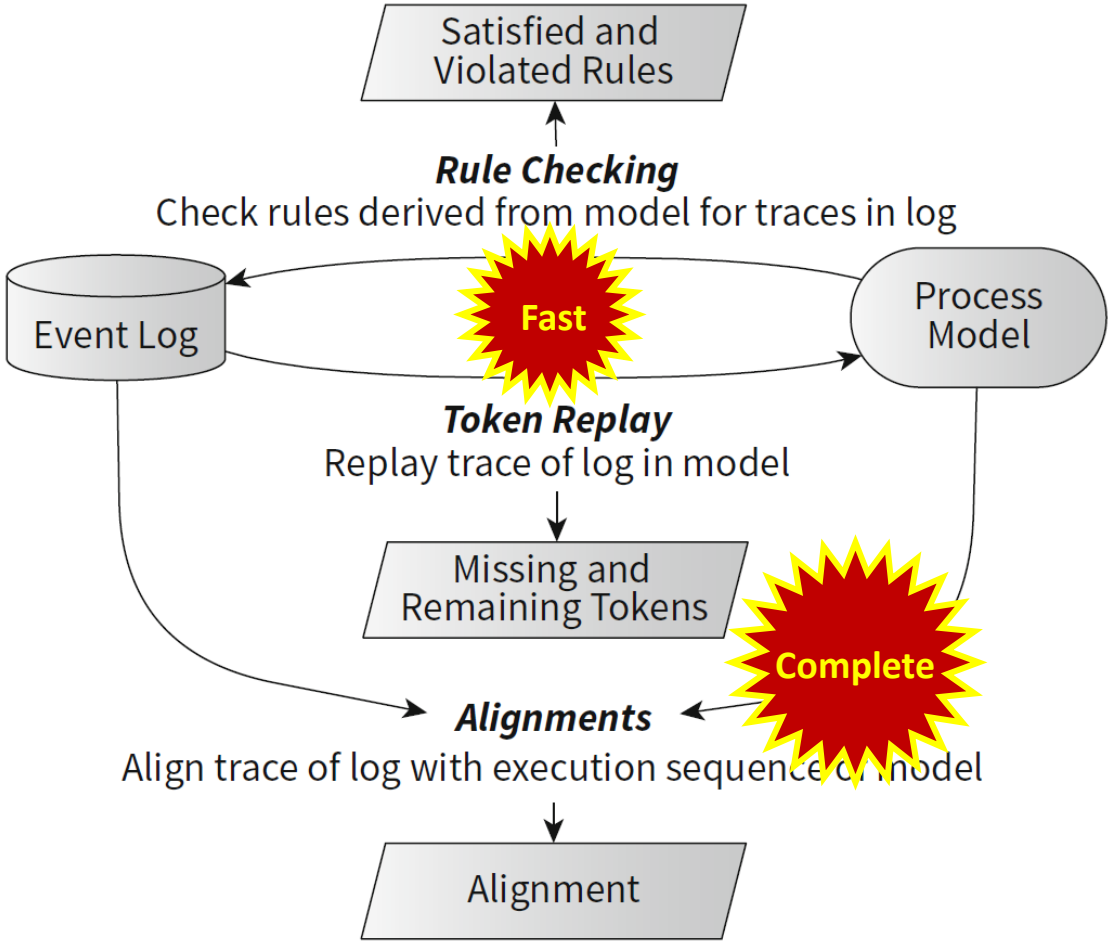
- One distinguished input place **i** with no incoming arcs
- One distinguished output place **o** with no outgoing arcs
- Every place and transition being on a path from **i** to **o**



The Spectrum of Conformance Checking



Computing Conformance Artefacts



Computing Conformance Checking Artefacts

Rule Checking (30’)

Idea of Rule Checking

General idea:

- Process model constrains the possible behaviour of a process
- Formalise constraints by rules that execution sequences have to obey to
- Then, verify these rules for the traces of a log

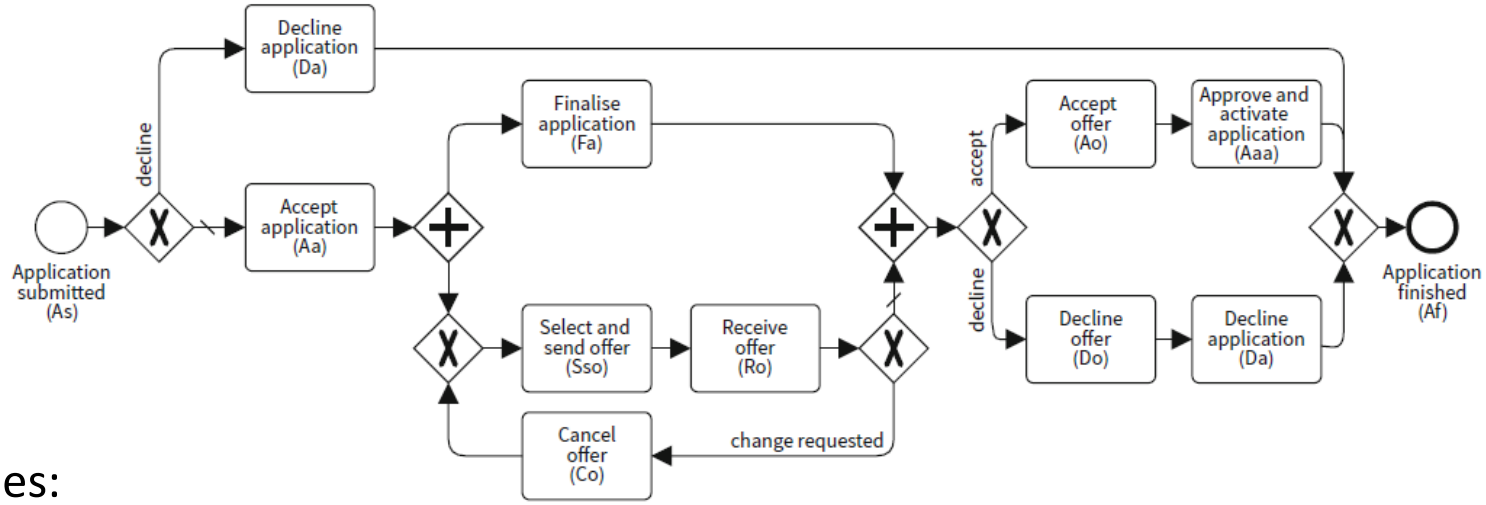
Rule checking focuses on the fitness dimension

- Traces are fitting, if they satisfy the rules
- Traces are non-fitting, if rules are violated

The dimension of precision is not targeted

- Specificity of rules is neglected
- Many more traces, not contained in the log, could also satisfy the rules

Illustration of Rules



Rules:

- R1: An application can be accepted (Aa) at most once
- R2: An accepted application (Aa), that must have been submitted (As) earlier, and eventually an offer needs to be selected and sent (Sso) for it
- R3: An application must never be finalised (Fa), if the respective offer has been declined (Do) already
- R4: An offer is either accepted (Ao) or declined (Do), but cannot be both accepted and declined

What rules?

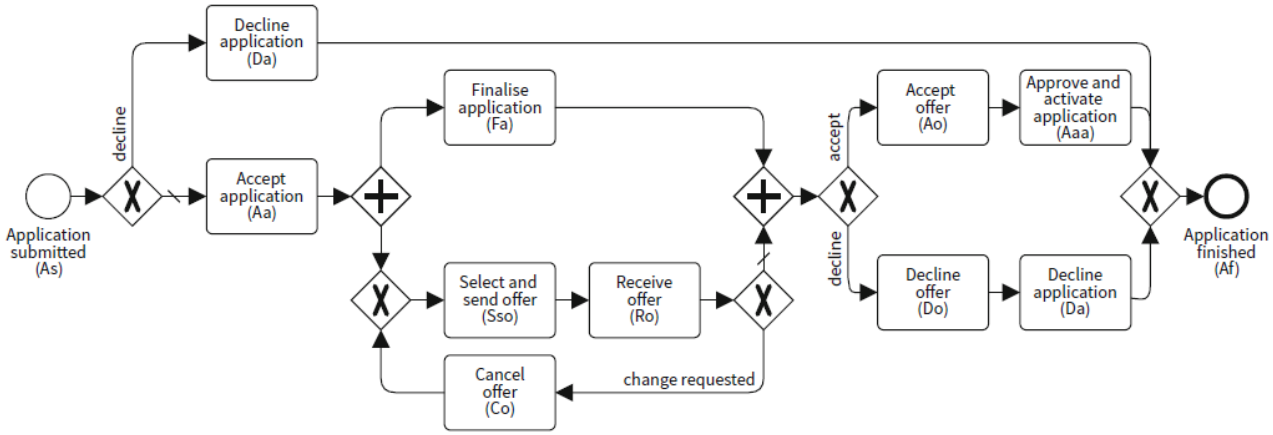
Often: unary and binary rules over activities

- Specify how a single activity is executed or how the executions of a pair of activities relate to each other
- N-ary rules would lead to combinatorial explosion

Common types of rules:

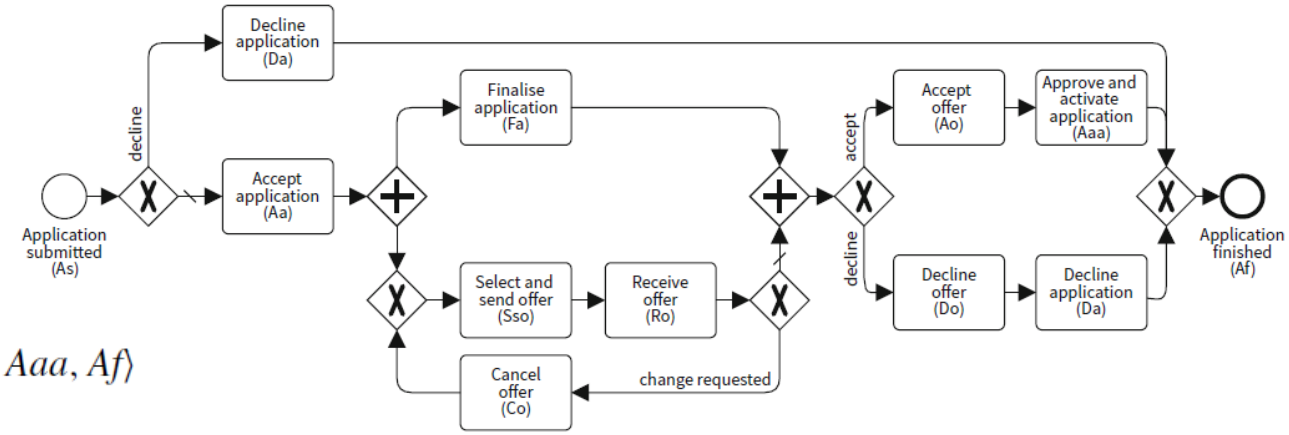
- Cardinality rules: Upper and/or lower bound for the number of executions of an activity
- Precedence and response rules: An activity is always preceded/succeeded by another one
- Ordering rules: Activities are independent, but if they are both executed, they are ordered
- Exclusiveness rules: Activities must not be executed in the same case

Cardinality Rules



Activity	***As***	***Da***	***Aa***	***Fa***	***Sso***	***Ro***	***Co***	***Ao***	***Aaa***	***Do***	***Af***
Cardinality:	**[1, 1]**	**[0, 1]**	**[0, 1]**	**[0, 1]**	**[0, *n*]**	**[0, *n*]**	**[0, *n*]**	**[0, 1]**	**[0, 1]**	**[0, 1]**	**[1, 1]**
$T_1 = \langle As,Aa,Sso,Ro,Ao,Aaa,Aaa \rangle$											
	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
$T_2 = \langle As,Sso,Fa,Ro,Co,Ro,Aaa,Af \rangle$											
	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$T_3 = \langle As,Aa,Sso,Ro,Fa,Ao,Do,Da,Af \rangle$											
	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

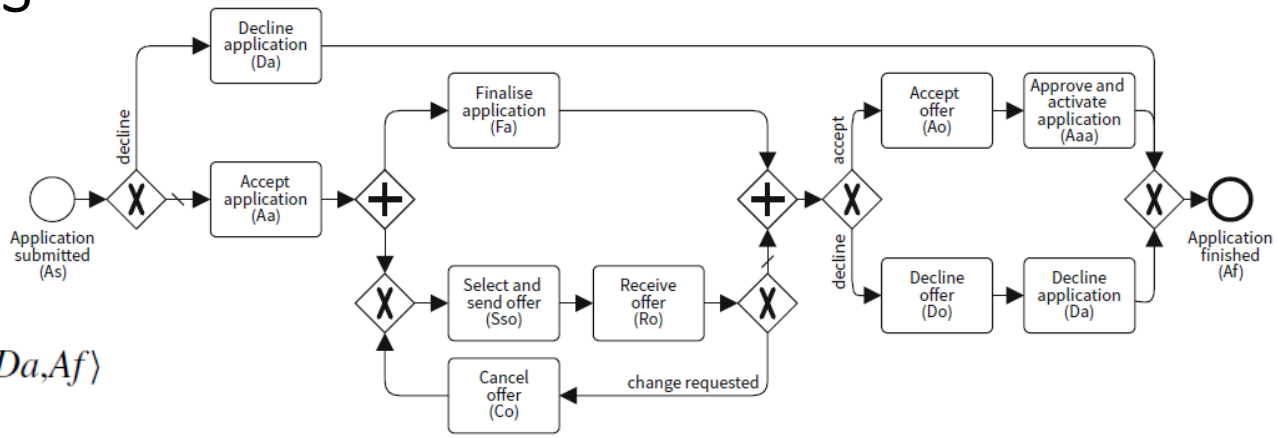
Precedence Rules



$T_2 = \langle As, Sso, Fa, Ro, Co, Ro, Aaa, Af \rangle$

	As	*Da*	*Aa*	*Fa*	*Sso*	*Ro*	*Co*	*Ao*	*Aaa*	*Do*	*Af*
As											
Da	✓										
Aa	✓										
Fa	✓		✗								
Sso	✓		✗								
Ro	✓		✗		✓						
Co	✓		✗		✓	✓					
Ao	✓		✓	✓	✓	✓					
Aaa	✓		✗	✓	✓	✓		✗			
Do	✓		✓	✓	✓	✓					
Af	✓										

Exclusiveness Rules



$T_3 = \langle As, Aa, Sso, Ro, Fa, Ao, Do, Da, Af \rangle$

	As	*Da*	*Aa*	*Fa*	*Sso*	*Ro*	*Co*	*Ao*	*Aaa*	*Do*	*Af*
As	✓										
Da		✓						✗	✓		
Aa			✓								
Fa				✓							
Sso											
Ro											
Co											
Ao		✗						✓		✗	
Aaa		✓							✓	✓	
Do								✗	✓	✓	
Af											✓

Feedback on Non-Conformance

Rule-checking gives information on fitness of trace

- Is the model well-suited to explain the recorded trace?
- Yes, if rules are satisfied; no, if rules are violated
- Fine-granular feedback on the level of activities

Aggregated feedback on the level of a log

- Rules that are frequently violated point to general conformance issues
- Indicator for hot-spots
- Association rule mining to highlight dependencies between two violations v and v': Ratio of number of traces showing both violations and number of traces showing only violation v

Fitness Measure

Counting of satisfied rules enables quantification of fitness:

$$\text{fitness}(\sigma, M) = \frac{|\{r \in R_M \mid r \text{ is satisfied by } \sigma\}|}{|R_M|}$$

$$\text{fitness}(L, M) = \frac{|\{r \in R_M \mid r \text{ is satisfied by all } \sigma \in L\}|}{|R_M|}$$

Obviously, the above measures depend on the chosen set of rules

- Values under different rule sets cannot be compared
- Measures are influence by how many rules are affected by a single conformance issue (e.g., missing execution of an activity)

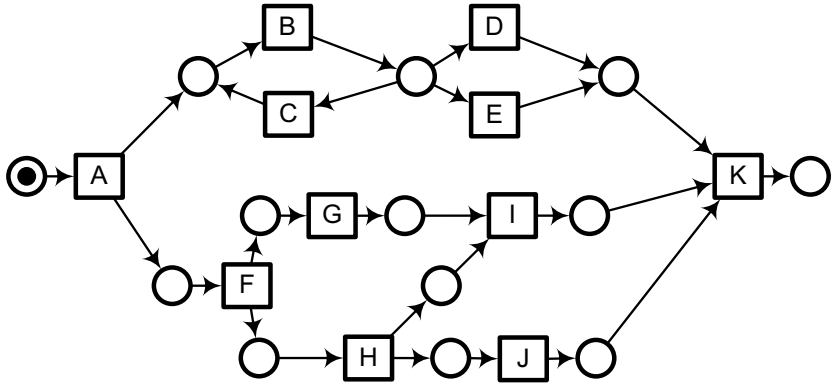
Computation of Rules

Many sets of rules can be computed efficiently

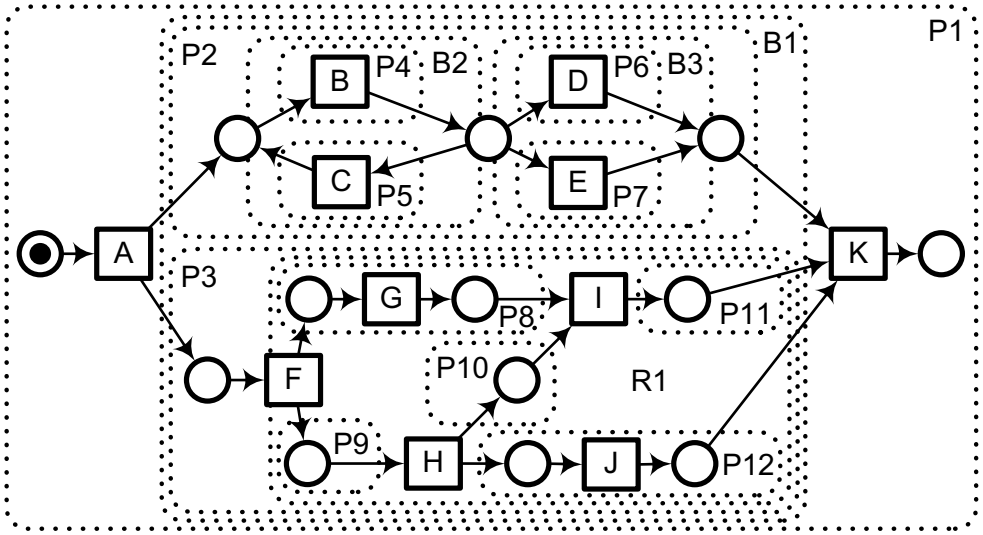
- Assumptions on class of process model (structural and behavioural)
- Exploit structure of a process model
- Thereby, avoid state space exploration

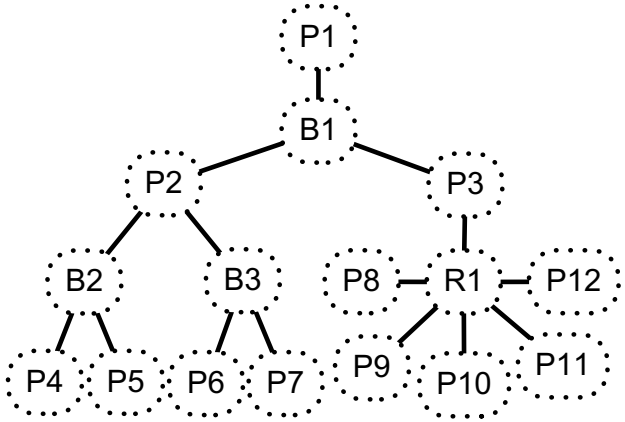
Example:

- Cardinality rule [0,n] for C
- Exclusiveness rule for (D,E)



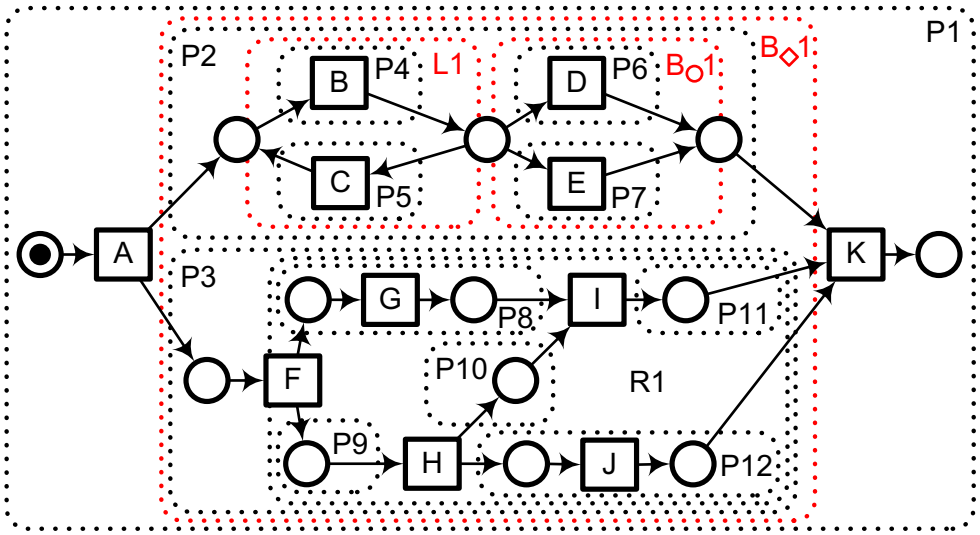
Computation of Rules Cont.

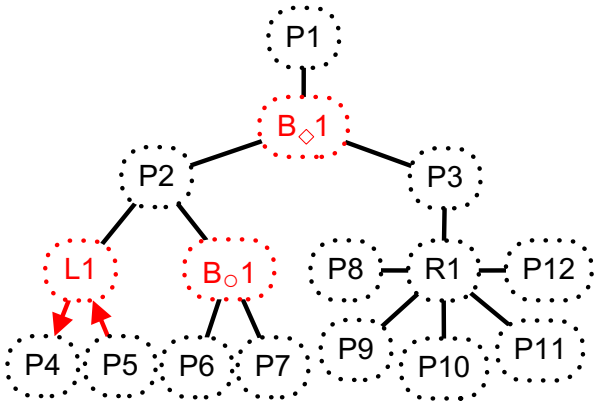




Parse tree of the structure of a process model

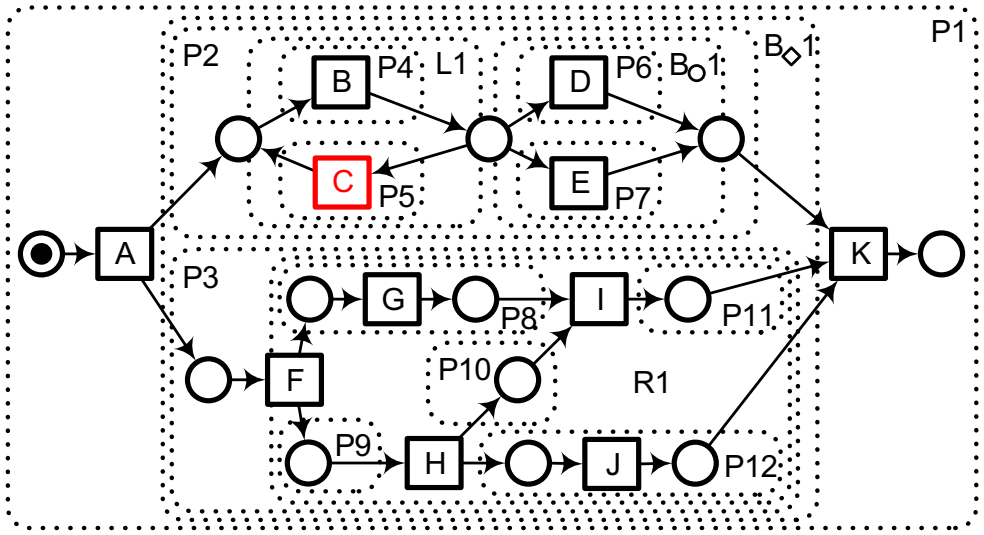
Computation of Rules Cont.

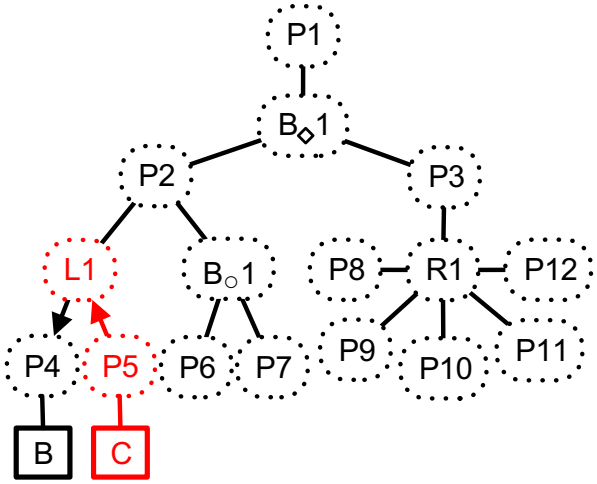




Annotation of the parse tree with behavioural information

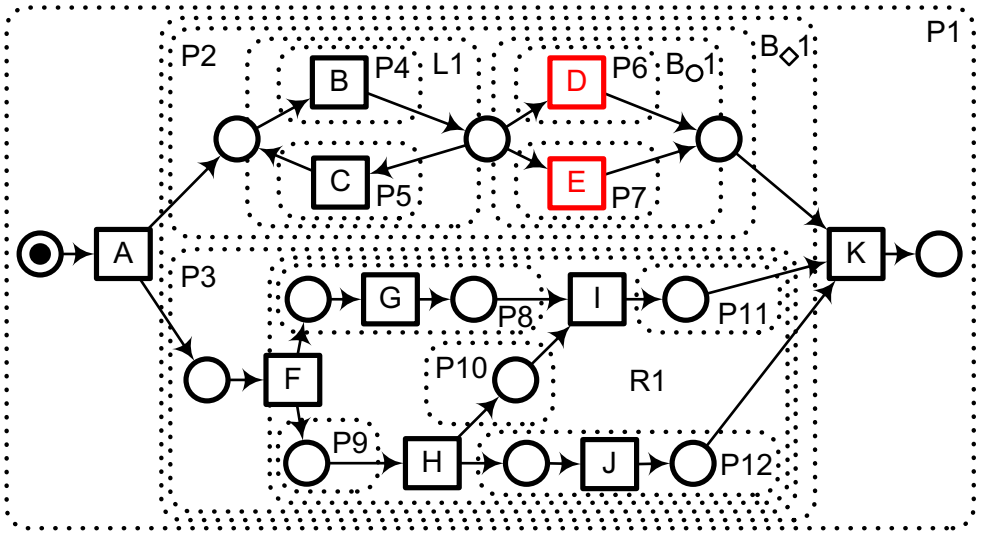
Computation of Rules Cont.

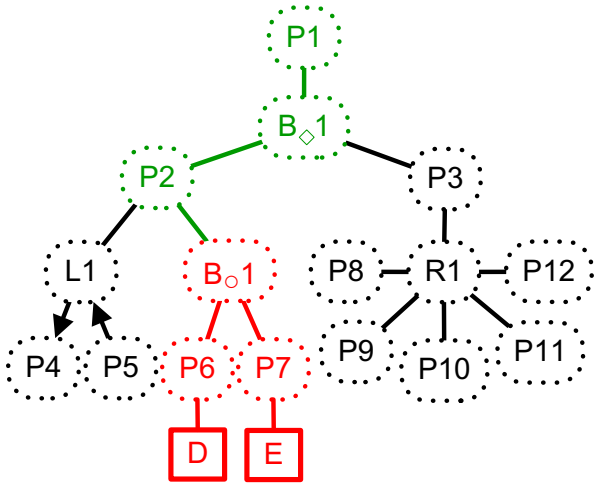




Cardinality rule for C

Computation of Rules Cont.

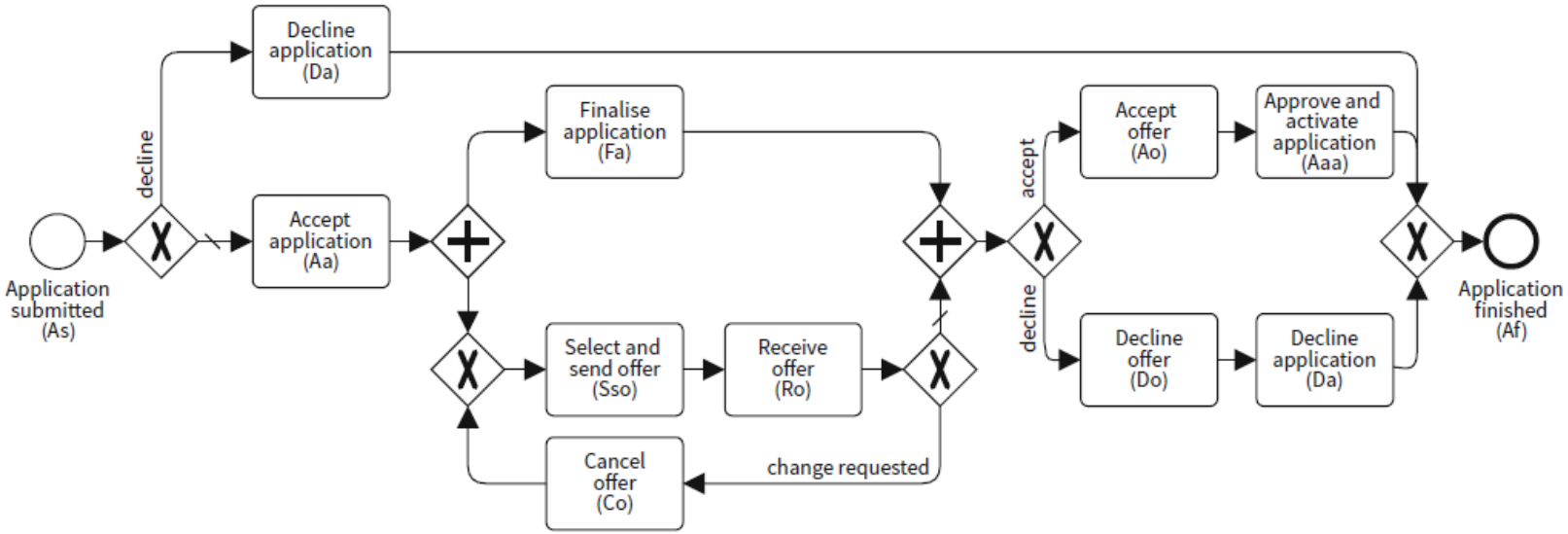




Exclusiveness rule for D and E

Expressiveness of Rules

Illustrated set of rules yields incomplete conformance results



$$T_4 = \langle As, Aa, Sso, Ro, Ro, Co, Fa, Ao, Aaa, Af \rangle$$

Expressiveness of Rules Cont.

Can this be fixed by introducing further rules?

In theory, yes, BUT:

- An exponential number of rules would be needed to fully characterise the behaviour of process model (specifically, a regular language)

Hence, if striving for completeness, rule checking becomes

- Inefficient—an exponential number of rules need to be checked
- Ineffective—the result of rule checking is overwhelming to users due to its exponential size

Computing Conformance Checking Artefacts

Token Replay (30’)

Idea of Token Replay

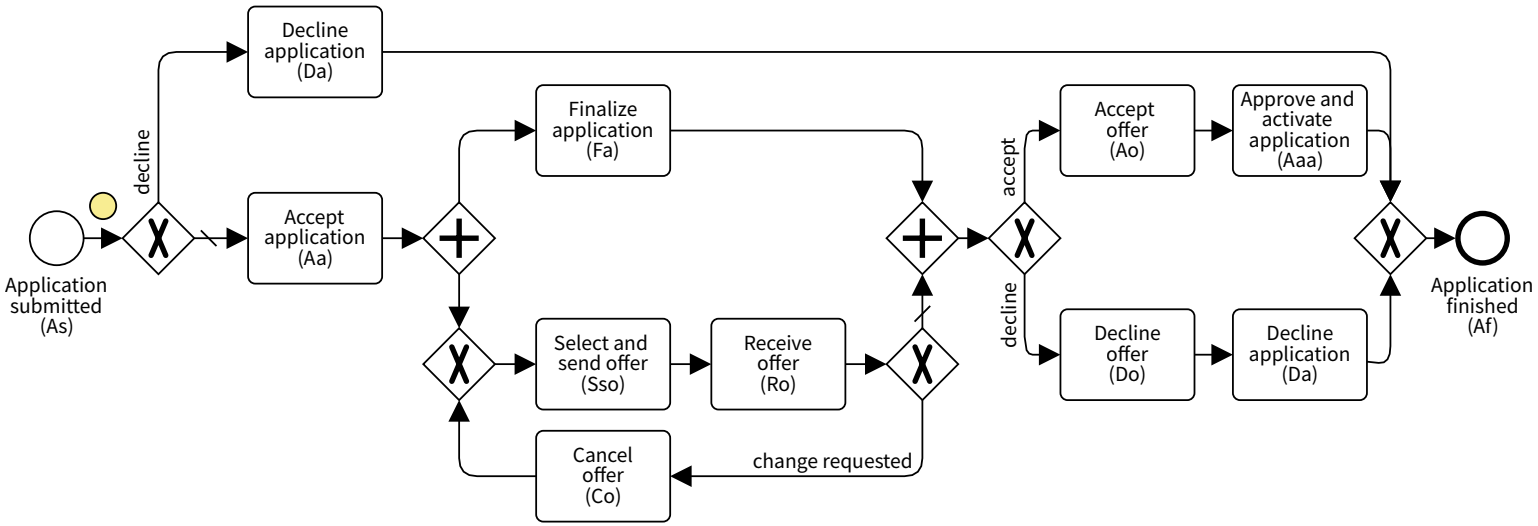
Idea

- Try to “replay” each trace in the model by executing activities according to the trace
- Not ok:
  - Activity is executed although it is not enabled
  - Tokens remain in the model and are not consumed

Procedure:

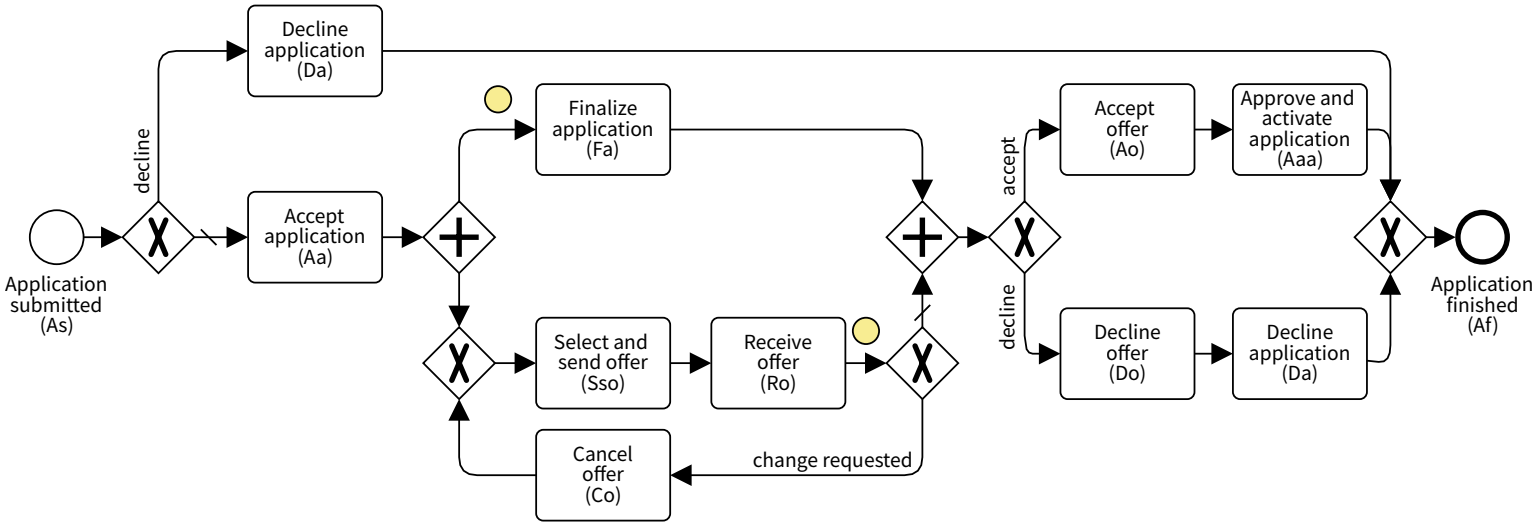
1) Initialise the replay procedure by setting the *first event* of the trace as the *current event* of the replay
2) Locate the activity for which execution is signalled by the *current event* of the trace in the model
3) Replay the *current event* of the trace by executing the identified activity in the *current state* of the model by:
   - Consuming a token on the incoming arc of the activity, even if there is none (if so, record as missing)
   - Producing a token on the outgoing arc of the respective task (if there is any)
4) Set the state reached as the *current state* of the process model
5) If the current event is not the last in the trace, consider the next event in the trace as the current event and continue the replay from step 2 onwards

Token-Based Replay



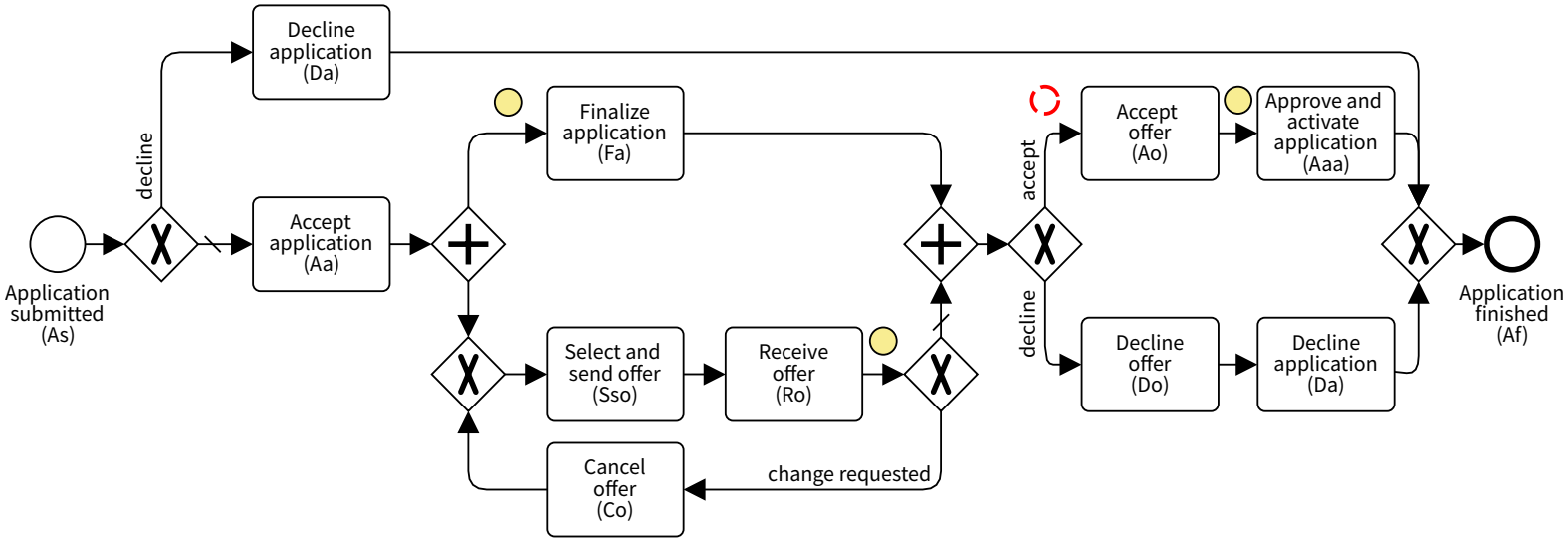
Observed Trace: < As, Aa, Sso, Ro, Ao, Aaa, Aaa >

Token-Based Replay



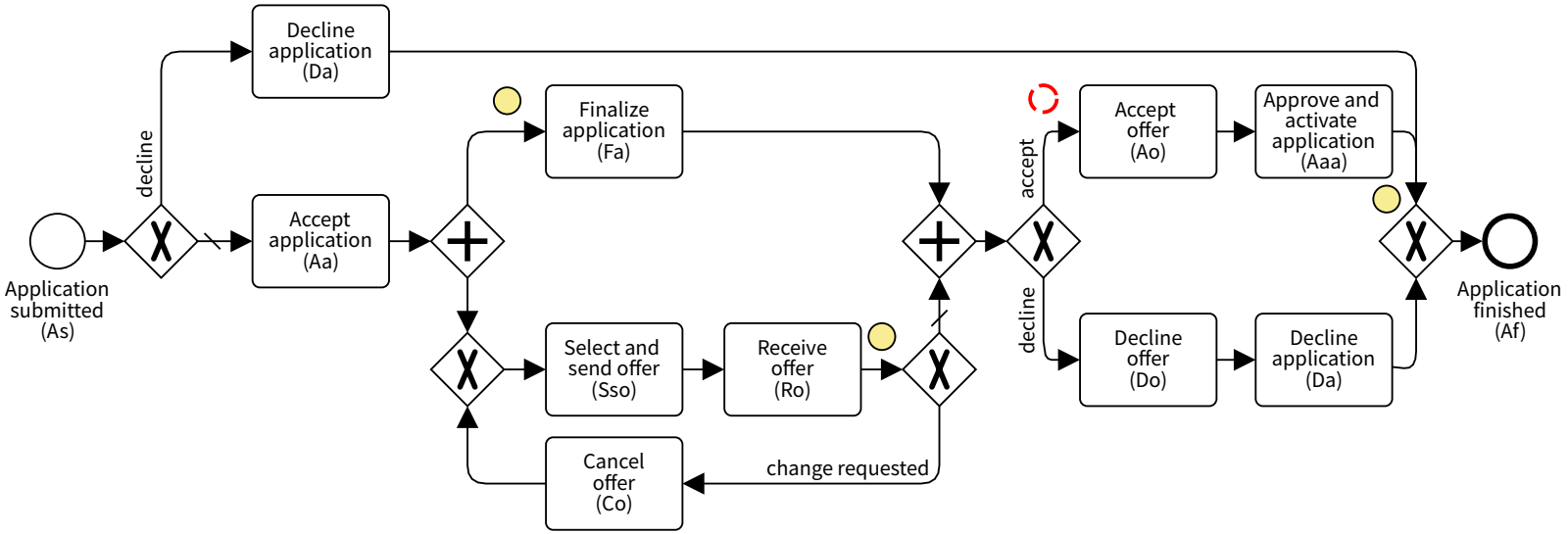
Observed Trace: < As, Aa, Sso, Ro, Ao, Aaa, Aaa >

Token-Based Replay



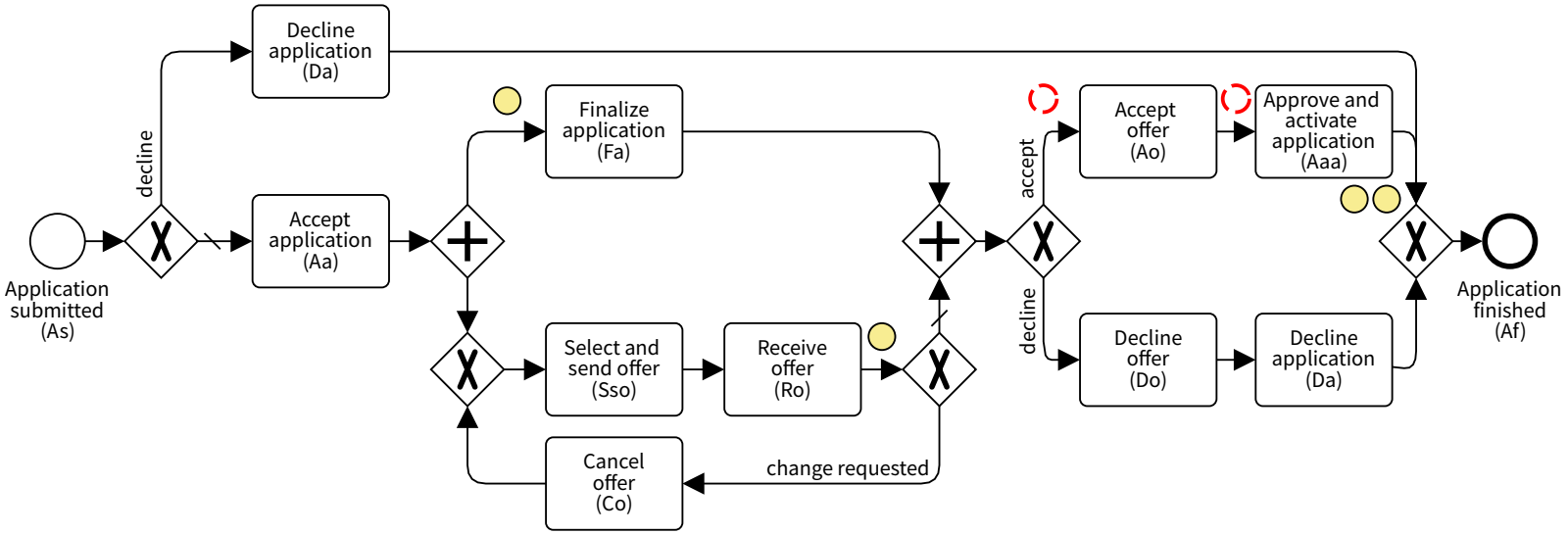
Observed Trace: < As, Aa, Sso, Ro, Ao, Aaa, Aaa >

Token-Based Replay



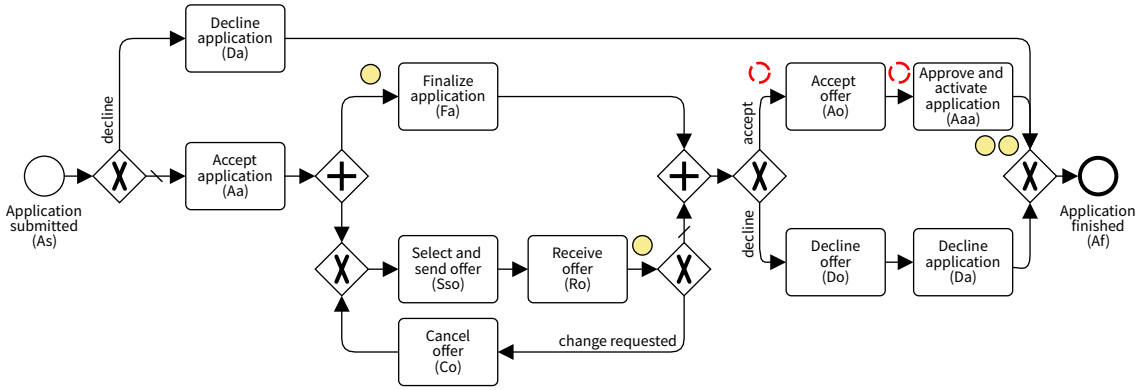
Observed Trace: < As, Aa, Sso, Ro, Ao, Aaa, Aaa >

Token-Based Replay



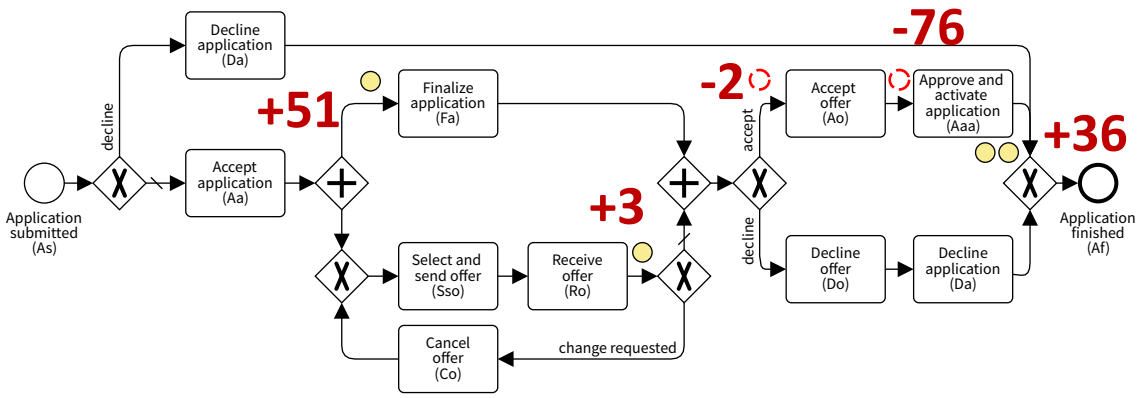
Observed Trace: < As, Aa, Sso, Ro, Ao, Aaa, Aaa >

Local Feedback on Non-Conformance



Deviations ≈ Tokens Missing or Tokens Remaining after replaying a trace

Aggregated Feedback on Non-Conformance



- Aggregated information about missing and remaining tokens
- Identification of “hotspots” of non-conformance
- Highlights major issues when replaying log and separates some from noise

Fitness Measure

Determine ratios based on missing and remaining tokens:

$$\text{fitness}(\sigma, M) = \frac{1}{2}\left(1 - \frac{missing(\sigma, M)}{consumed(\sigma, M)}\right) + \frac{1}{2}\left(1 - \frac{remaining(\sigma, M)}{produced(\sigma, M)}\right)$$

$$\text{fitness}(L, M) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} missing(\sigma, M)}{\sum_{\sigma \in L} consumed(\sigma, M)}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} remaining(\sigma, M)}{\sum_{\sigma \in L} produced(\sigma, M)}\right)$$

Overall conformance assessment based on aggregated fitness measure

Condenses (non-)conformance into a single value

Yet, hard to interpret in terms of the absolute value

Limitations

Replay is fast and works reasonable well in practice

But: Replay may become non-deterministic in two situations:

- It may be impossible to unambiguously locate the activity to execute when replaying an event
- It may be impossible to unambiguously characterize the state reached after replaying an event

Idea of Token Replay

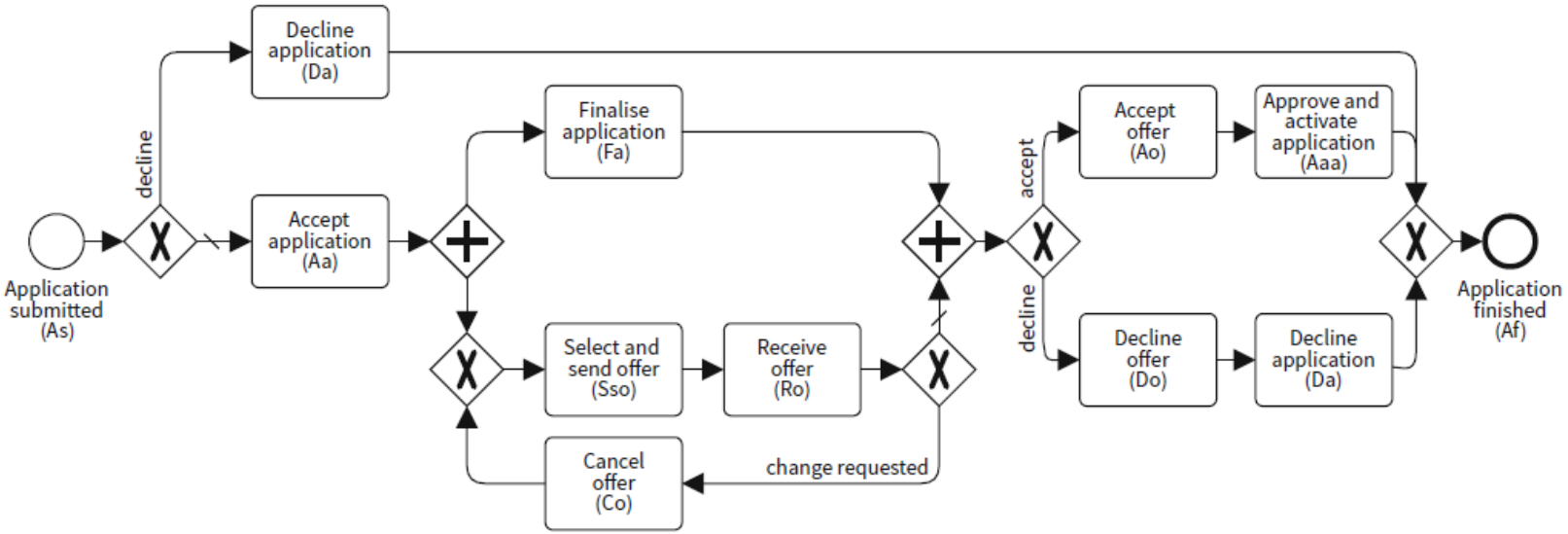
Idea

- Try to “replay” each trace in the model by executing activities according to the trace
- Not ok:
  - Activity is executed although it is not enabled
  - Tokens remain in the model and are not consumed

Procedure:

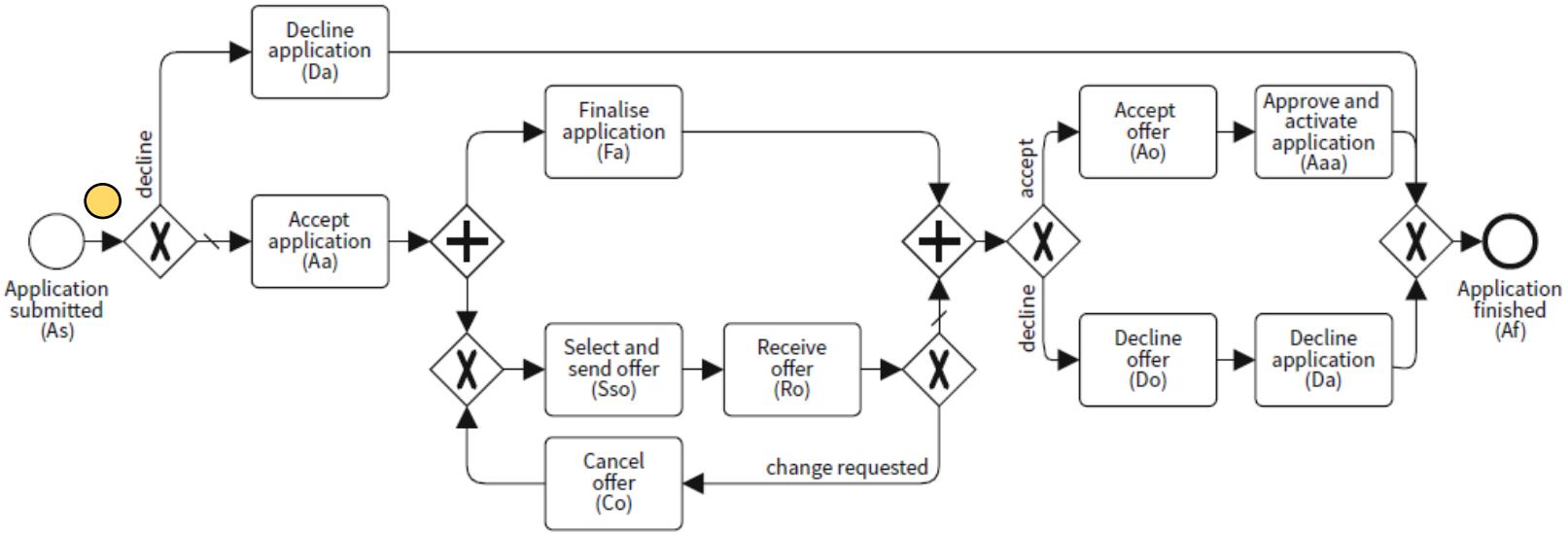
1) Initialise the replay procedure by setting the *first event* of the trace as the *current event* of the replay
2) Locate the activity for which execution is signalled by the *current event* of the trace in the model
3) Replay the *current event* of the trace by executing the identified activity in the *current state* of the model by:
   - Consuming a token on the incoming arc of the activity, even if there is none (if so, record as missing)
   - Producing a token on the outgoing arc of the respective task (if there is any)
4) Set the state reached as the *current state* of the process model
5) If the current event is not the last in the trace, consider the next event in the trace as the current event and continue the replay from step 2 onwards

Limitations Cont.



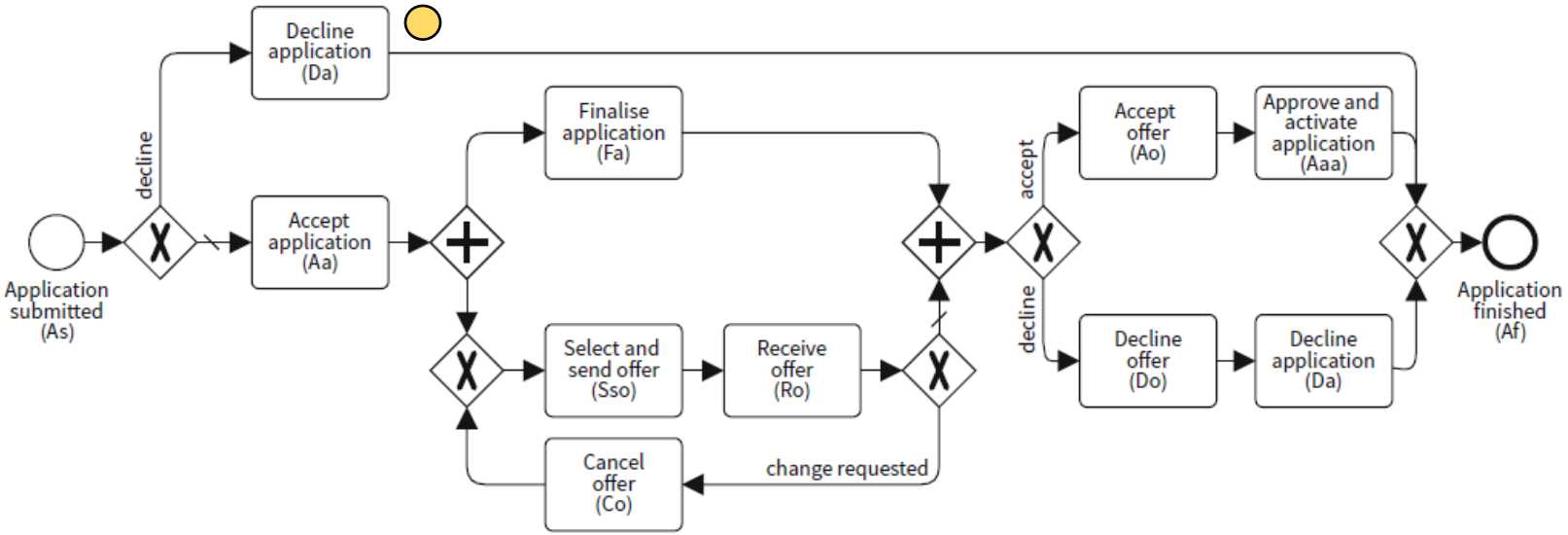
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



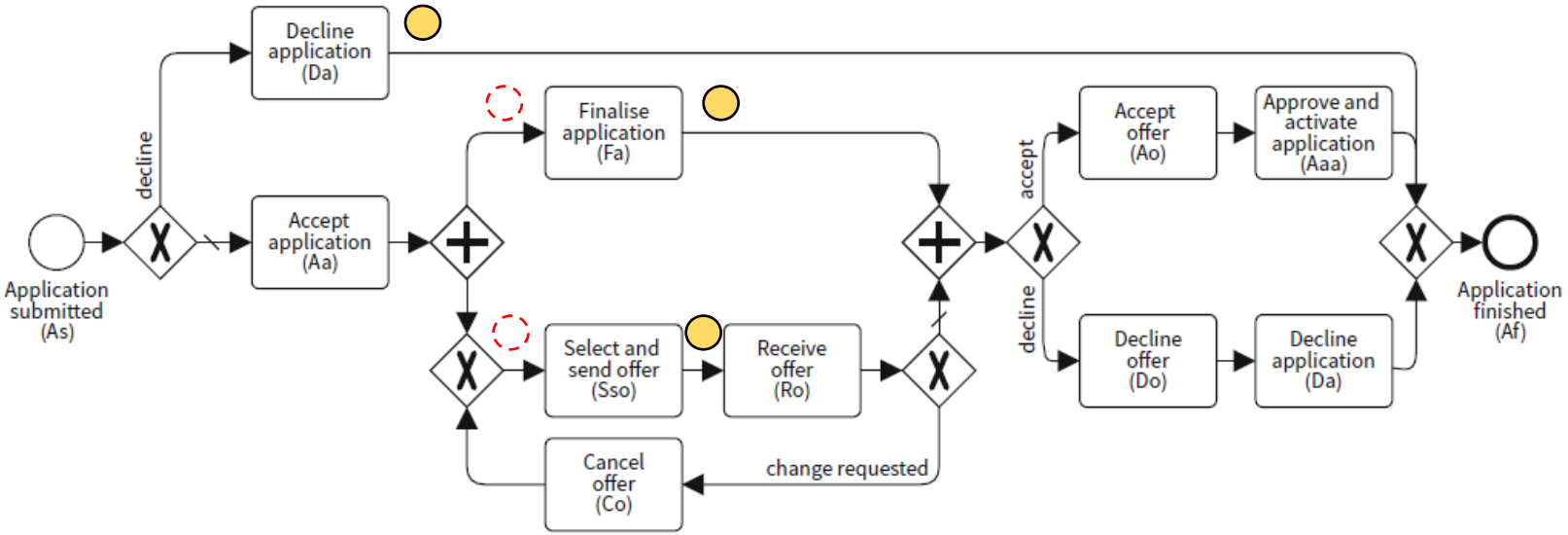
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



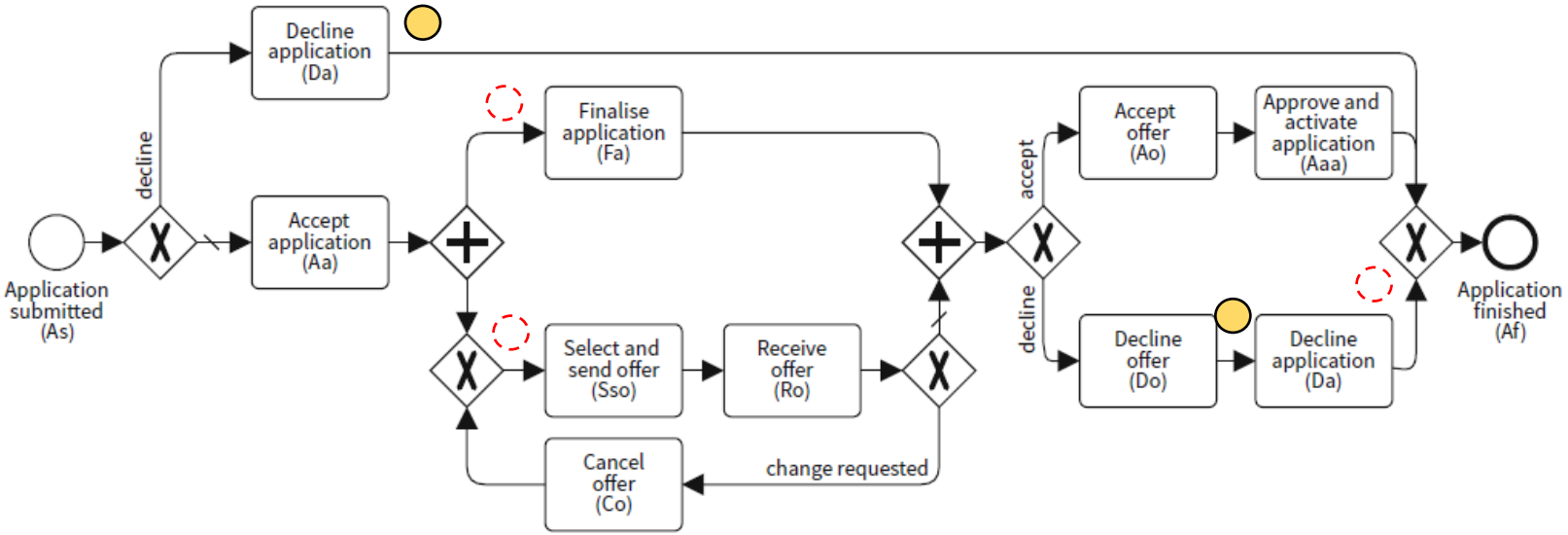
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



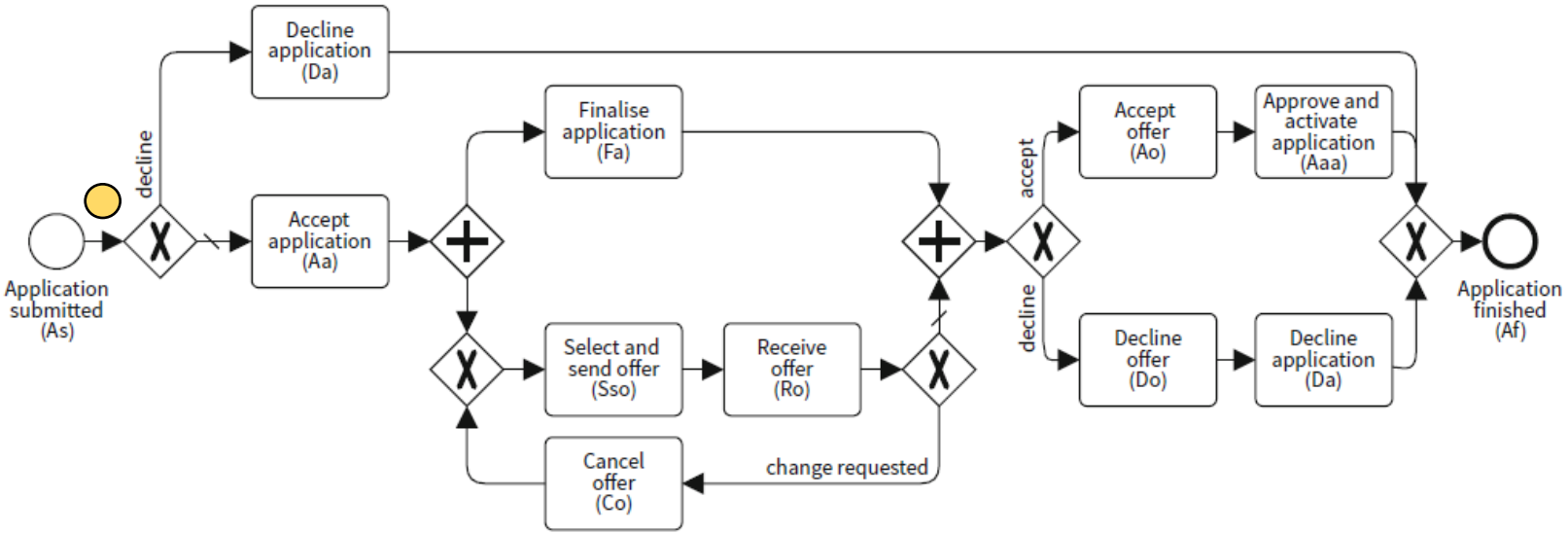
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



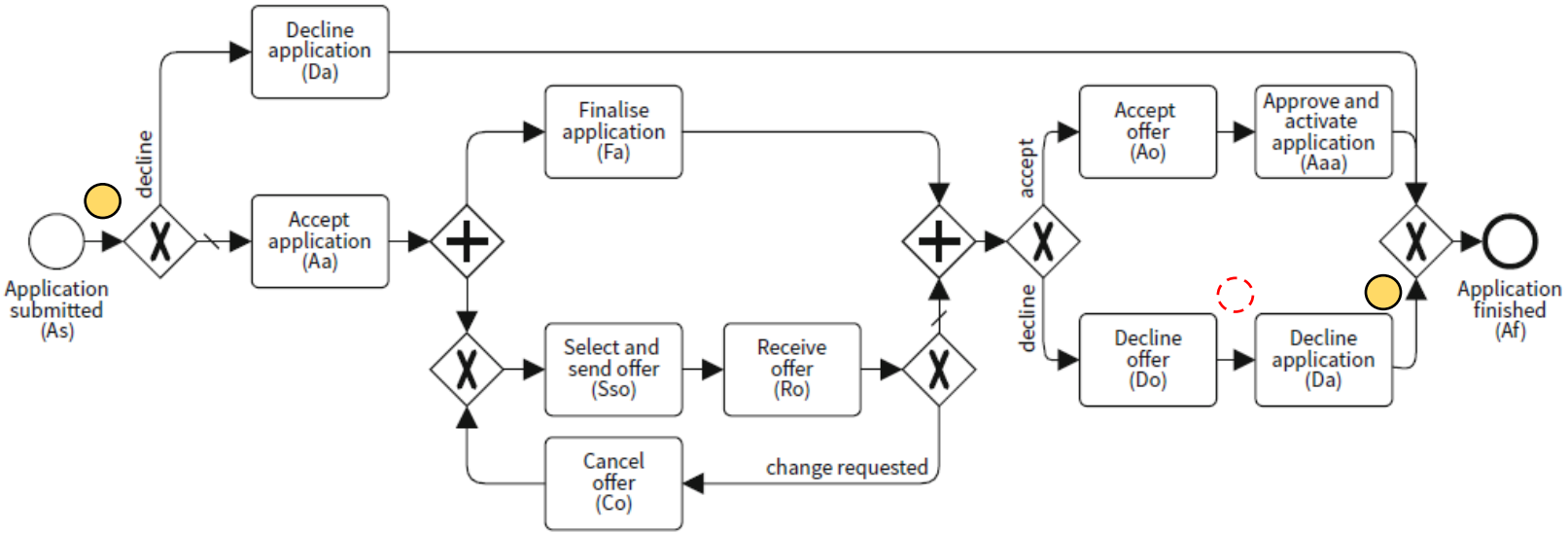
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



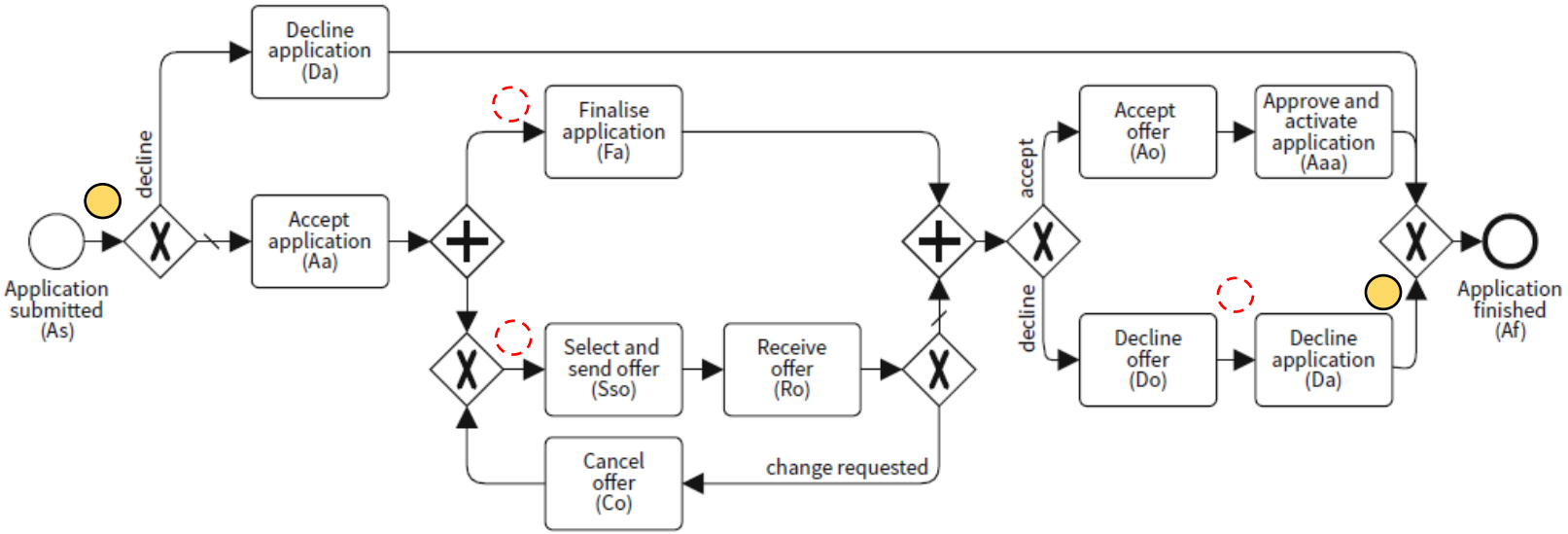
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



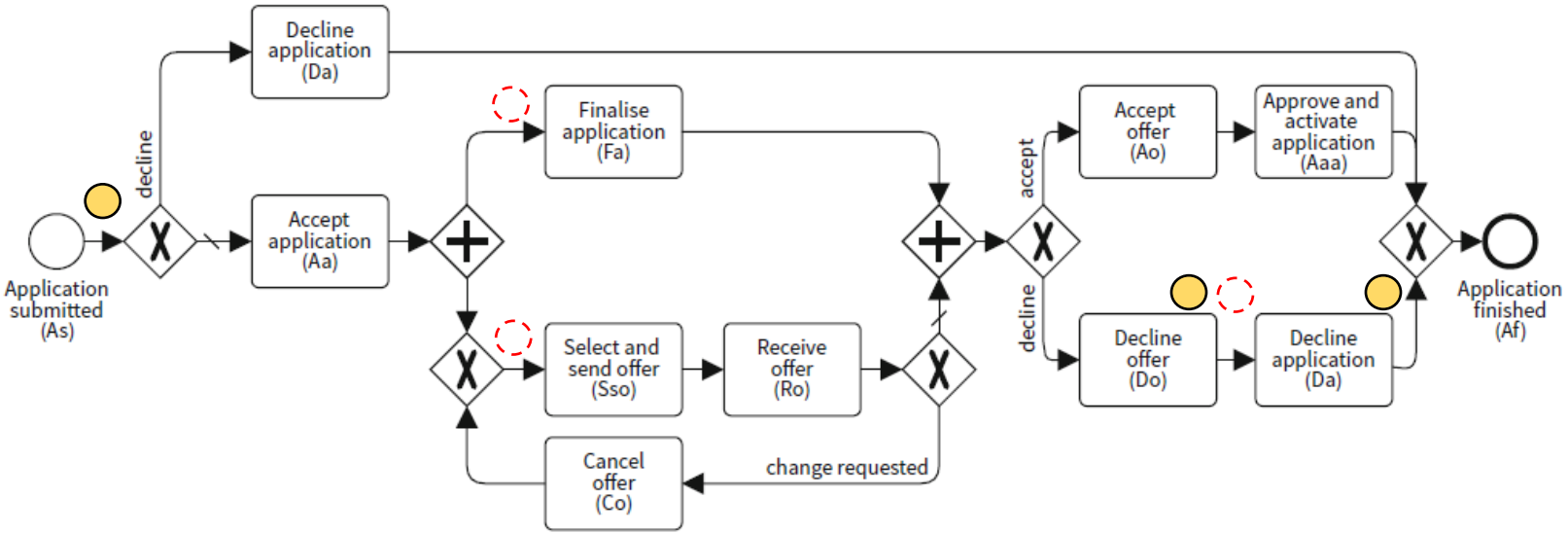
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



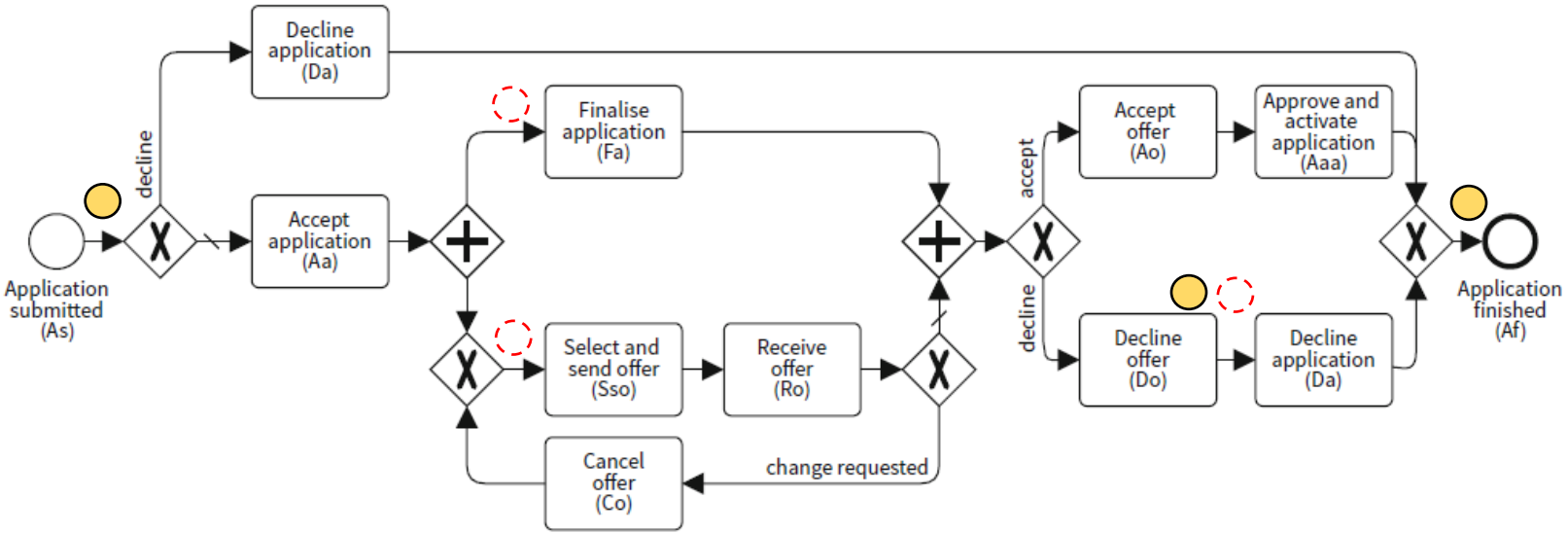
$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.



$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Limitations Cont.

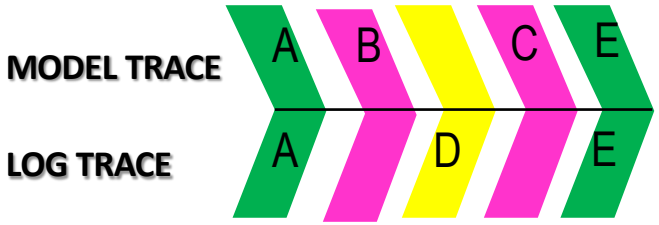


$$T_5 = \langle As, Da, Fa, Sso, Ro, Do, Af \rangle$$

Computing Conformance Checking Artefacts

Alignments (30’)

Conformance Artefact: Alignments



SYNCHRONOUS MOVE: LOG AND MODEL AGREE

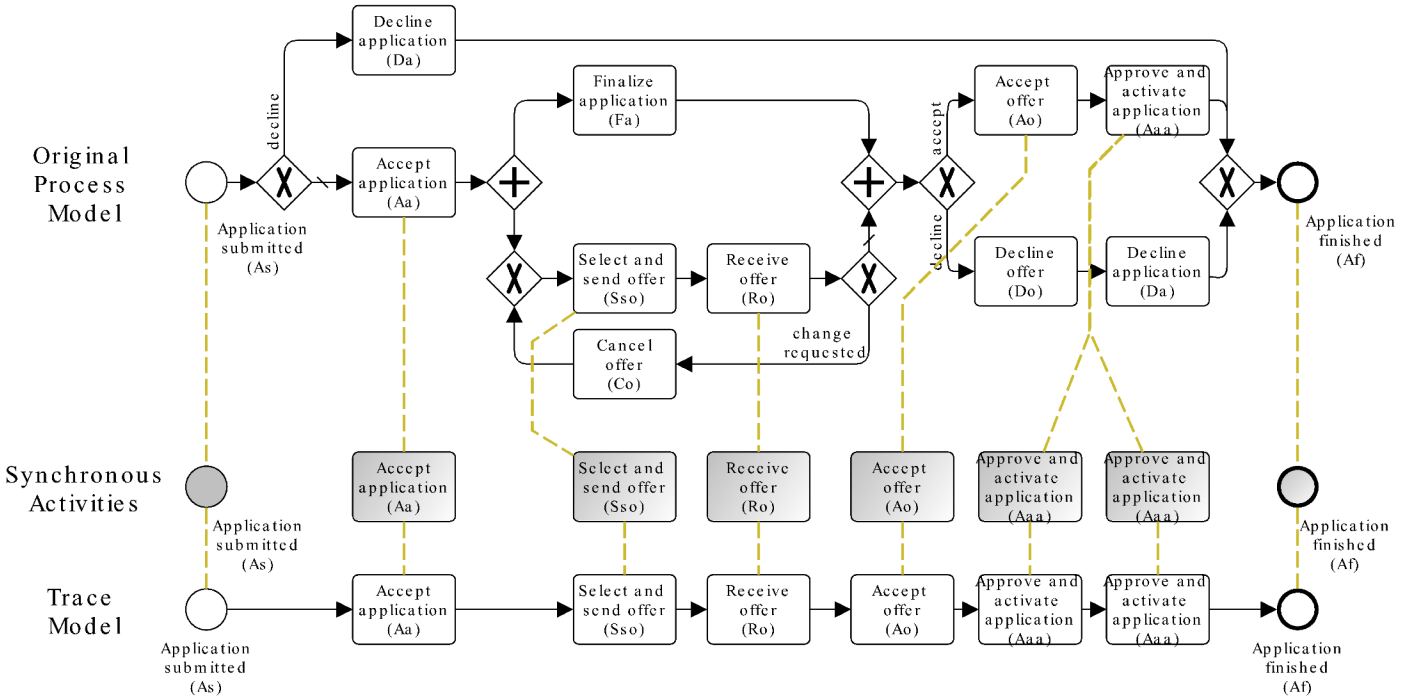
MODEL MOVE: TASKS NOT RECORDED THAT SHOULD BE EXECUTED

LOG MOVE: EVENTS RECORDED THAT CANNOT BE EXECUTED

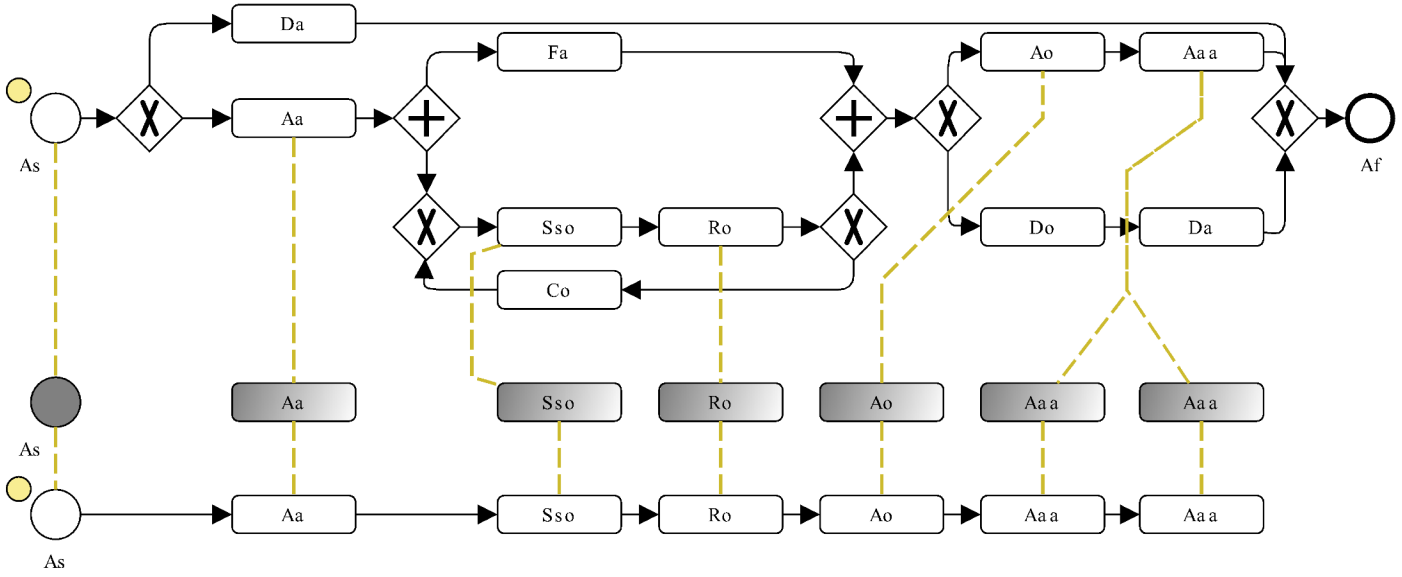
Computing Alignments through A*

- Reference technique for computing alignments
- Can guarantee optimality for an arbitrary cost function on deviations
- Can be adjusted to find not just one but all optimal alignments
- Implemented in ProM, with many optimizations.
- Based on the notion of *Synchronous Product Net*

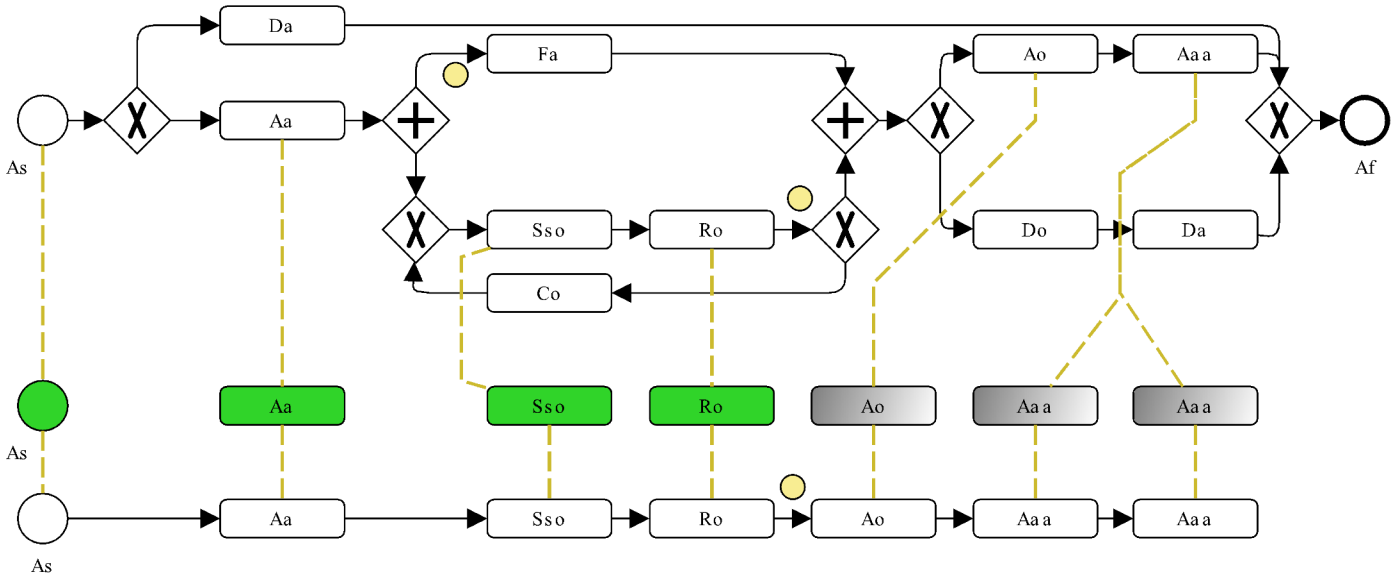
Synchronous Product Net (SPN)



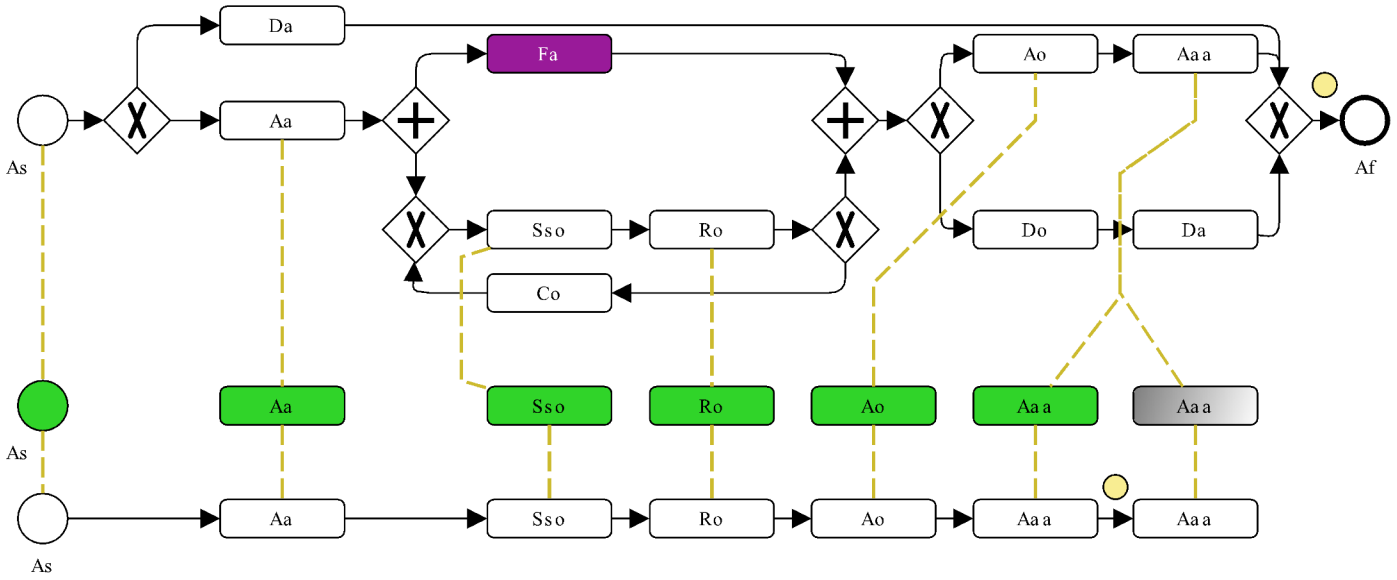
A run of the SPN



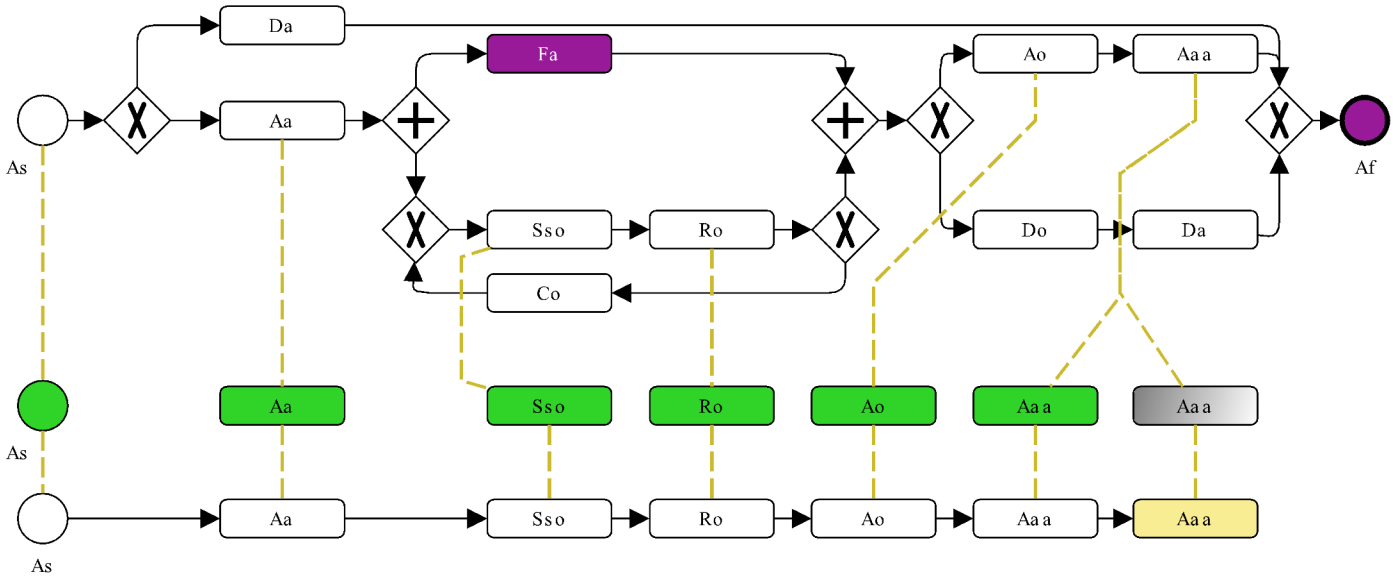
A run of the SPN



A run of the SPN



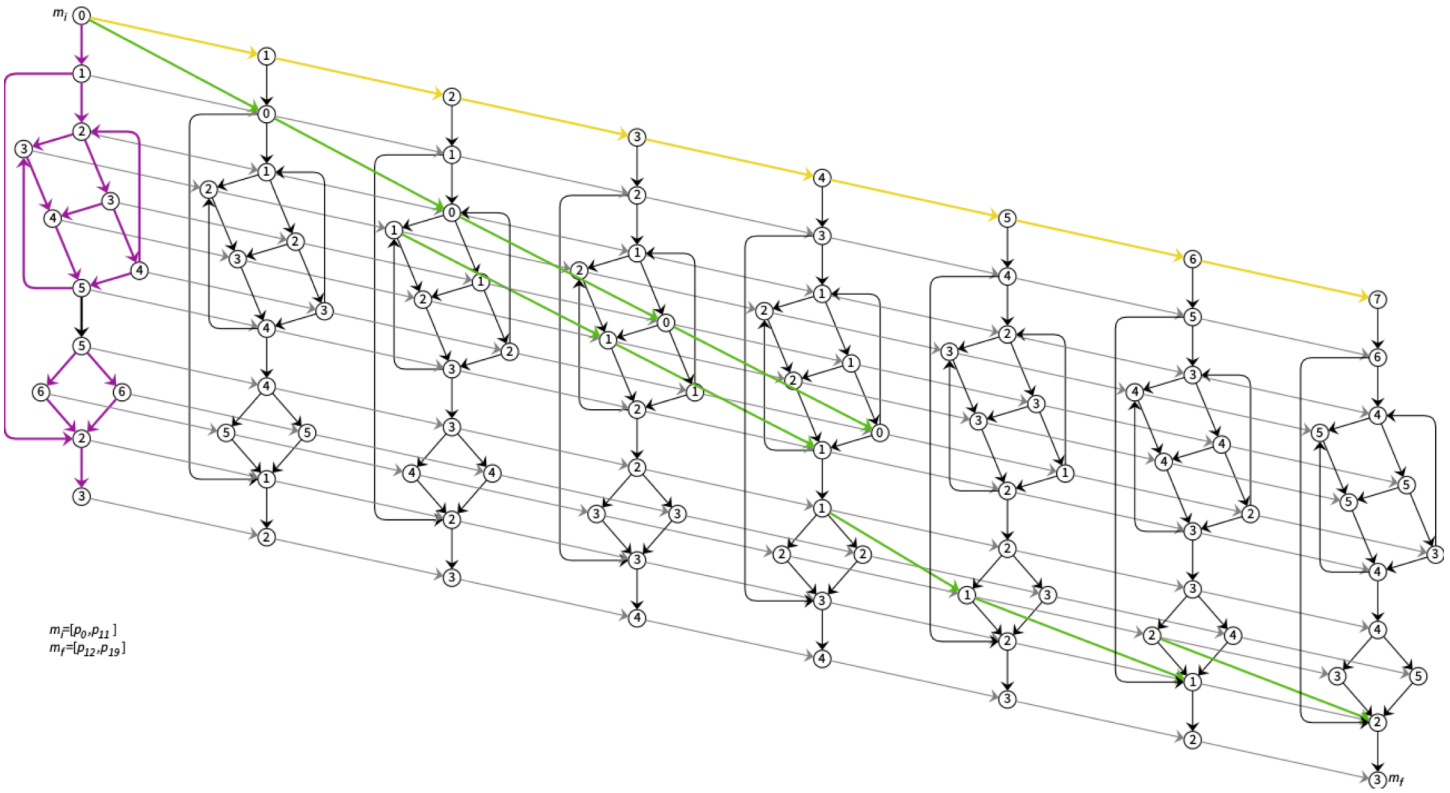
A run of the SPN



Algorithmics Behind the A*

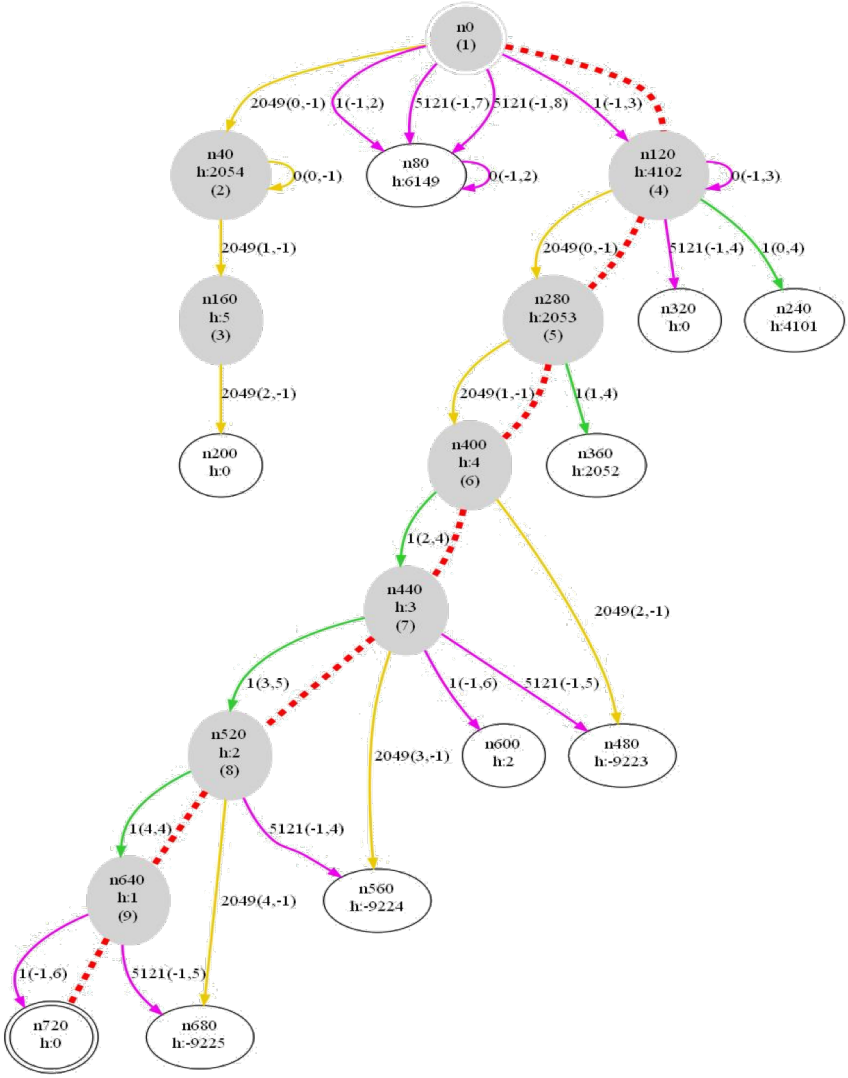
- In general, the SPN has an infinite set of runs !
- Instead: Traverse the SPN **state space** in a efficient way so that a cost-minimal path from the initial to the final state (representing the optimal alignment) is found.
- Using an estimation of the distance to the final state, prune parts of the state space.

SPN State Space



Computing Alignments

- The search space is the statespace of the synchronous product model
- Each node is a combination of a state in the model and the remaining events in the trace
- Each arc is a move on model, move on log or a synchronous move
- A heuristic function estimates the remaining distance to the final node (i.e. model in a final state and all events executed)
- We find a shortest path!



Computing Alignments (A* with fast lowerbound)

```
Initialize HashBackedPriorityQueue q
While (peek(q) is not target t)
  VisitedNode n = head(q)
  If the estimate for node(n) is not exact
    Compute the exact estimate for the remaining distance to t and
    add n to the priority queue
    continue
  add n to the considered nodes
  For each edge in the graph from node(n) to m
    If m was considered before, continue
    If m is in the queue with lower cost, continue
    If m is in the queue with higher cost, update and reposition it
    If m is new,
      compute a fast lowerbound for the remaining distance to t
      v = new VisitedNode(m)
      set n a predecessor for v
      add v to the priority queue
Return head(q)
```

O(log(size q)+α)

?

O(log(size q)+ α)

O(log(size q)+ α)

O(1)

O(log(size q)+ α)

Estimating remaining distance I

- Naïve estimation: 0 Trivial
- LP-based estimation: "Polynomial"
  - Minimize **c.x**
    Where **A.x = r**
    c.x ≥ c.x′
- ILP-based estimation: Exponential
- Hybrid ILP (1 sec for "I" part) Exponential
- Caching LP basis solutions Exponential

Little to no difference in practice

Parikh Vector

The Parikh vector as a basic concept of sequences:

- Given a sequence of elements of some alphabet Σ, the Parikh vector of a sequence $\sigma \in \Sigma^*$ over this alphabet is a function $\overrightarrow{\cdot} : \Sigma^* \to \mathbb{N}$ resulting in a vector of the cardinalities of all elements in σ
- An implicitly assumed order of elements in Σ determines the order in the vector

Example in our context with alphabet $\Sigma = \{A, B, C, D\}$:

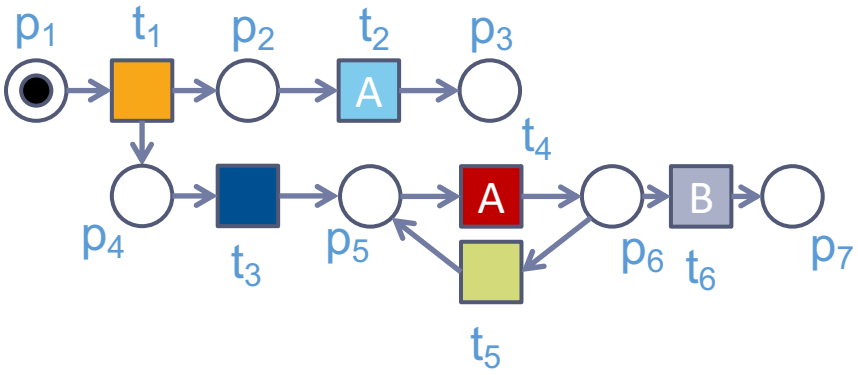
- Trace $\sigma = \langle A, A, B, A \rangle : \vec{\sigma} = [3, 1, 0, 0]^T$
- Same for firing sequence of net system

Incidence Matrix

Represent the structure of a Petri net (P,T,F) in terms of the flow relation as a matrix

- Matrix of |P| rows and |T| columns
- Values of −1 and 1 indicate the presence of a flow arc from a place to a transition, or from a transition to a place, respectively

Example:



	t_1	t_2	t_3	t_4	t_5	t_6
p_1	-1					
p_2	1	-1				
p_3		1				
p_4	1		-1			
p_5			1	-1	1	
p_6				1	-1	-1
p_7						1

Marking Equation

The Marking Equation of a Petri net system

- Establishes link between the net structure and reachability of markings
- Provides a necessary, yet not sufficient condition for reachability of markings

Intuition:

- Let (N,M_i) with $N = (P,T,F)$ be a Petri net system, $p \in P$ a place, and $\bullet p=\{x_1,\ldots,x_n\}$ and $p\bullet=\{y_1,\ldots,y_m\}$
- Then, for any firing sequence σ starting in M_i holds:
  $M(p) = M_i(p) + \vec{\sigma}(x_1) + \ldots + \vec{\sigma}(x_n) - \vec{\sigma}(y_1) - \ldots - \vec{\sigma}(y_n)$
- Generalising this:
  $M(p) = M_i(p) + \sum_{x \in \cdot p} \vec{\sigma}(x) - + \sum_{y \in p\cdot} \vec{\sigma}(y)$

Marking Equation Cont.

Considering all places of a Petri net system, we get the Marking Equation:

$$\overrightarrow{\mathrm{M}} = \overrightarrow{\mathrm{M}_i} + C \cdot \vec{\sigma}$$

where $\overrightarrow{\mathrm{M}}$ and $\overrightarrow{\mathrm{M}_i}$ are place vectors of the respective markings and C is the incidence matrix.

Markings are non-negative, so that:

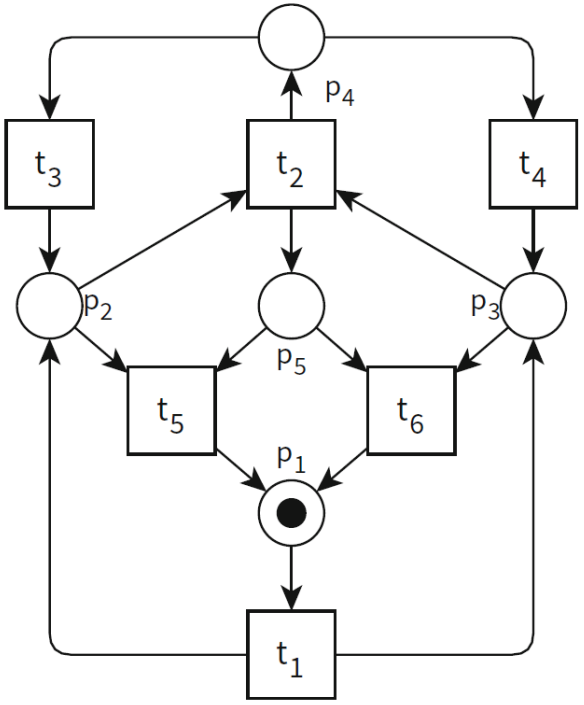
$$\overrightarrow{\mathrm{M}} = \overrightarrow{\mathrm{M}_i} + C \cdot \vec{\sigma} \geq \vec{0}$$

The above yields a necessary, but not sufficient conditions for reachable markings

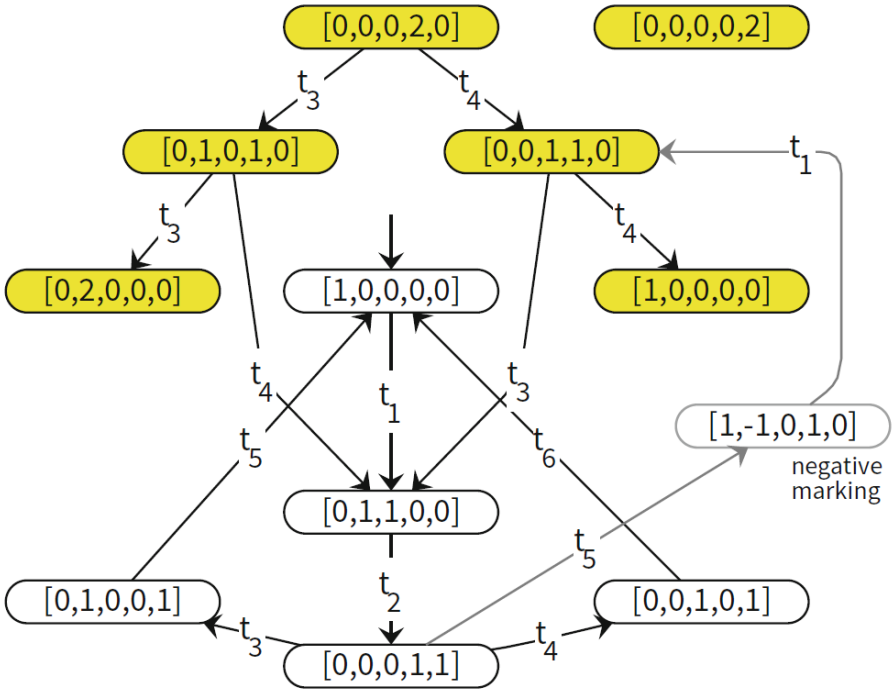
- All reachable markings satisfy the above inequality
- But, finding a marking that satisfies it, does not mean that the marking is reachable

All markings satisfying the inequality are *potentially reachable*

Marking Equation Example



Potentially reachable markings
(highlighted are not reachable):



Consider a firing sequence:

$\langle t_1, t_2, t_5, t_1, t_3 \rangle$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Another Heuristic for A*

Idea:

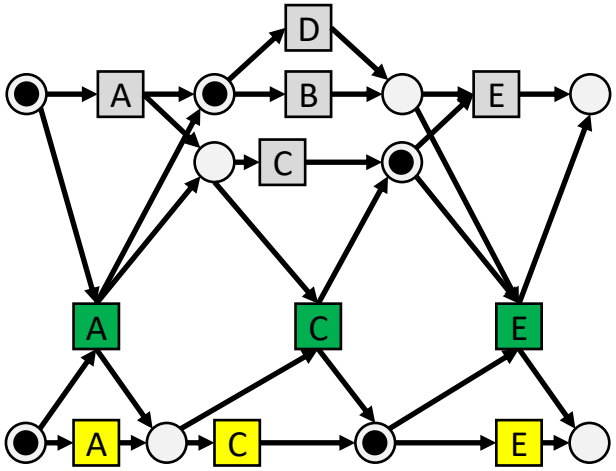
- Sequence of transition firings in the product state space yields an integer solution to the marking equation
- Cost function over this sequence is the alignment cost
- If we simply solve the linear equation system of the marking equation, while minimizing cost, we get a solution that cannot have higher costs than the actual path

With M as some marking and $\mathrm{M_f}$ as the final marking of the product state space, the heuristic is defined as ($\delta(\vec{\mathrm{x}})$ being the aggregated cost of the non-zero transitions in $\vec{\mathrm{x}}$) :

$$\mathrm{h(M)} \begin{cases} \infty & \text{if there is no solution to } \overrightarrow{\mathrm{M}} + \mathrm{C} \cdot \vec{\mathrm{x}} = \overrightarrow{\mathrm{M_f}} \\ \delta(\vec{\mathrm{x}}) & \text{where } \vec{\mathrm{x}} \text{ is a solution to } \overrightarrow{\mathrm{M}} + \mathrm{C} \cdot \vec{\mathrm{x}} = \overrightarrow{\mathrm{M_f}} \text{ and } \vec{\mathrm{x}} \geq \vec{0} \\ & \text{with } \delta(\vec{\mathrm{x}}) \text{ being mininal} \end{cases}$$

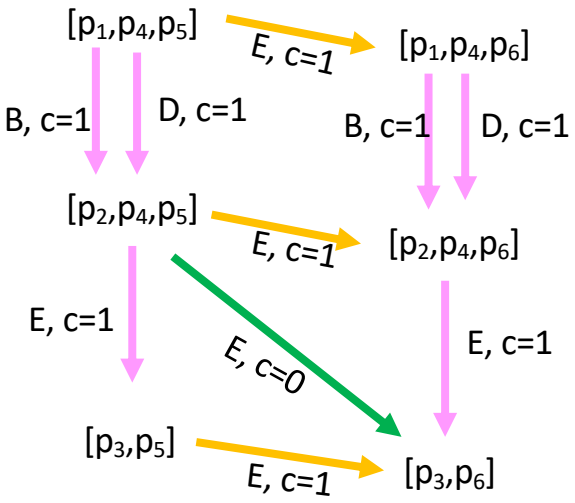
Estimating remaining distance II

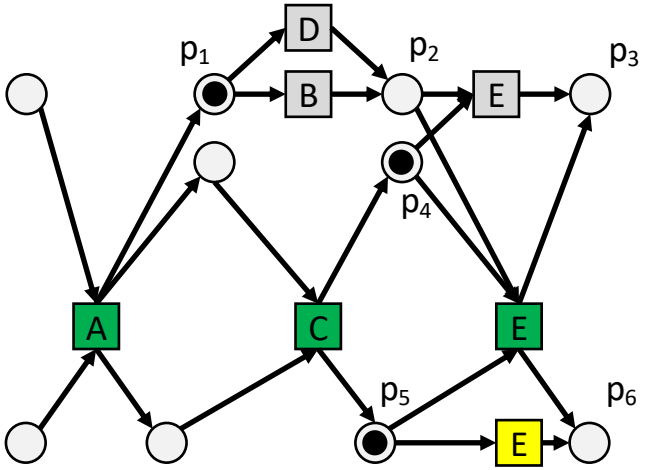
- Consider the model on the right
- And the trace <A,C,E>
- Make the synchronous product
- We reach the marking after A and C



Estimating remaining distance III

What is the remaining cost of getting from $[p_1,p_4,p_5]$ to $[p_3, p_6]$

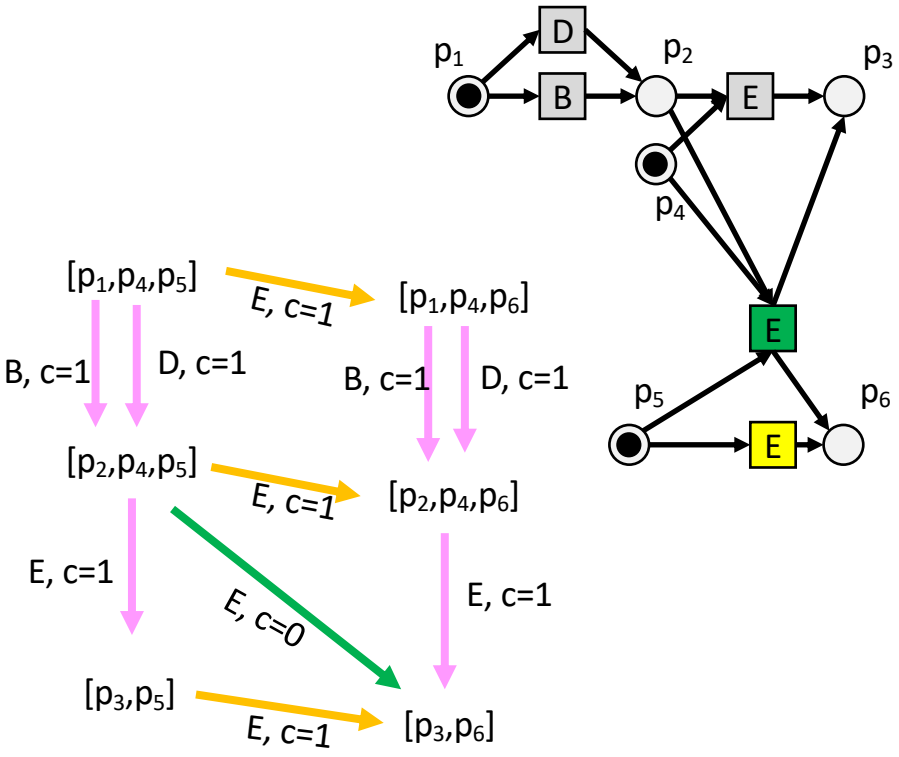




Estimating remaining distance IV

Estimators:

- Dijkstra: 0
- State-equation: 1
  solution: {(B,>>)(E,E)}
- State-equation: 1
  solution: {$^1/_2$(B,>>), $^1/_2$(D,>>),(E,E)}



Implementations: Estimator Versions

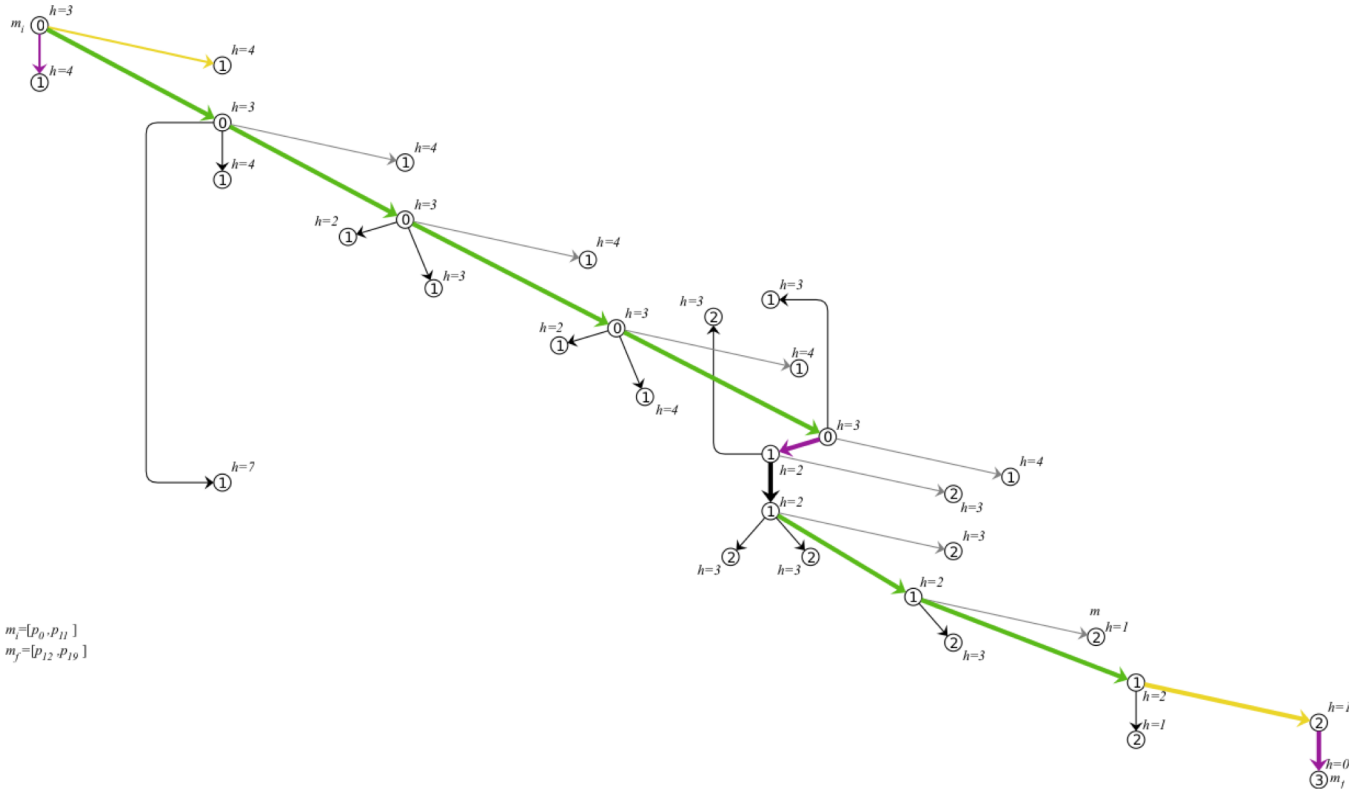
Petri nets:

- Dijkstra (estimator 0)
- Naive (estimator parikh)
- ILP (estimator using ILP)
- Basis caching
- Fast lowerbounds

Process Trees:

- Naive (estimator parikh)
- LP (estimator using LP)
- Hybrid ILP (ILP & LP)
- Special constraints for OR
- Fast lowerbounds
- Basis caching within LP
- Statespace reduction (stubborn sets)

SPN State Space after A* Prunning

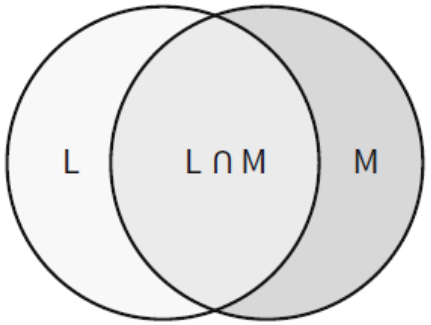


Alignment-based Measures

Alignments enable the quantification of the relation between an event log and a process model

In the context of conformance checking:
Fitness between a log and a model is most relevant

- Assume that the model is normative
- Quantify the extent of deviations of a trace from a model



However, alignments further enable the definition of measures to assess the precision-generalisation trade-off

Fitness Measures

Absolute fitness of a trace σ regarding a given model

- Defined by the cost δ of the optimal alignment of σ with the model
- Using the standard cost function, this corresponds to the number of moves in log and moves in model

Absolute fitness of a multiset $\mathrm{S} = [\sigma_1^{n_1}, \dots, \sigma_m^{n_m}]$ of traces regarding model M with complete execution sequences Π

- Sum of costs of optimal alignments of traces

$$\mathrm{fcost}(\mathrm{S}, \Pi) = \sum_{1 \le i \le m} \mathrm{n_i}\ \delta(\gamma^*(\sigma_i, \Pi))$$

Normalised Fitness

Various solutions to normalise the absolute fitness measure

Here: Division by maximal possible value

- Assume that a move in both (x, y) happens only if $x = y$
- Worst case: alignment is built only from moves in model and moves in log
- With S as the multiset of traces, the cost of moving through the log without moving in the model is:

$$\mathrm{c}(\mathrm{S}) = \sum_{1 \le i \le m} \mathrm{n_i} \sum_{x \in \sigma_i} \delta(x, \bot)$$

- The cost of moving only in the model (via the least expensive path) is:

$$\mathrm{c}(\Pi) = \min_{\pi \in \Pi} \sum_{y \in \pi} \delta(\bot, y)$$

Normalised fitness measure:

$$\mathrm{fitness}(\mathrm{S}, \Pi) = 1 - \frac{\mathrm{fcost}(\mathrm{S}, \Pi)}{c(\mathrm{S}) + \sum_{1 \le i \le m} \mathrm{n_i}\, \mathrm{c}(\Pi)}$$

Assess Precision

Quantify the behaviour of the model not seen in the log

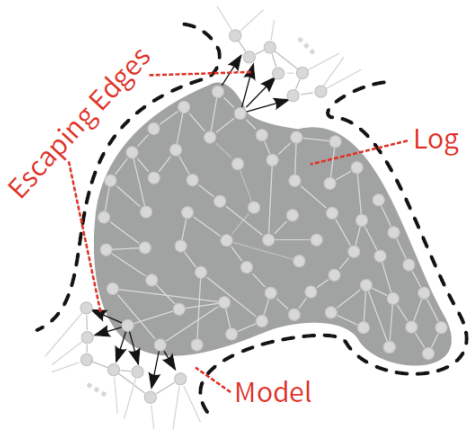
Assumption: All traces fit the model

Assumption: Model is deterministic

- In any state, there cannot be two transitions with the same label
- Non-deterministic models can be determinised (before computing an alignment)

General idea

- Based on traces of the log, characterise states in the model with “escaping edges”
- Those are transitions representing behaviour that immediately follows behaviour of the log but is not contained in the log



Precision Measure

Determine the language of "escaping edges", denoted L_{esc}

- It contains all sequences of activity labels $\sigma \cdot \langle a \rangle$, such that
  - σ is a prefix of a trace in the log and a prefix of an execution sequence of the model
  - $\sigma \cdot \langle a \rangle$ is not a prefix of a trace in the log, but still a prefix of an execution sequence of the model
- Note: Since both the log and the set of activity labels are finite, so is the language of escaping edges

Precision of a log as a set of traces $\mathrm{L} = [\sigma_1 \ , \dots, \sigma_m]$ and a model M with execution sequences Π:

$$\text{precision} = \frac{|\ \text{Prefix}(\mathrm{L} \cap \Pi)|}{|\ \text{Prefix}(\mathrm{L} \cap \Pi)| + \ |L_{esc}|}$$

where Prefix(X) is the set of all prefixes of language X.

Alignment-based Precision Measure

Alignments enable us to drop the assumption of fitting traces in the computation of precision

- Intuitively, the alignment gives an execution sequence best resembling an unfitting trace
- This execution sequence is used when building the joint language
- Instead of relying on $\mathrm{L} \cap \Pi$, we rely on $\mathrm{L}_{\mathbf{align}} = \{ \gamma^*(\sigma, \Pi)_{|2} \mid \sigma \in \mathrm{L} \}$ where $\gamma^*{}_{|2}$ is the projection of the alignment on the second component (i.e., the execution sequence)

Then, redefine the language of "escaping edges", denoted L_{esc} **based on alignments:**

- It contains all sequences of activity labels $\sigma \cdot \langle a \rangle$, such that
  - σ is a prefix of an element of $\mathrm{L}_{\mathbf{align}}$
  - $\sigma \cdot \langle a \rangle$ is not a prefix of a trace in the log, but still an element of $\mathrm{L}_{\mathbf{align}}$

Precision is then measured as :

$$\mathrm{precision} = \frac{|\ \mathrm{Prefix}(\mathrm{L}_{\mathrm{align}})|}{|\ \mathrm{Prefix}(\mathrm{L}_{\mathbf{align}})| + \ |\mathrm{L}_{esc}|}$$

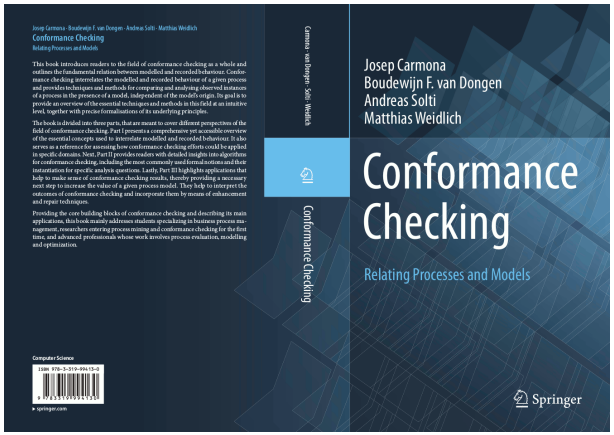
"Conformance Checking: Relating Processes and Models"

Josep Carmona

Boudewijn van Dongen

Andreas Solti

Matthias Weidlich



https://www.springer.com/gb/book/9783319994130

Springer

Take Away

Conformance checking relates observed and modeled behavior

Algorithmic alternatives: rule-checking, token-replay and alignments

Crucial to asses the quality of process models

But also crucial for improving process models based on reality (next weeks)