

# Distributed Dynamic Condition Response Structures

Workshop on Timed and Infinite Systems  
30th March, 2010

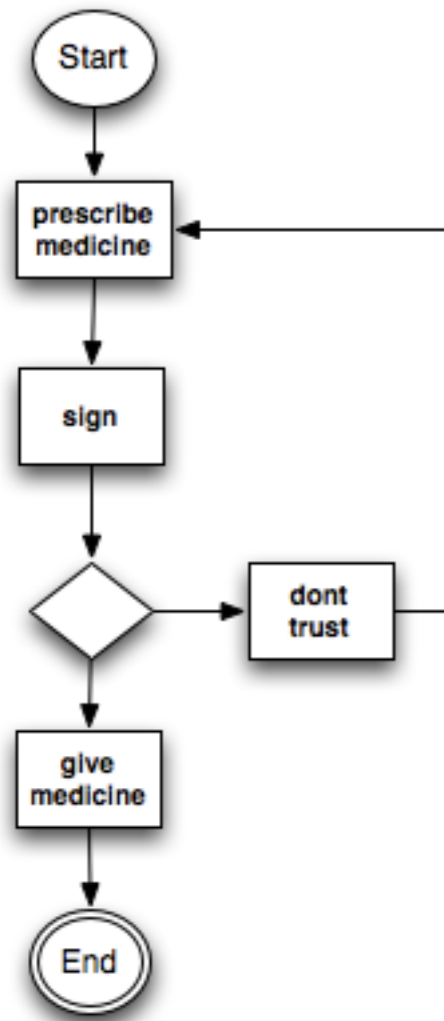
Thomas Hildebrandt and Raghava Rao Mukkamala  
TrustCare Project ([www.trustcare.eu](http://www.trustcare.eu))  
IT University of Copenhagen, Denmark

# Overview

- Motivation
- Event Structures
- Distributed Dynamic Condition Response Structures
- DCRS graphical notation
- Accepting Condition for finite and infinite runs
- Conclusion and Future Work

# Motivation

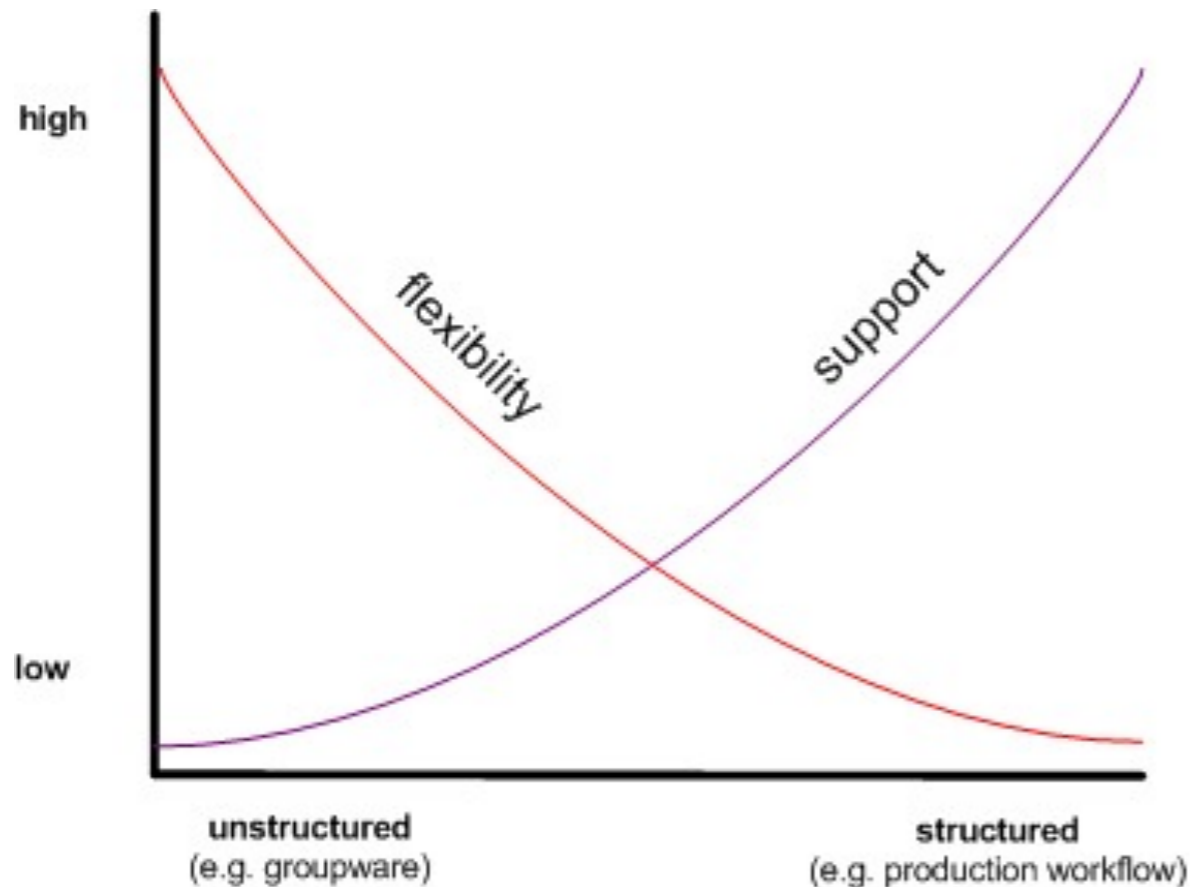
## Example: Health Care Workflow in Flow Chart



- OK
  - ✓ [pm, s, gm]
  - ✓ [pm, s, dt, pm, s, gm]
- Missing
  - ⊙ [pm, pm, s, gm]
  - ⊙ [pm, s, pm, s, gm]
  - ⊙ [pm, s, gm, gm, pm, s, gm]
- The given flow chart is rigid.
  - possibility of adding several prescriptions before and after signing is missing

# Motivation

## Flexibility versus Support in workflows



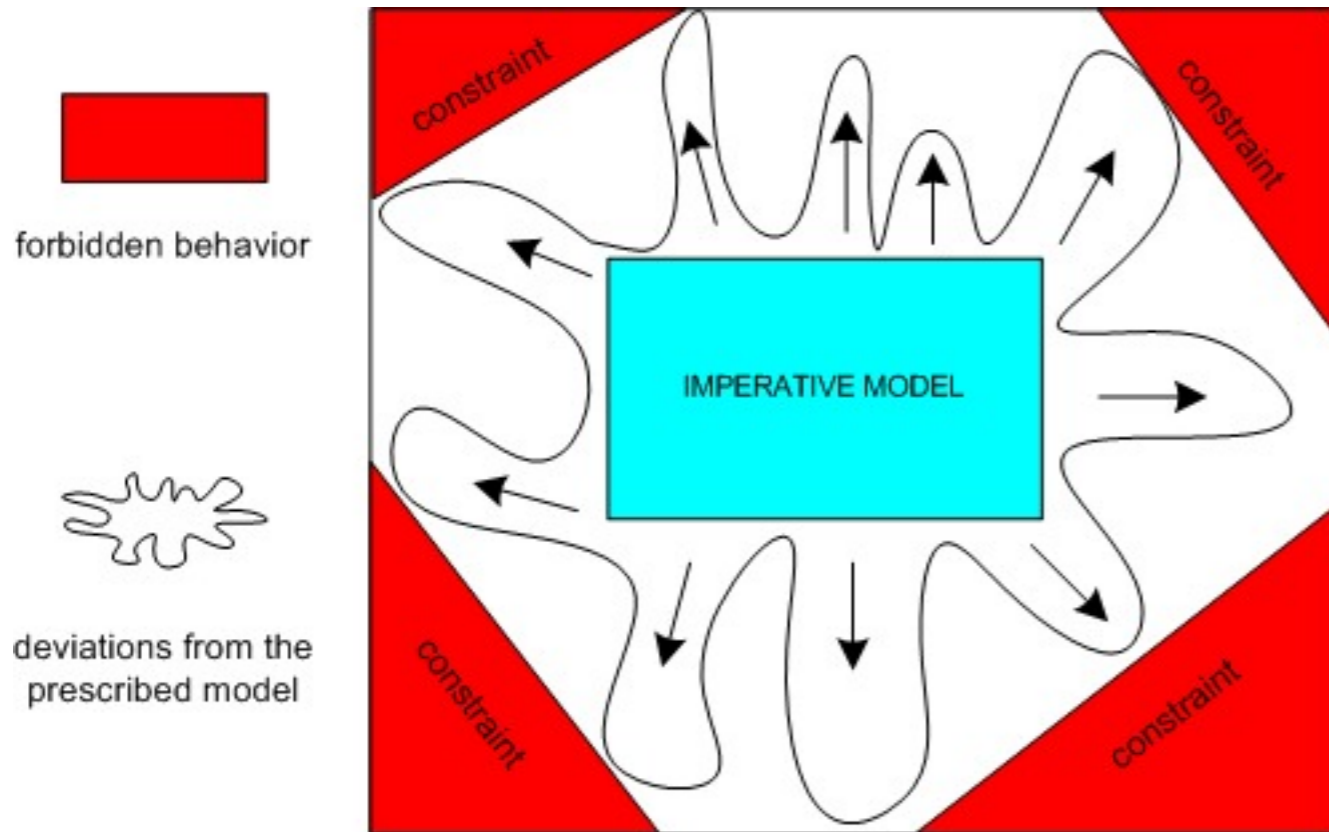
Classical trade-off between flexibility and support!

- Flexibility: ability to
  - defer: decide to decide later
  - change: decide to change model
  - deviate: decide to ignore model
- support: provide analysis and guidance
- unstructured: do what ever you want, but get no support (ex: email programs)
- structured: support, but no flexibility (traditional workflows)

[1] W.M.P. van der Aalst et al. Declarative workflows: Balancing between flexibility and support

# Motivation

## Imperative versus Declarative models



Imperative versus declarative modeling languages<sup>2</sup>

- Imperative:
  - explicitly specify the control flow
  - over-constrain the control flow
  - adding a new constraint requires to make a new model
- Declarative:
  - implicitly specify the control flow
  - specify constraints to forbid the unwanted behavior
  - difficult to perceive what are the next possible actions
  - difficult to get an overview about how to get to the end.

[2] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes Management

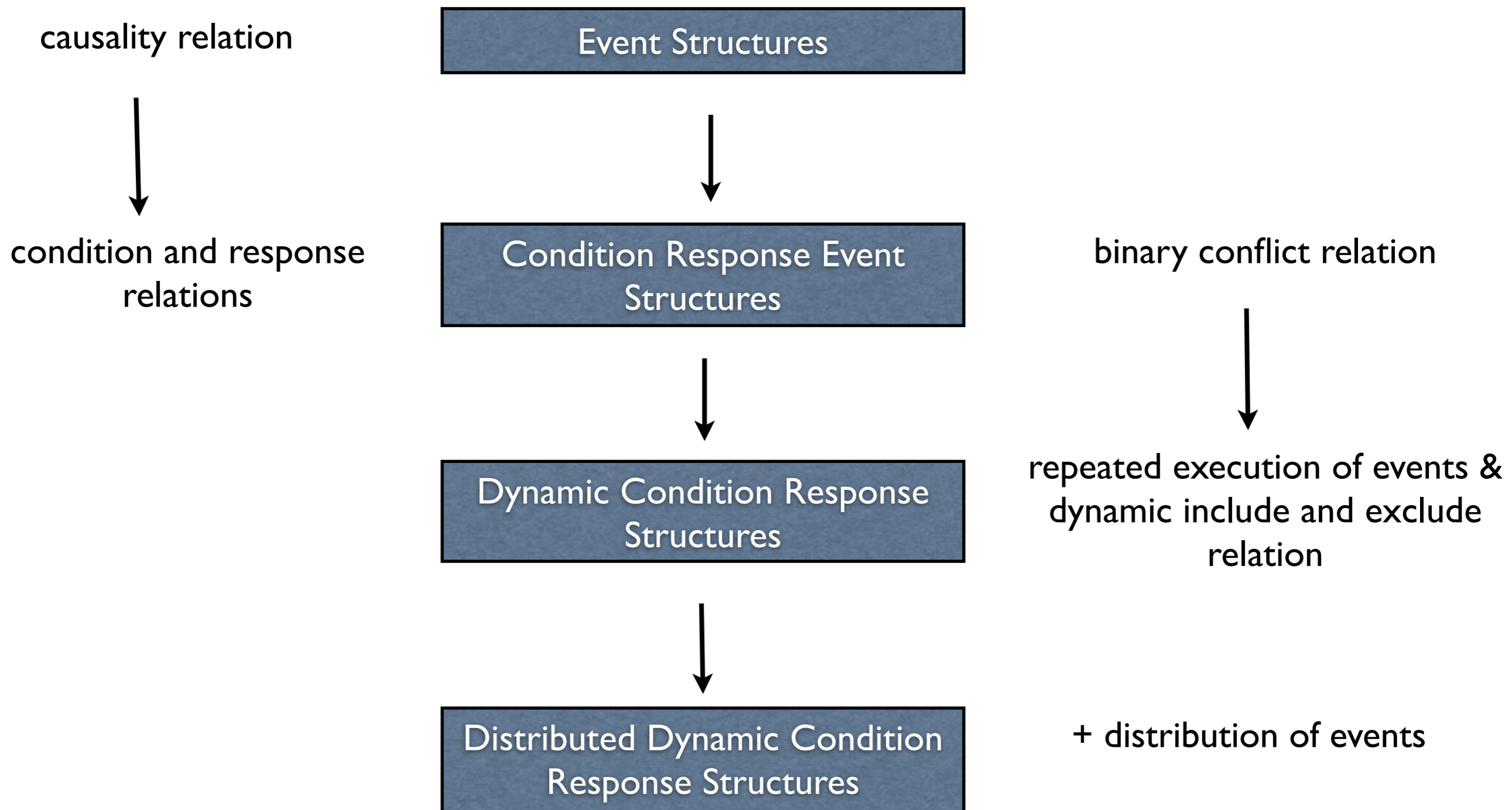
# Goal

- Develop a formal process model which makes it easy
  - to describe loosely constrained workflows in an incremental way
  - to understand and to execute
  - to serve as basis for well founded typed workflow language

## Related Work

- Declare: declarative workflow language
  - W. M. P. van der Aalst, M. Pesic and H. Schonenberg: Declarative workflows: Balancing between flexibility and support. (2009).
  - W.M.P. van der Aalst and M. Pesic: DecSerFlow: Towards a Truly Declarative Service Flow Language. (2006)
- Event Calculus
  - Nihan Kesim Cicekli and Ilyas Cicekli. Formalizing the specification and execution of workflows using the event calculus. (2006)

# Overview of Dynamic Condition Response Structures



# Event Structures

## Definition

A labeled *prime event structure* (ES) over an alphabet *Act* is a 4-tuple  $(E, \leq, \#, l)$  where

- ▶  $E$  is a (possibly infinite) set of events
- ▶  $\leq \subseteq E \times E$  is the causality relation between events which is partial order
- ▶  $\# \subseteq E \times E$  is a binary conflict relation between events which is irreflexive and symmetric
- ▶  $l : E \rightarrow Act$  is the labeling function mapping events to actions

1. Causality relation satisfies principle of finite causes:  
 $\forall e \in E : \{e' \in E \mid e' \leq e\}$  is finite.
2. Conflict relation satisfies principle of conflict heredity:  
 $\forall e, e', e'' \in E. e \# e' \wedge e \leq e'' \Rightarrow e' \# e''$
3. A run is a sequence  
 $e_0, e_1, \dots, e_n$  such that,  $e_i \downarrow \leq \{e_0, \dots, e_{i-1}\}$  for  $\forall i, j. \neg(e_i \# e_j)$

# Event Structures

- Events structures are missing
  - finite representation of infinite behavior
  - accepting condition
  - distribution of events

# Example in Event Structures

## Example: Health Care Workflow in Event Structures

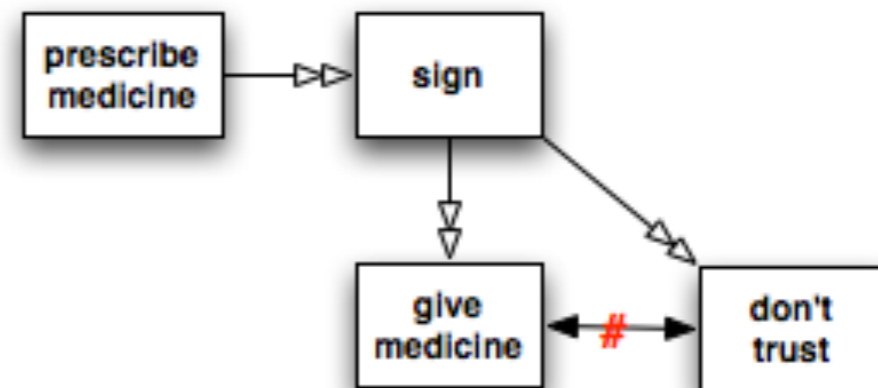
Events  $E = \{pm, s, gm, dt\}$

Causality Relation :  $pm \leq s \leq gm, s \leq dt$

Conflict Relation:  $gm \# dt$

Possible Runs

- [pm, s, gm]
- [pm, s, dt]
- [pm, s]
- [pm]



Causality Relation  $\longrightarrow$

Conflict Relation  $\longleftrightarrow$

How can we make sure that every time when *pm* happens eventually *gm* also happens?

# Condition Response Event Structures(CRES)

## Definition

A labeled *condition response event structure* (CRES) over an alphabet  $Act$  is a tuple  $(E, \leq_C, \leq_R, \#, l)$  where

- ▶  $\leq_C \subseteq E \times E$  is the *condition* relation between events which is partial order
- ▶  $\leq_R \subseteq E \times E$  is the *response* relation between events, satisfying that  $\leq = \leq_C \cup \leq_R$  is a acyclic relation
- ▶  $E, \#, l$  are same as Event Structures

## Runs and accepting conditions

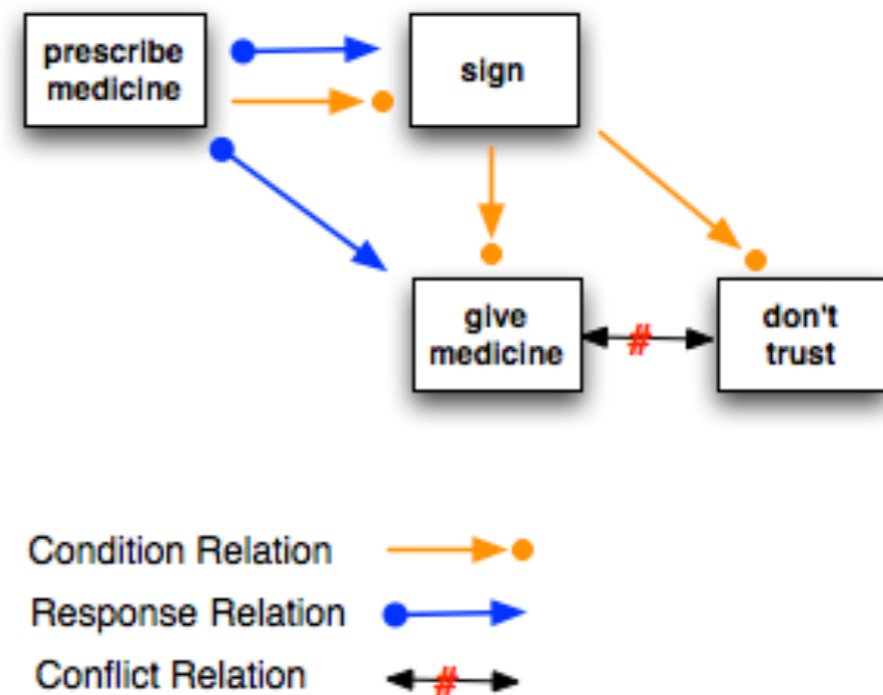
1. if  $e \leq_C e'$  then  $e$  must happen before  $e'$
2. if  $e \leq_R e'$  then after  $e$  happens,  $e'$  must eventually happen or become in conflict for the computation to be accepting.

# Condition Response Event Structures - 2

## Proposition

The labelled prime event structure  $(E, \leq, \#, I, \text{Act})$  has the same runs as the accepting runs of the CRES structure  $(E, \text{Act}, \leq_C, \leq_R, \#, I, \text{Act})$  where  $\leq_C = \leq, \leq_R = \emptyset$

- ▶  $E = \{pm, s, gm, dt\}$
- ▶  $\leq_C = \{(pm, s), (s, gm), (s, dt)\}$
- ▶  $\leq_R = \{(pm, s), (pm, gm)\}$
- ▶  $\# = \{(gm, dt)\}$
- ▶ Possible runs
  - ▶  $[pm, s]$  OK, but not accepting
  - ▶  $[pm, s, gm]$  accepting
  - ▶  $[pm, s, dt]$  accepting



How would we be able to re-execute *prescribe medicine* and/or *sign* after *don't trust*?

# Dynamic Condition Response Structures(DCRS)

## Definition

A *dynamic condition response structure* (DCR) is a tuple  $D = (E, Act, \rightarrow\bullet, \bullet\rightarrow, \pm, I)$  where

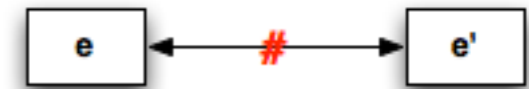
- ▶  $\rightarrow\bullet \subseteq E \times E$  is the *condition* relation
  - ▶  $\bullet\rightarrow \subseteq E \times E$  is the *response* relation
  - ▶  $\pm : E \times E \rightarrow \{+, \%, *\}$  is the *dynamic inclusion/exclusion* relation.
  - ▶ rest are same as Condition Response Event Structures
1. condition relation in DCRS ( $\rightarrow\bullet$ ) is same as condition relation in CRES ( $\leq_C$ )
  2. response relation in DCRS ( $\bullet\rightarrow$ ) is same as response relation in CRES ( $\leq_R$ )
  3. conflict relation is generalized to include relation + exclude relation to include/exclude events dynamically.
  4.  $\pm(e, e') = +$  if event  $e$  gets executed, then it will include event  $e'$
  5.  $\pm(e, e') = \%$  if event  $e$  gets executed, then it will exclude event  $e'$
  6. conflict relation is monotone (once an event is in conflict it stays in conflict), but dynamic inclusion/exclusion allows an event to alternate between being in conflict and not.
  7. execution of an event only depends on the condition relation restricted to the currently included events.

# Dynamic Condition Response Structures(DCRS)

## Proposition

The condition response event structure  $(E, \leq_C, \leq_R, \#, I, Act)$  has the same accepting runs as the accepting runs of the DCR structure

$(E, Act, \rightarrow\bullet, \bullet\rightarrow, \pm, I)$  where  $\rightarrow\bullet = \leq_C$ ,  $\bullet\rightarrow = \leq_R$ ,  $\forall e, e' \in E. \pm(e, e') = \%$  if  $e = e'$  or  $e\#e'$  and otherwise  $\pm(e, e') = *$ .



# relation in CRES



Encoding of # in DCRS

- ▶  $E = \{pm, s, gm, dt\}$
- ▶  $\rightarrow\bullet = \{(pm, s), (s, gm), (s, dt)\}$
- ▶  $\bullet\rightarrow = \{(pm, s), (pm, gm), (dt, ss)\}$
- ▶  $\rightarrow+ = \{(s, gm), (s, dt)\}$
- ▶  $\rightarrow\% = \{(gm, dt), (dt, gm)\}$
- ▶ Possible runs
  - ▶  $[pm, s, dt, s, gm]$
  - ▶  $[pm, s, dt, pm, s, gm]$
  - ▶  $[pm, s, pm, s, gm]$



# Distributed Dynamic Condition Response Structures

## Definition

A *distributed* dynamic condition response structure (DDCR) is a tuple

$$(E, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, I, R, P, \text{as})$$

where  $(E, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, I)$  is a dynamic condition response structure,  $R$  is a set of *roles*,  $P$  is a set of *principals* (e.g. persons/processors/agents) and  $\text{as} \subseteq (P \cup \text{Act}) \times R$  is the role assignment relation to executors and actions.

- assigning roles to actions provide granularity of permissions
- assigning principals to roles gives the permission to execute actions

# Semantics of DDCRS (Accepting condition for Finite Runs)

## Definition

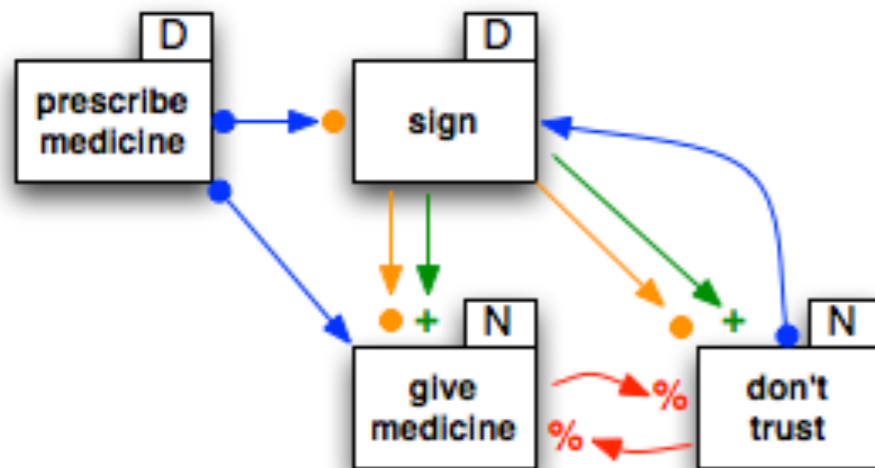
For a distributed DCR  $D = (E, \text{Act}, \rightarrow\bullet, \bullet\rightarrow, \pm, I, R, P, \text{as})$  the corresponding labelled transition systems  $T(D)$  to be the tuple  $(S, (\emptyset, E, \emptyset), \rightarrow\subseteq S \times \text{Act} \times S)$  where  $S = \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E)$  is the set of states,  $(\emptyset, E, \emptyset) \in S$  is the initial state,  $\rightarrow\subseteq S \times (\text{P} \times \text{Act} \times \text{R}) \times S$  is the transition relation given by

$$(E, I, R) \xrightarrow{(e, (p, a, r))} (E \cup \{e\}, I', R') \text{ where}$$

- ▶  $e \in I, I(e) = a, p \text{ as } r, \text{ and } a \text{ as } r$
  - ▶  $\{e' \in I \mid e' \rightarrow\bullet e\} \subseteq E$
  - ▶  $I' = (I \cup \{e' \mid \pm(e, e') = +\}) \setminus \{e' \mid \pm(e, e') = \%\}$
  - ▶  $R' = (R \setminus \{e\}) \cup \{e' \mid e \bullet\rightarrow e'\}$
- map to a labelled transition system to defined accepting runs
  - states of transition system will be  $(E, I, R)$  where  $E \subseteq E$  is set of happened events,  $I \subseteq E$  represents set of currently included events,  $R \subseteq E$  represents set of pending response events
  - first condition says that only currently include events can be executed, the label  $(p, a, r)$  says that the label of event  $e$  must be  $a$ , which must be assigned to a role  $r$  and principal  $p$
  - second condition says that all condition events to  $e$  must have been executed
  - third and fourth conditions are updates to sets of included and pending responses.
  - **Accepting condition: all runs which are ending with states where  $R \cap I = \emptyset$**

# DCRS Graphical Notation

## Example: Health Care Workflow in DDCRS



- OK and Accepting

- ✓ [pm, s, gm, gm]
- ✓ [pm, s, dt, s, gm]
- ✓ [pm, s, dt, pm, s, gm]
- ✓ [pm, pm, s, gm]
- ✓ [pm, s, pm, s, gm]

- Not Possible

- ⊙ [pm, gm]
- ⊙ [pm, s, dt, gm]

- Not Accepting

- ⊙ [pm, s, gm, pm]
- ⊙ [pm, s, dt]

Condition relation is same as precedence relation in DECLARE<sup>2</sup>

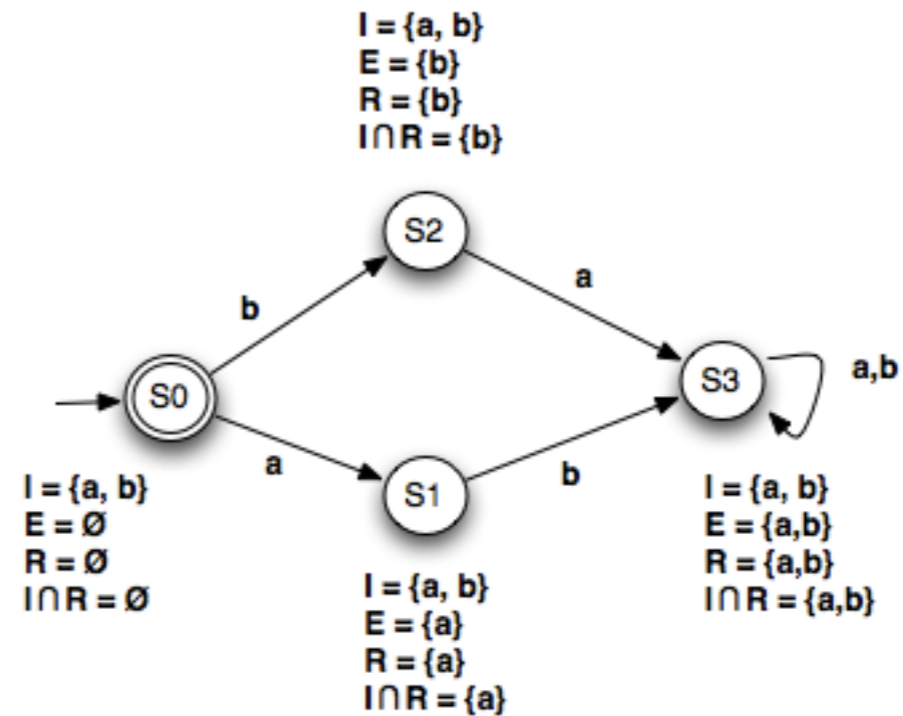
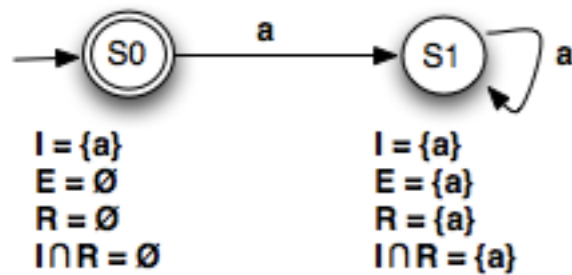
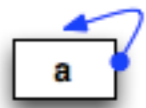
$$\diamond(b) \Rightarrow ((\neg b) \cup a)$$

Response relation is same as that of DECLARE<sup>2</sup>

$$\square(a \Rightarrow \diamond b)$$

[2] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes Management

# DCRS for Infinite Runs - Motivation



- In the first example a run  $a^\omega$  should be accepting.
- In the second example, a run  $a^* b^* (ab)^\omega$  should be accepting.
- But strong accepting condition ( $R \cap I = \emptyset$ ) does not allow those runs accepting.

# (Weaker) Accepting Condition for infinite runs

For a finite distributed DCR  $D = (E, \text{Act}, \rightarrow, \bullet, \bullet \rightarrow, \pm, I, R, P, \text{as})$  where  $E = \{e_1, \dots, e_n\}$  we define the corresponding Büchi-automaton  $\text{Aut}(D)$  to be the tuple  $(S, s, \rightarrow, F)$  where  $S = \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E) \times \{1, \dots, n\} \times \{0, 1\}$  is the set of states and  $s = (\emptyset, E, \emptyset, 1, 1) \in S$  is the initial state and  $F = \mathcal{P}(E) \times \mathcal{P}(E) \times \mathcal{P}(E) \times \{1, \dots, n\} \times \{1\}$  is the set of final or accepting states.  $\rightarrow \subseteq S \times (\mathcal{P} \times \text{Act} \times R) \times S$  is the transition relation given by

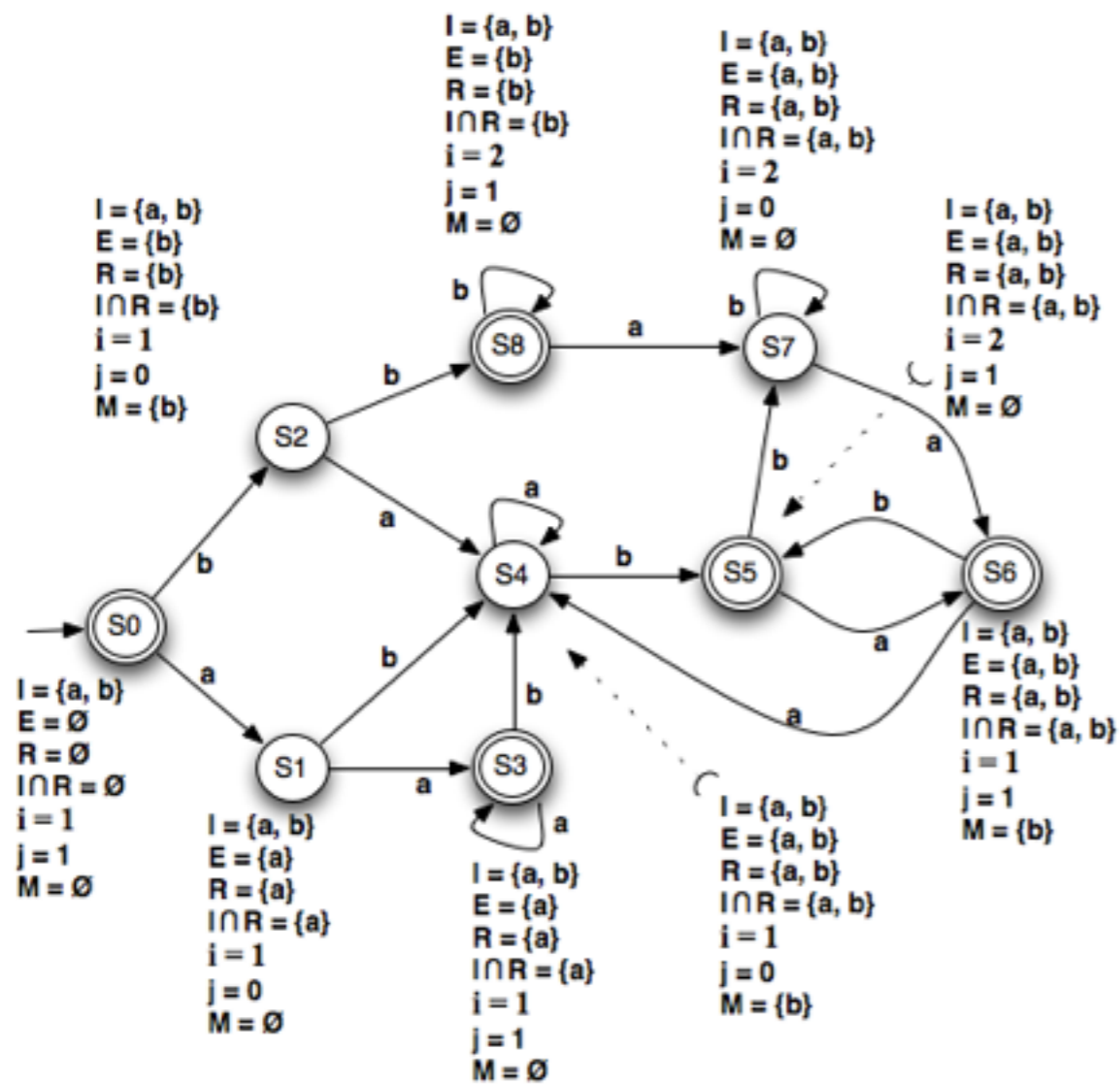
$$(E, I, R, i, j) \xrightarrow{(p,a,r)} (E \cup \{e\}, I', R', i', j') \text{ where}$$

- ▶ Semantics of  $E, I, R$  are same as previous accepting condition
- ▶  $j' = 1$  if
  1.  $I' \cap R' = \emptyset$
  2.  $\min(M) \in (I \cap R \setminus (I' \cap R')) \cup \{e\}$
  3.  $M = \emptyset$  and  $\min(I \cap R) \in (I \cap R \setminus (I' \cap R')) \cup \{e\}$otherwise  $j' = 0$ .
- ▶  $i' = \text{rank}(\min(M))$  if  $\min(M) \in (I \cap R \setminus (I' \cap R')) \cup \{e\}$
- ▶  $i' = \text{rank}(I \cap R)$  if  $M = \emptyset$  and  $\min(I \cap R) \in (I \cap R \setminus (I' \cap R')) \cup \{e\}$
- ▶  $i' = i$  otherwise.

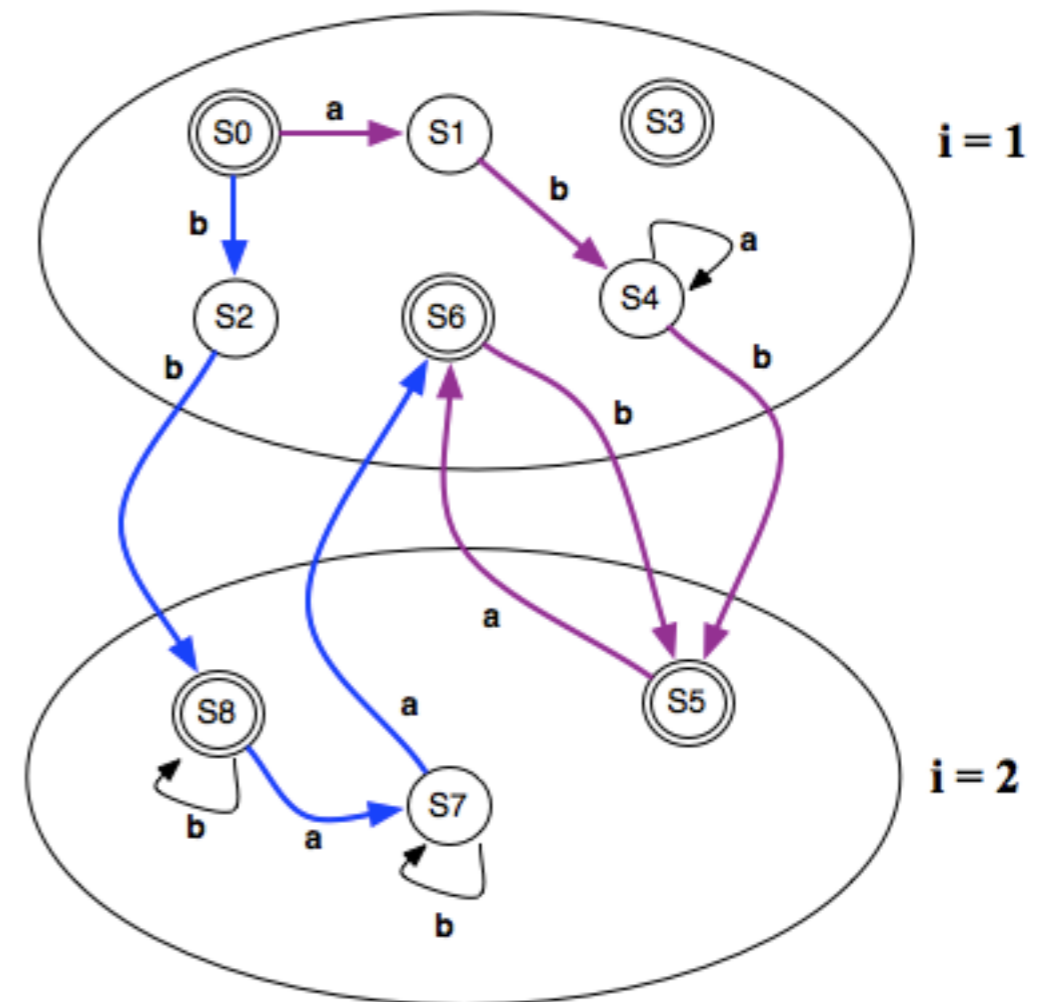
for  $M = \{e \in I \cap R \mid \text{rank}(e) > i\}$ .

- Mapped to a Generalized Buchi automata
- Instead of looking for  $R \cap I = \emptyset$ , we make sure that all pending response events ( $R \cap I$ ) will be eventually executed infinitely often.
- In other words, infinite run is not accepting if one or more pending response event(s) will never executed.

# Accepting condition for infinite runs - example



Automaton for example



accepting run for case where both a and b are executed with i copies of state

# Conclusion and Future Work

## Conclusion

- We have given a formal process model which is
  - easy to describe loosely constrained workflows in incremental way
  - easy to understand and to execute
  - conservatively extends both the model of event structures and the process language used by our industrial partner Resultmaker.

## Future work

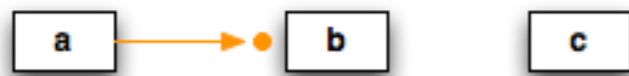
- model checking and verification using SPIN tool.
- Support for Data, Time, Sub-processes, Exceptions and Compensation
- Quantitative and probabilistic modalities on constraints
- Type system for DCRS and categorical constructions

Thank You  
Questions & Comments ?

# DCRS graphical notation

## Condition and Response relations

### 1. Condition relation

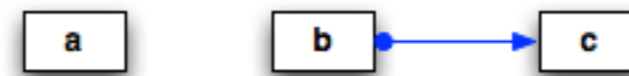


- OK
  - ✓ []
  - ✓ [c, a, a, b, c, b]
  - ✓ [a, b, b, a, a]
  - ✓ [c, c, a, a, a, c]
- Not OK
  - ⊙ [b]
  - ⊙ [c, b, c, a, b]

same as precedence relation in DECLARE<sup>2</sup>

$$\diamond(b) \Rightarrow ((!b) \cup a)$$

### 2. Response relation



- OK but not accept
  - ✓ [b]
  - ✓ [a, c, b, a]
- OK and Accept
  - ✓ []
  - ✓ [c, a, a, b, b, c]
  - ✓ [a, c, c, a, a]

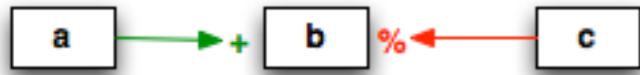
same as response relation in DECLARE<sup>2</sup>

$$\square(a \Rightarrow \diamond b)$$

[2] M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes Management

# DCRS Graphical Notation

## Include and Exclude relations



- OK
  - ✓ []
  - ✓ [a, b, c, a, b]
  - ✓ [b, c, a, a]
- Not OK
  - ⊙ [c, b]
  - ⊙ [a, b, c, b]