Logical Termination of Worflows:
An Interdisciplinary Approach

Glória Cravo

Center for Linear Structures and Combinatorics and University of Madeira

The 61st Séminaire Lotharingien de Combinatoire
Curia, September 21-24, 2008

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks.*

Definition 1 *A workflow is a tri-logic acyclic directed graph* $WG = (T, A)$, *where* $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite nonempty set of vertices representing workflow tasks. Each task* t_i *(i.e., a vertex) has an input logic operator (represented by* $\succ t_i$*) and an output logic operator (represented by* $t_i \prec$*).*

Definition 1 *A* *workflow* *is a tri-logic acyclic directed graph* $WG = (T, A)$, *where* $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite nonempty set of vertices representing* *workflow tasks*. *Each task* t_i (*i.e., a vertex*) *has an* *input logic operator* (*represented by* $\succ t_i$) *and an* *output logic operator* (*represented by* $t_i \prec$).

An input/output logic operator can be the logical AND ($\bullet$), *the OR* ($\otimes$), *or the XOR - exclusive-or* ($\oplus$).

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$).*

An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR - exclusive-or ($\oplus$). The set $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions.

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$).*

An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR - exclusive-or ($\oplus$). The set $A = \{a_\sqcup, a_\sqcap, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions. Each transition $a_i, i \in \{1, \ldots, m\}$, is a tuple (t_k, t_l) where $t_k, t_l \in T$.

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$).*

An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR - exclusive-or ($\oplus$). The set $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions. Each transition $a_i, i \in \{1, \ldots, m\}$, is a tuple (t_k, t_l) where $t_k, t_l \in T$. The transition $a_{\sqcup}$ is a tuple of the form $(\sqcup, t_1)$ and transition $a_{\sqcap}$ is a tuple of the form $(t_n, \sqcap)$.

Definition 1 *A* *workflow* *is a tri-logic acyclic directed graph* $WG = (T, A)$, *where* $T = \{t_1, t_2, \ldots, t_n\}$ *is a finite nonempty set of vertices representing* *workflow tasks*. *Each task* t_i (*i.e., a vertex*) *has an* *input logic operator* (*represented by* $\succ t_i$) *and an* *output logic operator* (*represented by* $t_i \prec$).

An input/output logic operator can be the logical AND ($\bullet$), *the OR* ($\otimes$), *or the XOR - exclusive-or* ($\oplus$). *The set* $A = \{a_{\sqcup}, a_{\sqcap}, a_1, a_2, \ldots, a_m\}$ *is a finite nonempty set of arcs representing* *workflow transitions*. *Each transition* $a_i, i \in \{1, \ldots, m\}$, *is a tuple* (t_k, t_l) *where* $t_k, t_l \in T$. *The transition* $a_{\sqcup}$ *is a tuple of the form* $(\sqcup, t_1)$ *and transition* $a_{\sqcap}$ *is a tuple of the form* $(t_n, \sqcap)$. *The symbols* $\sqcup$ *and* $\sqcap$ *represent abstract tasks which indicate the entry and ending point of the workflow, respectively.*

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$).*

An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR - exclusive-or ($\oplus$). The set $A = \{a_\sqcup, a_\sqcap, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions. Each transition $a_i, i \in \{1, \ldots, m\}$, is a tuple (t_k, t_l) where $t_k, t_l \in T$. The transition $a_\sqcup$ is a tuple of the form $(\sqcup, t_1)$ and transition $a_\sqcap$ is a tuple of the form $(t_n, \sqcap)$. The symbols $\sqcup$ and $\sqcap$ represent abstract tasks which indicate the entry and ending point of the workflow, respectively. We use the symbol $'$ to reference the label of a transition, i.e., a'_i references transition a_i, $a_i \in A$.

Definition 1 *A workflow is a tri-logic acyclic directed graph $WG = (T, A)$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a finite nonempty set of vertices representing workflow tasks. Each task t_i (i.e., a vertex) has an input logic operator (represented by $\succ t_i$) and an output logic operator (represented by $t_i \prec$).*

An input/output logic operator can be the logical AND ($\bullet$), the OR ($\otimes$), or the XOR - exclusive-or ($\oplus$). The set $A = \{a_\sqcup, a_\sqcap, a_1, a_2, \ldots, a_m\}$ is a finite nonempty set of arcs representing workflow transitions. Each transition $a_i, i \in \{1, \ldots, m\}$, is a tuple (t_k, t_l) where $t_k,\ t_l \in T$. The transition $a_\sqcup$ is a tuple of the form $(\sqcup, t_1)$ and transition $a_\sqcap$ is a tuple of the form $(t_n, \sqcap)$. The symbols $\sqcup$ and $\sqcap$ represent abstract tasks which indicate the entry and ending point of the workflow, respectively. We use the symbol $'$ to reference the label of a transition, i.e., a'_i references transition a_i, $a_i \in A$. The elements a'_i are called Boolean terms and form the set A'.

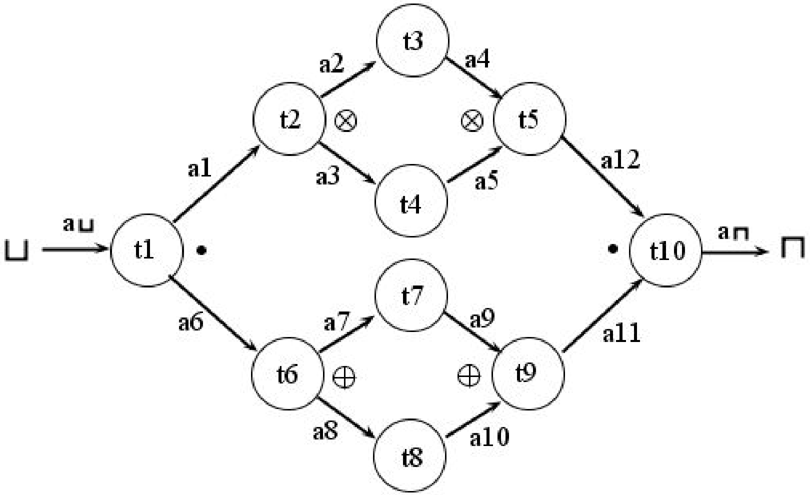

Figure: Example of a tri-logic acyclic directed graph (i.e., a workflow)

Example 2 Figure 1 shows a workflow $WG = (T, A)$, where $T = \{t_1, \ldots, t_{10}\}$, $A = \{a_{\sqcup}, a_{\sqcap}, a_1, \ldots, a_{12}\}$ and $A' = \{a'_{\sqcup}, a'_{\sqcap}, a'_1, \ldots, a'_{12}\}$. The tuple $a_2 = (t_2, t_3)$ is an example of a transition. In task t_2, $\otimes$ is the output logic operator $(t_2 \prec)$.

Example 2 Figure 1 shows a workflow $WG = (T, A)$, where $T = \{t_1, \ldots, t_{10}\}$, $A = \{a_\sqcup, a_\sqcap, a_1, \ldots, a_{12}\}$ and $A' = \{a'_\sqcup, a'_\sqcap, a'_1, \ldots, a'_{12}\}$. The tuple $a_2 = (t_2, t_3)$ is an example of a transition. In task t_2, $\otimes$ is the output logic operator $(t_2 \prec)$.

Definition 3 *The incoming transitions for task $t_i \in T$ are the tuples of the form $a_j = (x, t_i), x \in T, a_j \in A$, and the outgoing transitions for task t_i are the tuples of the form $a_l = (t_i, y), y \in T, a_l \in A$.*

Example 2 Figure 1 shows a workflow $WG = (T, A)$, where $T = \{t_1, \dots, t_{10}\}$, $A = \{a_\sqcup, a_\sqcap, a_1, \dots, a_{12}\}$ and $A' = \{a'_\sqcup, a'_\sqcap, a'_1, \dots, a'_{12}\}$. The tuple $a_2 = (t_2, t_3)$ is an example of a transition. In task t_2, $\otimes$ is the output logic operator $(t_2 \prec)$.

Definition 3 *The incoming transitions for task $t_i \in T$ are the tuples of the form $a_j = (x, t_i), x \in T, a_j \in A$, and the outgoing transitions for task t_i are the tuples of the form $a_l = (t_i, y), y \in T, a_l \in A$.*

Example 4 In Figure 1, the incoming transition for task t_2 is $a_1 = (t_1, t_2)$ and the outgoing transitions are $a_2 = (t_2, t_3)$ and $a_3 = (t_2, t_4)$.

Definition 5 *The* *incoming condition* *for task* $t_i \in T$ *is a Boolean expression with terms* $a' \in A'$, *where* a *is an incoming transition of task* t_i. *The terms* a' *are connected with the logical operator* $\succ t_i$.

Definition 5 *The incoming condition for task $t_i \in T$ is a Boolean expression with terms $a' \in A'$, where a is an incoming transition of task t_i. The terms a' are connected with the logical operator $\succ t_i$.*

Example 6 In Figure 1, the incoming condition for task t_2 is a'_1.

Definition 5 *The incoming condition for task $t_i \in T$ is a Boolean expression with terms $a' \in A'$, where a is an incoming transition of task t_i. The terms a' are connected with the logical operator $\succ t_i$.*

Example 6 In Figure 1, the incoming condition for task t_2 is a'_1.

Definition 7 *The outgoing condition for task $t_i \in T$ is a Boolean expression with terms $a' \in A'$, where a is an outgoing transition of task t_i. The terms a' are connected with the logical operator $t_i \prec$.*

Definition 5 *The incoming condition for task* $t_i \in T$ *is a Boolean expression with terms* $a' \in A'$, *where* a *is an incoming transition of task* t_i. *The terms* a' *are connected with the logical operator* $\succ t_i$.

Example 6 In Figure 1, the incoming condition for task t_2 is a'_1.

Definition 7 *The outgoing condition for task* $t_i \in T$ *is a Boolean expression with terms* $a' \in A'$, *where* a *is an outgoing transition of task* t_i. *The terms* a' *are connected with the logical operator* $t_i \prec$.

Example 8 In Figure 1, the outgoing condition for task t_2 is $a'_2 \otimes a'_3$.

Definition 9 *Given a workflow* $WG = (T, A)$, *an Event-Action (EA) model for a task* $t_i \in T$ *is an implication of the form* $t_i : f_E \rightsquigarrow f_C$, *where* f_E *and* f_C *are the incoming and outgoing conditions of task* t_i, *respectively. For any EA model* $t_i : f_E \rightsquigarrow f_C$, f_E *and* f_C *have the same Boolean value. The condition* f_E *is called the event condition and the condition* f_C *is called the action condition.*

Definition 9 *Given a workflow $WG = (T, A)$, an Event-Action (EA) model for a task $t_i \in T$ is an implication of the form $t_i : f_E \rightsquigarrow f_C$, where f_E and f_C are the incoming and outgoing conditions of task t_i, respectively. For any EA model $t_i : f_E \rightsquigarrow f_C$, f_E and f_C have the same Boolean value. The condition f_E is called the event condition and the condition f_C is called the action condition.*

Remark The behavior of an *EA* model is described in Table 1.

f_E	f_C	$f_E \rightsquigarrow f_C$
0	0	0
1	1	1

Table 1

Example 10 Let us consider task t_9 illustrated in Figure 1. Task t_9 has the following Event-Action model $\mathrm{t}_9 : a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$.
This model expresses that when only one of the Boolean terms a'_9, or a'_{10} is true, the event condition f_E is evaluated to true. In this case, the action condition f_C is evaluated to true, i.e., a'_{11} is true. Consequently, the model $f_E \rightsquigarrow f_C$ is true if and only if only one of the terms a'_9, a'_{10} is true and a'_{11} is true.

Example 10 Let us consider task t_9 illustrated in Figure 1. Task t_9 has the following Event-Action model $\mathrm{t}_9 : a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$.
This model expresses that when only one of the Boolean terms a'_9, or a'_{10} is true, the event condition f_E is evaluated to true. In this case, the action condition f_C is evaluated to true, i.e., a'_{11} is true. Consequently, the model $f_E \rightsquigarrow f_C$ is true if and only if only one of the terms a'_9, a'_{10} is true and a'_{11} is true.

Definition 11 *Let WG be a workflow and let* $t_i : f_E \rightsquigarrow f_C$ *be an EA model. We say that the EA model is positive if its value is 1, otherwise we say that the model is negative.*

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

(1) *The workflow starts its execution by asserting $a'_{\sqcup}$ to be true.*

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

(1) *The workflow starts its execution by asserting* $a'_{\sqcup}$ *to be true.*

(2) *For every EA model* $t_i : f_{E_i} \rightsquigarrow f_{C_i}, i \in \{1, \ldots, n\}$, *the Boolean values of* f_{E_i} *and* f_{C_i} *will be asserted according to Table* 1.

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

(1) *The workflow starts its execution by asserting* $a'_{\sqcup}$ *to be true.*

(2) *For every EA model* $t_i : f_{E_i} \rightsquigarrow f_{C_i}, i \in \{1, \ldots, n\}$*, the Boolean values of* f_{E_i} *and* f_{C_i} *will be asserted according to Table* 1.

(3) *The workflow stops its execution when one of the following cases occurs:*

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

(1) *The workflow starts its execution by asserting* $a'_{\sqcup}$ *to be true.*

(2) *For every EA model* $t_i : f_{E_i} \rightsquigarrow f_{C_i}, i \in \{1, \ldots, n\}$, *the Boolean values of* f_{E_i} *and* f_{C_i} *will be asserted according to Table* 1.

(3) *The workflow stops its execution when one of the following cases occurs:*

(3.1) $a'_{\sqcap}$ *is asserted to be true*;

Definition 12 *Let WG be a workflow. The behavior of WG is described by its EA models, according to the following rules:*

(1) *The workflow starts its execution by asserting* $a'_{\sqcup}$ *to be true.*

(2) *For every EA model* $t_i : f_{E_i} \rightsquigarrow f_{C_i}, i \in \{1, \ldots, n\}$, *the Boolean values of* f_{E_i} *and* f_{C_i} *will be asserted according to Table* 1.

(3) *The workflow stops its execution when one of the following cases occurs:*

(3.1) $a'_{\sqcap}$ *is asserted to be true;*

(3.2) $a'_{\sqcap}$ *is asserted to be false.*

Definition 13 *Let WG be a workflow. We say that WG logically terminates if $a'_{\sqcap}$ is true whenever $a'_{\sqcup}$ is true.*

Definition 13 *Let WG be a workflow. We say that WG* *logically terminates* *if $a'_{\sqcap}$ is true whenever $a'_{\sqcup}$ is true.*

Definition 14 *An EA model $f_E \rightsquigarrow f_C$ is said to be* *simple* *if $f_E = a'_i$ and $f_C = a'_j$, $i, j \in \{\sqcup, \sqcap, 1, \ldots, m\}$, with $i \neq j$.*

Definition 13 *Let WG be a workflow. We say that WG logically terminates if* $a'_{\sqcap}$ *is true whenever* $a'_{\sqcup}$ *is true.*

Definition 14 *An EA model* $f_E \leadsto f_C$ *is said to be simple if* $f_E = a'_i$ *and* $f_C = a'_j$, $i, j \in \{\sqcup, \sqcap, 1, \ldots, m\}$, *with* $i \neq j$.

Definition 15 *An EA model* $f_E \leadsto f_C$ *is said to be complex if* $f_E = a'_i$ *and* $f_C = a'_{j_1} \varphi a'_{j_2} \varphi \ldots \varphi a'_{j_k}$, *or* $f_E = a'_{j_1} \varphi a'_{j_2} \varphi \ldots \varphi a'_{j_k}$ *and* $f_C = a'_i$, *where* $\varphi \in \{\otimes, \bullet, \oplus\}$.

Definition 13 *Let WG be a workflow. We say that WG logically terminates if $a'_{\sqcap}$ is true whenever $a'_{\sqcup}$ is true.*

Definition 14 *An EA model $f_E \rightsquigarrow f_C$ is said to be simple if $f_E = a'_i$ and $f_C = a'_j$, $i, j \in \{\sqcup, \sqcap, 1, \ldots, m\}$, with $i \neq j$.*

Definition 15 *An EA model $f_E \rightsquigarrow f_C$ is said to be complex if $f_E = a'_i$ and $f_C = a'_{j_1} \varphi a'_{j_2} \varphi \ldots \varphi a'_{j_k}$, or $f_E = a'_{j_1} \varphi a'_{j_2} \varphi \ldots \varphi a'_{j_k}$ and $f_C = a'_i$, where $\varphi \in \{\otimes, \bullet, \oplus\}$.*

Definition 16 *An EA model $f_E \rightsquigarrow f_C$ is said to be hybrid if $f_E = a'_{i_1} \varphi a'_{i_2} \varphi \ldots \varphi a'_{i_l}$ and $f_C = a'_{j_1} \psi a'_{j_2} \psi \ldots \psi a'_{j_k}$, where $\varphi, \psi \in \{\otimes, \bullet, \oplus\}$.*

Definition 17 *The EA models from definitions 15, 16 are called non-simple EA models.*

Definition 17 *The EA models from definitions 15, 16 are called non-simple EA models.*

Example 18 *In Figure 1 the EA model* $t_3 : a'_2 \rightsquigarrow a'_4$ *is simple, while the EA models* $t_2 : a'_1 \rightsquigarrow a'_2 \otimes a'_3$ *and* $t_9 : a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$ *are non-simple.*

Theorem 19 *A hybrid EA model* $f_E \rightsquigarrow f_C$ *can be split into two derived equivalent complex EA models* $f_E \rightsquigarrow a_i^*$ *and* $a_i^* \rightsquigarrow f_C$.

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$).

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$). Let us create two auxiliary tasks t_i', t_i'' and an auxiliary transition $a_i^\top = (t_i', t_i'')$.

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$). Let us create two auxiliary tasks t_i', t_i'' and an auxiliary transition $a_i^\intercal = (t_i', t_i'')$. Let a_i^* be the Boolean term associated with the auxiliary transition $a_i^\intercal$, such that a_i^* has the same Boolean value of f_E.

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and $(\bullet)$, an or $(\otimes)$, or an exclusive-or $(\oplus)$. Let us create two auxiliary tasks t_i', t_i'' and an auxiliary transition $a_i^\intercal = (t_i', t_i'')$. Let a_i^* be the Boolean term associated with the auxiliary transition $a_i^\intercal$, such that a_i^* has the same Boolean value of f_E. Let $t_i' : f_E \rightsquigarrow a_i^*$ and $t_i'' : a_i^* \rightsquigarrow f_C$ be new *EA* models.

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$). Let us create two auxiliary tasks t_i', t_i'' and an auxiliary transition $a_i^{\mathsf{T}} = (t_i', t_i'')$. Let a_i^* be the Boolean term associated with the auxiliary transition a_i^{T}, such that a_i^* has the same Boolean value of f_E. Let $t_i' : f_E \rightsquigarrow a_i^*$ and $t_i'' : a_i^* \rightsquigarrow f_C$ be new *EA* models. Since a_i^* has the same Boolean value of f_E and, as a consequence, f_C has its Boolean value depending on the Boolean value of a_i^*, when we consider these new *EA* models instead of the initial hybrid *EA* model, the behavior of the workflow is not modified.

Theorem 19 *A hybrid EA model $f_E \rightsquigarrow f_C$ can be split into two derived equivalent complex EA models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$.*

Proof. Suppose that $t_i : f_E \rightsquigarrow f_C$ is a hybrid *EA* model. Then both f_E and f_C are Boolean terms with an and ($\bullet$), an or ($\otimes$), or an exclusive-or ($\oplus$). Let us create two auxiliary tasks t_i', t_i'' and an auxiliary transition $a_i^{\top} = (t_i', t_i'')$. Let a_i^* be the Boolean term associated with the auxiliary transition $a_i^{\top}$, such that a_i^* has the same Boolean value of f_E. Let $t_i' : f_E \rightsquigarrow a_i^*$ and $t_i'' : a_i^* \rightsquigarrow f_C$ be new *EA* models. Since a_i^* has the same Boolean value of f_E and, as a consequence, f_C has its Boolean value depending on the Boolean value of a_i^*, when we consider these new *EA* models instead of the initial hybrid *EA* model, the behavior of the workflow is not modified. Clearly the new *EA* models $f_E \rightsquigarrow a_i^*$ and $a_i^* \rightsquigarrow f_C$ are complex and so the result is satisfied. ∎

Notation *The set of all auxiliary tasks created from* T, *will be denoted by* T^* *and the set of all auxiliary transitions created from* A *will be denoted by* A^*.

Notation *The set of all auxiliary tasks created from T, will be denoted by T^* and the set of all auxiliary transitions created from A will be denoted by A^*.*

Definition 20 *Let $WG_1 = (T_1, A_1)$ and $WG_2 = (T_2, A_2)$ be workflows. Suppose that $T_2 = T_1 \cup T_1^*$ and $A_2 = A_1 \cup A_1^*$. Let NH_i be the set of all non-hybrid EA models of $WG_i, i \in \{1, 2\}$. We say that WG_2 is derived from WG_1, or WG_1 derives WG_2, if the following conditions are satisfied:*

Notation *The set of all auxiliary tasks created from T, will be denoted by T^* and the set of all auxiliary transitions created from A will be denoted by A^*.*

Definition 20 *Let $WG_1 = (T_1, A_1)$ and $WG_2 = (T_2, A_2)$ be workflows. Suppose that $T_2 = T_1 \cup T_1^*$ and $A_2 = A_1 \cup A_1^*$. Let NH_i be the set of all non-hybrid EA models of $WG_i, i \in \{1, 2\}$. We say that WG_2 is derived from WG_1, or WG_1 derives WG_2, if the following conditions are satisfied:*

(a) $NH_1 = NH_2$;

Notation *The set of all auxiliary tasks created from T, will be denoted by T^* and the set of all auxiliary transitions created from A will be denoted by A^*.*

Definition 20 *Let $WG_1 = (T_1, A_1)$ and $WG_2 = (T_2, A_2)$ be workflows. Suppose that $T_2 = T_1 \cup T_1^*$ and $A_2 = A_1 \cup A_1^*$. Let NH_i be the set of all non-hybrid EA models of $WG_i, i \in \{1, 2\}$. We say that WG_2 is derived from WG_1, or WG_1 derives WG_2, if the following conditions are satisfied:*

(a) $NH_1 = NH_2$;

(b) *Every hybrid EA model of WG_1 is split into two complex EA models of WG_2.*

Theorem 21 *Let $WG_1 = (T_1, A_1)$ and $WG_2 = (T_2, A_2)$ be workflows and assume that WG_2 is derived from WG_1. Then, WG_1 logically terminates if and only if WG_2 logically terminates.*

Theorem 21 *Let $WG_1 = (T_1, A_1)$ and $WG_2 = (T_2, A_2)$ be workflows and assume that WG_2 is derived from WG_1. Then, WG_1 logically terminates if and only if WG_2 logically terminates.*

Remark According to Theorem 19, from now on, we can consider workflows without hybrid *EA* models.

Clearly, if all *EA* models of the workflow are simple, then its structure is the following:

$$\sqcup \xrightarrow{a_{\sqcup}} t_1 \xrightarrow{a_1} t_2 \xrightarrow{a_2} t_3 \ldots t_{n-1} \xrightarrow{a_n} t_n \xrightarrow{a_{\sqcap}} \sqcap.$$

Clearly, if all *EA* models of the workflow are simple, then its structure is the following:

$$\sqcup \xrightarrow{a_\sqcup} t_1 \xrightarrow{a_1} t_2 \xrightarrow{a_2} t_3 \ldots t_{n-1} \xrightarrow{a_n} t_n \xrightarrow{a_\sqcap} \sqcap.$$

In this case, the set of non-simple *EA* models is empty. This situation is a trivial case of logical termination, since all the *EA* models present in the workflow are positive, and consequently, $a'_\sqcap$ is *true* whenever $a'_\sqcup$ is *true*, i.e., the workflow logically terminates. From now on, we will assume that the workflow contains non-simple *EA* models.

Definition 22 *Let $WG = (T, A)$ be a workflow. A materialized workflow instance of WG is an assignment of Boolean values to all Boolean terms $a'_j \in A'$, according to Table 1.*

Definition 22 *Let $WG = (T, A)$ be a workflow. A materialized workflow instance of WG is an assignment of Boolean values to all Boolean terms $a'_j \in A'$, according to Table* 1.

Notation Let $N = \{i \in \{1, \ldots, n\} | t_i : f_{E_i} \rightsquigarrow f_{C_i}$ is a non-simple *EA* model$\}$.

Definition 23 *Assume that* $N = \{i_1, \ldots, i_l\}$ *and the elements* $i_1, \ldots, i_l$ *appear in increasing order, i.e.,* $i_1 < \cdots < i_l$. *For any materialized workflow instance of* WG, *let* $B = [b_{i,j}] \in F^{l \times l}$ *be the Boolean matrix, which entries are defined as follows:*

$$b_{i,j} = \begin{cases} \text{Boolean value of the } EA \text{ model } t_i : f_{E_i} \rightsquigarrow f_{C_i} \ (i \in N), & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} .$$

The matrix B *is called the Event Action Boolean matrix.*

Definition 23 *Assume that $N = \{i_1, \ldots, i_l\}$ and the elements $i_1, \ldots, i_l$ appear in increasing order, i.e., $i_1 < \cdots < i_l$. For any materialized workflow instance of WG, let $B = [b_{i,j}] \in F^{l \times l}$ be the Boolean matrix, which entries are defined as follows:*

$$b_{i,j} = \begin{cases} \text{Boolean value of the } EA \text{ model } t_i : f_{E_i} \rightsquigarrow f_{C_i} \ (i \in N), & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} .$$

The matrix B is called the Event Action Boolean matrix.

Theorem 24 *Let $WG = (T, A)$ be a workflow and assume that $N = \{i_1, \ldots, i_l\}$, $i_1 < \cdots < i_l$. Then WG logically terminates if and only if every Event Action Boolean matrix is equal to the identity matrix of type $l \times l$.*

Example 25 The workflow from Figure 1 has the following non-simple *EA* models: $t_1 : a'_{\sqcup} \rightsquigarrow a'_1 \bullet a'_6$, $t_2 : a'_1 \rightsquigarrow a'_2 \otimes a'_3$, $t_5 : a'_4 \otimes a'_5 \rightsquigarrow a'_{12}$, $t_6 : a'_6 \rightsquigarrow a'_7 \oplus a'_8$, $t_9 : a'_9 \oplus a'_{10} \rightsquigarrow a'_{11}$, $t_{10} : a'_{11} \bullet a'_{12} \rightsquigarrow a'_{\sqcap}$. Hence $N = \{1, 2, 5, 6, 9, 10\}$. We have as many Event Action Boolean matrices as materialized workflow instances of WG. It is easy to verify that every Event Action Boolean matrix is equal to the identity matrix of type 6×6. Therefore, the workflow logically terminates.