Fly-automata for checking monadic second-order properties of graphs of bounded tree-width

Bruno Courcelle [a,1]

[a] *LaBRI, CNRS and Bordeaux University, 33405 Talence, France*

Abstract

Every graph property expressible in *monadic second-order* (MSO) logic, possibly with quantifications over edges, can be checked in linear time on graphs of bounded *tree-width*, in particular by means of finite automata running on terms denoting tree-decompositions. However, implementing these automata is difficult because of their huge sizes. *Fly-automata* (FA) are deterministic automata that *compute* the necessary states and transitions when running (instead of *looking into tables*); they allow us to overcome this difficulty. In previous works, we constructed FA to check MSO properties of graphs of bounded *clique-width*. An MSO property with edge quantifications (called an MSO_2 property) of a graph is an MSO property of its *incidence graph* and, on the other hand, graphs of tree-width k have incidence graphs of clique-width $O(k)$. Thus, our existing constructions can be used for MSO_2 properties of graphs of bounded tree-width. We examine concrete aspects of this adaptation.

Keywords: Graph algorithm, fixed-parameter tractability, tree-width, incidence graph, clique-width, finite automaton, fly-automaton, monadic second-order logic, edge quantification, model-checking.

1 Introduction

Graphs are finite and directed. The extension to undirected graphs is easy. Our goal is to check their *monadic second-order* (MSO) properties by using finite

[1] Email: courcell@labri.fr. This work has been supported by the French National Research Agency (ANR) within the IdEx Bordeaux program "Investments for the future", CPU, ANR-10-IDEX-03-02.

automata running on the terms that denote input graphs of bounded tree-width or clique-width, and to get *fixed-parameter tractable* (FPT) algorithms, constructed by automatic and usable methods.

Fact 1: For each k and MSO expressible graph property P, there is an $O(n^3)$-algorithm that checks this property on graphs with n vertices of *clique-width* $\leq k$. It is based on a finite automaton $\mathcal{A}_{P,k}$ that takes as input a term t of width at most k that denotes the input graph assumed of clique-width $\leq k$ [6,7].

Fact 2: The size of $\mathcal{A}_{P,k}$ is usually so large that it forbids implementation in the classical way. Instead of trying to list transitions in huge tables, we describe them by small programs, and so, we make $\mathcal{A}_{P,k}$ into a *fly-automaton* (an *FA* in short). Implementation of FA has been tested in significant cases [4].

Fact 3 : The *algorithmic meta-theorem* of Fact 1 has a similar version for graphs of bounded *tree-width* and properties expressed by MSO formulas using *edge set quantifications* (MSO_2 properties), hence for "less graphs" but for "strictly more properties" [6,2]. It suffices to take as input the *incidence graph* $Inc(G)$ of the input graph G. This method works because $cwd(Inc(G))$, the clique-width of $Inc(G)$, is bounded by a function of $twd(G)$, the tree-width of G.

Fact 4 : This method is usable because $cwd(Inc(G)) \leq 2.twd(G)+4$ for G directed by a recent result [1,3] improving a previous exponential upper-bound. See the appendix. The reduction of Fact 3 needs some work on automata.

The present contribution : The reduction of Fact 3 needs some work on automata. We will detail some new necessary constructions and we will give a proof of Fact 4.

This work has been presented at the international conference LAGOS, Beberibe, Ceará, Brazil, in May 2015. A shortened version of this text will appear in the proceedings (Electronic Notes in Discrete Mathematics).

2 Definitions and discussion of background results.

About Fact 1 : *MSO logic.* A simple graph G is identified with the logical structure $\langle V_G, edg_G \rangle$ with domain V_G, the set of vertices and edg_G, the binary

relation such that $(x, y) \in edg_G$ if and only if there is an edge from x to y (denoted by $x \to y$). 3-vertex colorability is expressible by the MSO formula $\exists X, Y.Col(X, Y)$ where $Col(X, Y)$ expresses that X, Y and $V_G - (X \cup Y)$ are the three color classes of a 3-coloring. The existence of an acyclic p-coloring, connectedness, planarity (via forbidden minors) are MSO-expressible.

Clique-width is a graph complexity measure, comparable to tree-width, that is defined from operations that construct simple graphs equipped with vertex labels. Let C be a finite set of labels. A *C-graph* is a triple $G = \langle V_G, edg_G, \pi_G \rangle$ where π_G is a mapping: $V_G \to C$. We let F_C be the following finite set of operations on C-graphs: $\oplus$ is the union of two disjoint graphs, the unary operation $relab_h$ changes every vertex label a into $h(a)$, where h is a mapping from C to C, the unary operation $\overrightarrow{add}_{a,b}$, for $a \neq b$ adds an edge from every a-labelled x to every b-labelled y (unless we already have an edge $x \to y$) and for each $a \in C$, the nullary symbol $\mathbf{a}$ denotes an isolated vertex labelled by a.

Every term t in $T(F_C)$ denotes a C-graph $G(t)$. The clique-width of a graph G, denoted by $cwd(G)$, is the least integer k such that G is isomorphic, up to vertex labels, to $G(t)$ for some t in $T(F_{[k]})$ ($[k] := \{1, ..., k\}$). The clique-width of a simple graph G is bounded in terms of its tree-width $twd(G)$ by $cwd(G) \leq 2^{2twd(G)+2} + 1$, see [6].

For every integer k and MSO formula expressing a property P, one can construct a finite deterministic automaton $\mathcal{A}_{P,k}$ that recognizes the terms in $T(F_{[k]})$ that define graphs satisfying P. This gives a linear time recognition algorithm for graphs given by such terms (finding a term needs cubic time).

About Fact 2 : These automata $\mathcal{A}_{P,k}$ have in most cases so many states that their transition tables cannot be built. This is not avoidable [9]. In the article [4] we have introduced automata called *fly-automata* (*FA*) whose states are described (but not listed) and whose transitions are computed "on the fly" (and not tabulated) [2]. The states, although numerous, have a common

[2] FA can have infinitely many states: a state can record the (unbounded) number of occurrences of a particular symbol. We can construct fly-automata [5] that check properties that are not MSO expressible (e.g., that a graph is regular or can be partitioned into p disjoint regular graphs). These automata yield FPT or XP algorithms [8] for clique-width as parameter. By equipping fly-automata with output functions, we can make them compute values attached to graphs. We can compute the number of s-colorings, or, assuming that the graph is s-colorable, the minimum size of X_1 in an s-coloring $(X_1, \ldots, X_s)$. The number of acyclic 4-colorings of Petersen's graph is 10800 and the number of acyclic 3-colorings of McGee's graph is 57024, see [5].

syntactic structure. An FA having $2^{2^{10}}$ states computes only 100 states on a term of length 100. (If P is s-colorability, the automaton $\mathcal{A}_{P,k}$ has more than $2^{2^{s.k}}$ states.) The maximum size of a state (the number of bits for encoding it) *used on an input term* is more important in order to bound the computation time than the total number of states (see [5]).

About Facts 3 and 4: Since the domain of the structure $\langle V_G, edg_G\rangle$ is the vertex set and quantifications over binary relations are not allowed, MSO formulas cannot quantify over sets of edges. The *incidence graph* of G is the bipartite graph $Inc(G) := \langle V_G \cup E_G, inc_G\rangle$ such that E_G is the set of edges and inc_G is the set of pairs (x, u) such that x is the tail of edge u and (u, y) such that y is its head. As each edge of G is made into a vertex of $Inc(G)$, an MSO formula over $Inc(G)$ can be seen as an MSO formula using quantifications on edges, called an MSO_2 formula. That the considered graph has a directed Hamiltonian cycle is MSO_2 expressible but not MSO expressible over $\langle V_G, edg_G\rangle$, [6]. However, the meta-theorem of Fact 1 does not extend to inputs $Inc(G)$ for G of *cwd* at most k because the clique-width of such graphs $Inc(G)$ is unbounded. But it can be used for graphs G of tree-width at most k, because :

Theorem 2.1 *([1,3] and the appendix) : If a directed graph G has tree-width k, then $cwd(Inc(G)) \leq 2k + 4$ and a term in $T(F_{[2k+4]})$ that defines $Inc(G)$ can be constructed in linear time from a tree-decomposition of G of width k. If G is undirected, then $cwd(Inc(G)) \leq k + 3$. Conversely, $twd(G) = O(cwd(Inc(G)))$.*

3 Automata

In [4], we have constructed fly-automata for atomic MSO formulas (like $X \subseteq Y$ or $edg(X, Y)$ expressing that $X = \{x\}$, $Y = \{y\}$ and $x \to y$) and particular MSO properties (like $Partition(X_1, \ldots, X_s)$, $Conn(X)$ expressing that $G[X]$ is connected or $Cycle(X)$ expressing that $G[X]$ has an undirected cycle). They are useful for other properties (e.g. that G contains a fixed minor H or is acyclically p-vertex colorable) because automata can be combined so as to reflect logical connectives : $\vee, \wedge, \neg, \exists X$.

Adapting the basic construction of "automata for MSO", cf. Fact 1.

MSO logic : We use two types of set variables: $X, Y, ...$ for sets of vertices and U, V, W for sets of edges (i.e., sets of degree two vertices representing in $Inc(G)$ the edges of G).

Clique-width terms for incidence graphs : We use two disjoint sets of labels, C for the "real" vertices and D for the vertices representing edges. No label in C can be changed to a label in D, and no edge-addition $\overrightarrow{add}_{a,b}$ can be used with a and b both in C or in D. We let $T(F_{C,D})$ be the corresponding set of terms. If G has tree-width k, then $Inc(G)$ is defined by a term in $T(F_{C,D})$ such that $|C| = 2$ and $|D| = 2k + 3$. (See the appendix).

Automata on $T(F_{C,D})$.

1) Some terms in $T(F_{C,D})$ may generate graphs that are not incidence graphs: those with a vertex having label in D and indegree or outdegree different from 1. An automaton described in appendix can check the property that the given term is *correct*, i.e., defines an incidence graph. Its states are $Error$ and tuples in $\mathcal{P}(C)^2 \times \mathcal{P}(D)^4$ hence, they can be encoded by words of length $O(k)$ for recognizing incidence graphs of graphs of tree-width k, by Theorem 2.1. ($\mathcal{P}(X)$ denotes the powerset of a set X).

2) In the basic case [4,6], the atomic formula $edg(X, Y)$ is checked by a "small" automaton with k^2+k+3 states for terms in $T(F_{[k]})$. This construction is easily applicable to the atomic formula $inc(X, U)$ (resp. $inc(U, X)$) stating that X is one vertex x, U is one edge u whose tail (resp. head) is x. In $Inc(G)$, the property $edg(X, Y)$ is no longer atomic; it is expressed by $\exists U.(inc(X, U) \wedge inc(U, Y))$. We can apply the general construction of [4,6] to this formula, but it is more efficient to define directly an automaton over $F_{C,D}$. Its states are 5-tuples in $C^2 \times \mathcal{P}(D)^3$ hence, they can be encoded by words of length $O(|C| + |D|)$. We have an exponential jump from $O(cwd^2)$ to $O(2^{6twd})$ for the number of states which is not surprizing because the formula expressing $edg(X, Y)$ has an existential quantification.

3) For the three properties:

$Link^{\exists\exists}(X, Y)$ meaning that $x \to y$ for some $x \in X$ and $y \in Y$,

$Link^{\forall\exists}(X, Y)$ meaning that for all $x \in X$ there is $y \in Y$ such that $x \to y$, (Y *dominates* X in some sense), and

$Link^{\forall\forall}(X, Y)$ meaning that $x \to y$ for all $x \in X$ and $y \in Y$,

in the appendix, we will contruct automata whose states are tuples, belonging respectively, to the following sets:

$\mathcal{P}(C)^2 \times \mathcal{P}(D)^3$; states are of size $O(k)$, where $k = |C| + |D|$,

$\mathcal{P}(C)^2 \times \mathcal{P}(D)^2 \times \mathcal{P}(C \times \mathcal{P}(D))^2$; states are of size $O(k^2.2^k))$,

$\mathcal{P}(C) \times \mathcal{P}(D) \times \mathcal{P}(C \times \mathcal{P}(D))^2 \times \mathcal{P}(C^2 \times \mathcal{P}(D)^2)$; states of size $O(k^3.2^{2k})$.

For comparison, the corresponding automata over $F_{[k]}$, constructed as in [4], have states of respective sizes $O(k)$, $O(k)$ and $O(k^2)$. A universal quantifi-

cation corresponds to a negation before an existential quantification. This explains the exponential jump between $Link^{\exists\exists}(X,Y)$ and the two others. These sets of states may look very large, but an FA uses only the necessary states on a given term, and usual terms do not have the combinatorial properties yielding the maximal sizes. So these automata are actually implementable.

4) *Inc-stable properties are easy.*

A property P is *Inc-stable* if $P(G) \iff P(Inc(G))$. So are, for instance, connectedness and strong connectedness, the properties that G has a directed cycle, or an undirected cycle, or a path from vertex x to vertex y or has all its vertices of outdegree at most p. For them, FA on terms in $T(F_{C,D})$ are constructible from the FA $\mathcal{A}_{P,k}$ (cf. Fact 1) without needing any significant modification.

To summarize, we have the following theorem.

Theorem 3.1 *Let k bound the tree-width of the considered graphs G: this value determines sets C and D and incidence graphs $Inc(G)$ are given by terms in $T(F_{C,D})$.*

(1) That a term in $T(F_{C,D})$ is correct is checked by an FA with states of size $O(k)$.

(2) Properties $edg(X,Y)$ and $Link^{\exists\exists}(X,Y)$ are checked by FA with states of size $O(k)$.

(3) Properties $Link^{\forall\exists}(X,Y)$ and $Link^{\forall\forall}(X,Y)$ are checked by FA with states of size $2^{O(k)}$.

(4) Inc-stable properties are checked by FA of same state sizes as for clique-width terms.

Conclusion : These results indicate that the tools of [4,5] can be applied to the verification of MSO_2 properties of graphs of bounded tree-width given by their tree-decompositions. The software AUTOGRAPH can be used basically as it is (up to minor syntactic adaptations) although the terms that describe tree-decompositions are fairly different from those defining clique-width [2,6].

References

[1] T. Bouvier, Graphes et décompositions, Doctoral dissertation, Bordeaux University, December 2014.

[2] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications, *Discrete Applied Mathematics* **160**(2012) 866-887.

[3] B. Courcelle, Clique-width and tree-width of sparse graphs, Research report, 2015.

[4] B. Courcelle and I. Durand, Automata for the verification of monadic second-order graph properties, *J. Applied Logic* **10** (2012) 368-409.

[5] B. Courcelle and I. Durand, Computations by fly-automata beyond monadic second-order logic, June 2013, to appear in *Theor. Comput. Sci.*, Short version in Proceedings of the Conference on Algebraic Informatics, *Lecture Notes in Computer Science* **8080** (2013) 211-222.

[6] B. Courcelle and J. Engelfriet, *Graph structure and monadic second-order logic, a language theoretic approach*, Volume **138** of *Encyclopedia of mathematics and its application*, Cambridge University Press, June 2012.

[7] B. Courcelle, J. Makowsky and U. Rotics, Linear-time solvable optimization problems on graphs of bounded clique-width, *Theory Comput. Syst.* **33** (2000) 125-150.

[8] R. Downey and M. Fellows, *Parameterized complexity*, Springer-Verlag, 1999.

[9] M. Frick, M. Grohe, The complexity of first-order and monadic second-order logic revisited, *Ann. Pure Appl. Logic* **130** (2004) 3-31.

A Appendix

A: MSO expressible properties.

MSO expressibility

3-colorability is expressible by the MSO formula $\exists X, Y.Col(X, Y)$ where $Col(X, Y)$ is the formula

$X \cap Y = \emptyset \wedge \forall u, v.\{edg(u, v) \Longrightarrow [\neg(u \in X \wedge v \in X)$

$\wedge\neg(u \in Y \wedge v \in Y) \wedge \neg(u \notin X \cup Y \wedge v \notin X \cup Y)]\}$.

Capital variables X and Y denote sets of vertices and $Col(X, Y)$ expresses that X, Y and $V_G - (X \cup Y)$ are the three color classes of a 3-coloring. More generally, p-colorability is expressible in a similar way. A p-coloring defined by a partition $(X_1, ..., X_p)$ is *acyclic* if each induced subgraph $G[X_i \cup X_j]$ has no undirected cycle (i.e., neglecting edge directions), which is MSO expressible. The existence of an acyclic p-coloring is MSO-expressible. Connectivity, planarity (via forbidden minors) are also MSO-expressible.

Quantifications over binary relations are not allowed in MSO logic; equipotence of two sets is not MSO expressible.

MSO_2 expressibility (through incidence graphs)

The formula : "there exists a set of edges that induces a directed cycle and goes through all vertices", expresses that the considered graph has a directed Hamiltonian cycle. This property is not expressible in MSO logic over $\langle V_G, edg_G\rangle$ [6].

B: Clique-width of incidence graphs.

We sketch the proof that $cwd(Inc(G)) \leq 2.twd(G) + 4$ if G is directed, where $twd(G)$ denotes the tree-width of G.

Tree-width is based on tree-decompositions and these decompositions can be expressed by terms [6]. By induction on the structure of a term t that defines a tree-decomposition of G, we construct a term in $T(F_{C,D})$ that defines $Inc(G)$, in order to prove the claimed bounding. We need some definitions.

Graphs with sources.

Let G be a directed graph with vertex set V_G and edge set E_G. It may have loops and multiple edges. The notation $u : x \to y$ means that edge u links x to y. We equip G with distinguished vertices called *sources* in [6] and

boundary vertices or *terminals* by other authors [8]. Let K be a finite set of labels, let src be an injective mapping: $K \to V_G$. We say that x is the *a-source* if $src(a) = x$. If H is a *sourced graph* (G, src), we let $\tau(H) := K$ and $Src(H)$ be the set of sources of H, i.e., the set $src(K)$. The set of *internal vertices* of H is $Int_H := V_G - Src(H)$.

We now define some operations on sourced graphs:

If $H = (G, src)$, then $fg_a(H) := (G, src \restriction (\tau(H) - \{a\}))$. In words, the a-source of H is no longer a source in $fg_a(H)$ (fg means "forget" and $\restriction$ denotes the restriction of a function to a subset of its domain) but an internal vertex; nothing else is changed. If H has no a-source, then $fg_a(H) = H$.

If $H = (G, src)$ and $H' = (G', src')$ then $H//H'$ is constructed as follows: one takes the union of G and a copy of G' disjoint from G and one fuses the a-source of G and the a-source of G' for each $a \in \tau(H) \cap \tau(H')$. We have $\tau(H//H') = \tau(H) \cup \tau(H')$. (Any two isomorphic copies of G' yield isomorphic results).

Basic graphs are $\mathbf{a}$, $\mathbf{a}^\ell$ and $\overrightarrow{\mathbf{ab}}$, respectively an isolated a-source, an a-source with a loop and an edge directed from an a-source to a b-source ($b \neq a$).

We let L_K be the set of operations $fg_a, //, \mathbf{a}, \mathbf{a}^\ell$ and $\overrightarrow{\mathbf{ab}}$ for $a, b \in K$. Every term $t \in T(L_K)$ defines a sourced graph $\widehat{G}(t)$ such that $\tau(\widehat{G}(t)) \subseteq K$. ($\widehat{G}(t)$ is well-defined up to isomorphism because of $//$; details are in [6]; K can be empty: every graph is a sourced graph).

Proposition 1 ([6], Theorem 2.83 and Remark 2.84(1)) : A graph has tree-width at most k iff it is (isomorphic to) $\widehat{G}(t)$ for some $t \in T(L_{[k+1]})$. A term t can be constructed in linear time (for fixed k) from a tree-decomposition of width at most k.

Constructing $Inc(G)$ from G :

We insert on each edge of G a new vertex labelled by $\square$ intended to represent this edge. The sources of G remain sources in $Inc(G)$ with same labels. The new vertices, called *edge-vertices* are not sources and form the EV_G. The operations $//$ and fg_a applied to incidence graphs with sources do not modify edge-vertices. Clearly:

$$Inc(G//H) = Inc(G)//Inc(H) \text{ and } Inc(fg_a(G)) = fg_a(Inc(G)).$$

The main construction

Let G be a graph with source labels in a finite set K. We define a sourced graph $\alpha(G)$ and a labelled graph $\beta(G)$.

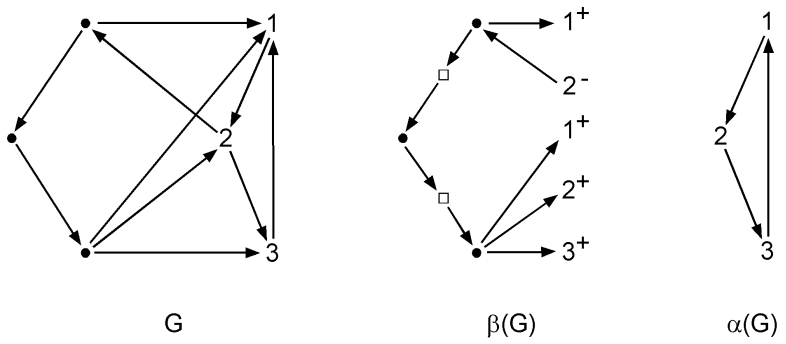

Fig. A.1. A sourced graph $G, \beta(G)$ and $\alpha(G)$.

1) $\alpha(G) := G[Src(G)]$, i.e., it is the subgraph of G induced on its sources. The sources of $\alpha(G)$ are those of G with same labels.

2) We let $D := \{c^+, c^- \mid c \in K\} \cup \{\Box\}$ and $C' := \{\bullet\}$.

We define a labelled graph $\beta(G)$: its vertex set is $Int_G \cup W$ where $W \subseteq EV_G$ is the set of vertices of $Inc(G)$ that represent edges of G incident with at least one internal vertex (the other edges of G are in $G[Src(G)] = \alpha(G)$); the edges of $\beta(G)$ are those of $Inc(G)[Int_G \cup W]$ and its vertices are labelled as follows:

a vertex in Int_G has label $\bullet$,

a vertex in W that represents an edge of G whose two ends are internal vertices has label $\Box$,

a vertex in W that represents an edge $u : x \to y$ such that x is the c-source has label c^-,

a vertex in W that represents an edge $u : x \to y$ such that y is the c-source has label c^+.

If G has no source, then $\beta(G) = Inc(G)$ (with labels $\Box$ and $\bullet$) and $\alpha(G)$ is the empty graph. Figure A.1 shows an example where $\tau(G) = Src(G) = \{1, 2, 3\}$.

Claim 2 : For all sourced graphs G and H:

(i) $\alpha(G//H) = \alpha(G)//\alpha(H)$ and $\beta(G//H) = \beta(G) \oplus \beta(H)$.

(ii) Assume $a \in \tau(G)$ (otherwise $fg_a(G) = G$)):

$\alpha(fg_a(G)) = \alpha(G)[Src(G) - \{x\}]$ where x is the a-source of G and

$\beta(fg_a(G))$ is constructed from $\beta(G)$ as follows:

1) it has one new vertex, say $\overline{x}$, labelled by $\bullet$ corresponding to x, the a-source of G (and of $\alpha(G)$) made internal in $fg_a(G)$,

2) for each vertex u of $\beta(G)$ labelled by a^-, we add an edge: $\overline{x} \to u$, and we relabel u by $\Box$,

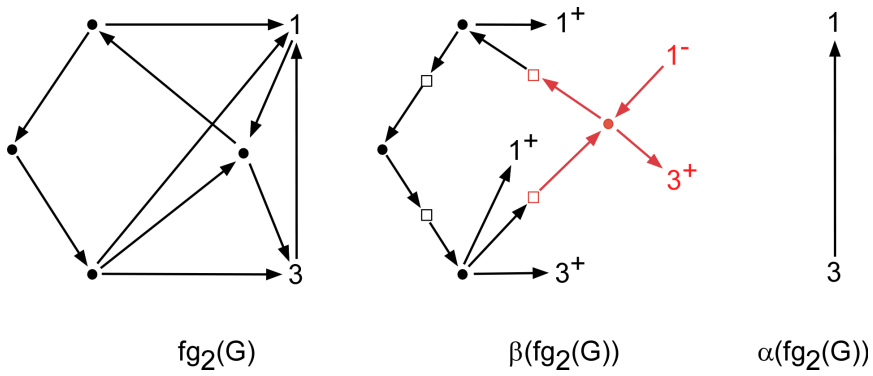

Fig. A.2. The graphs $fg_2(G), \beta(fg_2(G))$ and $\alpha(fg_2(G))$ for G of Fig. A.1.

3) for each vertex u of $\beta(G)$ labelled by a^+, we add an edge: $u \to \overline{x}$, and we relabel u by $\square$,

4) for each edge of $\alpha(G)$ from the a-source to some b-source, we add a vertex labelled by b^+ and an edge from $\overline{x}$ to this vertex,

5) for each edge of $\alpha(G)$ from some b-source to the a-source, we add a vertex labelled by b^- and an edge from this vertex to $\overline{x}$,

6) for each loop of $\alpha(G)$ incident with the a-source, we add a vertex labelled by $\square$, an edge from $\overline{x}$ to this vertex and an edge from it to $\overline{x}$.

Figure A.2 illustrates this description. The vertices and edges in red are added to $\beta(G)$ (or labels are modified) to build $\beta(fg_2(G))$.

We now express the construction of $\beta(fg_a(G))$ in Claim 2 (ii) with clique-width operations using the sets of label $C := \{\bullet, *\}$ and D. We denote $relab_h$ by $relab_{a\to b}$ if h only changes a into b.

We define the following labelled graphs:

$H := relab_{a^-\to\square}(relab_{a^+\to\square}(\overrightarrow{add}_{*,a^-}(\overrightarrow{add}_{a^+,*}(\beta(G) \oplus *))))$.

This graph H is obtained by performing steps 2) and 3) of Claim 2 (ii); $\overline{x}$ is introduced by $*$. Note that a^- and a^+ do not belong to $\pi(H)$, defined as the set of labels of the vertices of H. For implementing steps 4) to 6) we take:

$H' := Add_{e_p}(....(Add_{e_1}(H))...))$ where $e_1, ..., e_p$ are the edges of $\alpha(G)$ incident with the a-source, and for each such edge e and labelled graph X, we define:

$Add_e(X) := relab_{a^+\to b^+}(\overrightarrow{add}_{*,a^+}(X \oplus \mathbf{a}^+))$ if e goes from the a-source to the b-source,

$Add_e(X) := relab_{a^+\to b^-}(\overrightarrow{add}_{a^+,*}(X \oplus \mathbf{a}^+))$ if e goes from the b-source to

the a-source,

$Add_e(X) := relab_{a^+\to\square}(\overrightarrow{add}_{a^+,*}(\overrightarrow{add}_{*,a^+}(X \oplus \mathbf{a}^+)))$ if e is a loop incident to the a-source.

Here, a^+ is used as an auxiliary (temporary) label: it does not belong to the sets $\pi(X)$ for any of the graphs X we use to construct H'. We could use a^- instead or an extra label (but we wish to use as few labels as possible). Finally, we have :

$\beta(fg_a(G)) = relab_{*\to\bullet}(H')$.

Note that $*$ is an auxiliary label, with no occurrence in the graphs $\beta(G)$ and $\beta(fg_a(G))$. To sum up, we have

$\beta(fg_a(G)) = B_{\alpha(G)}[\beta(G)]$,

where $B_{\alpha(G)}$ is a sequence of operations that depends only on $\alpha(G)$.

In the case of Figure A.2 we have :

$\beta(fg_2(G)) =$

$relab_{*\to\bullet}(relab_{2^+\to 3^+}(\overrightarrow{add}_{*,2^+}(relab_{2^+\to 1^-}(\overrightarrow{add}_{2^+,*}(H \oplus \mathbf{2}^+)) \oplus \mathbf{2}^+)))$

where

$H := relab_{2^-\to\square}(relab_{2^+\to\square}(\overrightarrow{add}_{*,2^-}(\overrightarrow{add}_{2^+,*}(\beta(G) \oplus *))))$,

hence $B_{\alpha(G)}[X]$ is

$relab_{*\to\bullet}(relab_{2^+\to 3^+}(...(relab_{2^-\to\square}(...(X \oplus *)))) \oplus \mathbf{2}^+)) \oplus \mathbf{2}^+)))$.

Theorem A.1 *: (1) Let G be a directed graph of tree-width at most k. Then $Inc(G)$ is defined by a term in $T(F_{C,D})$ where $|C| = 2$ and $|D| = 2k + 3$. The clique-width of $Inc(G)$ is at most $2k + 4$.*

If G is undirected, we have the same result with $|C| = 2$ and $|D| = k + 2$ and $cwd(Inc(G)) \leq k + 3$.

(2) In all cases, $twd(G) = O(cwd(Inc(G))$.

Proof:

Claim : Let K be finite and $C := \{\bullet, *\}$, $D := \{c^+, c^- \mid c \in K\} \cup \{\square\}$. From every term $t \in T(L_K)$, one can construct in linear time a term $\gamma(t) \in T(F_{C,D})$ that defines $\beta(\widehat{G}(t))$.

Proof of the claim : By induction on the structure of t, we construct simultaneously $\tau(\widehat{G}(t))$, $\alpha(\widehat{G}(t))$ and $\gamma(t)$. We use Claim 2, and, in particular, for γ:

$\gamma(t_1//t_2) := \gamma(t_1) \oplus \gamma(t_2)$,
$\gamma(fg_a(t_1)) := \gamma(t_1)$ if $a \notin \tau(val(t_1))$,
$\gamma(fg_a(t_1)) := B_{\alpha(val(t_1))}[\gamma(t_1)]$ if $a \in \tau(\widehat{G}(t_1))$ by Claim 2 (ii).
$\gamma(\mathbf{a})$, $\gamma(\mathbf{a}^\ell)$ and $\gamma(\overrightarrow{\mathbf{ab}})$ are $\varnothing$, another nullary symbol denoting the empty graph.

Inductive rules for $\tau(G(t)\ )$ and $\alpha(G(t)\ )$ are easy.□

Main proof: (1) Let G be a graph of tree-width at most k without sources. We apply this claim to a term $t \in T(L_K)$ representing a tree-decomposition of G of width at most k where K has cardinality $k+1$, C has cardinality 2 and D cardinatity $2(k+1)+1$. We get a term $\gamma(t) \in T(F_{C,D})$ that defines $\beta(\widehat{G}(t)) = Inc(\widehat{G}(t)) = Inc(G)$.

In this construction, we distinguish the vertices of V_G from those of EV_G by the two labels $\bullet$ and $\square$. This is motivated by our constructions of automata. If we are only interested in bounding the clique-width, we can identify $\square$ and $\bullet$: it follows that $cwd(Inc(G)) \leq 2.twd(G)+4$.

For undirected graphs, we can use c instead of c^+ and c^-. If furthermore we identify $\square$ and $\bullet$, we can construct $Inc(G)$ with the set of labels $K \cup \{*, \bullet\}$, hence $cwd(Inc(G)) \leq twd(G)+3$.

(2) If G be undirected, then $Inc(G)$ is undirected and has no subgraph isomorphic to $K_{3,3}$. By a result due to Gurski and Wanke ([6], Proposition 2.115) $twd(Inc(G)) \leq 6.cwd(Inc(G))-1$. But $twd(Inc(G))$ is $twd(G)$ or $twd(G)+1$. If G is directed, the undirected graph H obtained from $Inc(G)$ by omitting edge directions has no subgraph isomorphic to $K_{3,3}$. We have $twd(Inc(G)) = twd(H) \leq 6.cwd(H)-1 \leq 6.cwd(Inc(G))-1$. (T. Bouvier [1] has proved that : $twd(H) \leq 2.cwd(H)-1$). □

Assertion (2) shows that our method needs the condition that the input graphs have bounded tree-width.

C : Some constructions of automata.

If G is a graph, we denote by E_G its set of edges and by EV_G the set of vertices of $Inc(G)$ that represent the edges of G. Of course, E_G and EV_G are in bijection, but conceptually, these sets are distinct.

C.1 *The automaton* $\mathcal{A}_{CT}$, for checking that a term is *correct*.

Every term $t \in T(F_{C,D})$ defines a directed bipartite graph $G(t)$ that is an incidence graph iff its vertices labelled in D have indegree and outdegree 1. A state of $\mathcal{A}_{CT}$ is either *Error* or a 6-tuple $(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11}) \in \mathcal{P}(C)^2 \times \mathcal{P}(D)^4$. At the root of a term $t \in T(F_{C,D})$, $\mathcal{A}_{CT}$ reaches the state $(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11})$ if and only if:

γ_1 is the set of labels in C that label a single vertex,
γ_2 is the set of labels in C that label at least two vertices (hence $\gamma_1 \cap \gamma_2 = \emptyset$),
δ_{00} is the set of labels in D of isolated vertices,
δ_{01} is the set of labels in D of vertices of indegree 0, outdegree 1,
δ_{10} is the set of labels in D of vertices of indegree 1, outdegree 0,
δ_{11} is the set of labels in D of vertices of indegree 1, outdegree 1,
and no vertex labelled in D has indegree or outdegree 2 or more.

It reaches the state *Error* if and only if some vertex labelled in D has indegree or outdegree 2 or more. The accepting states are the tuples $(\gamma_1, \gamma_2, \emptyset, \emptyset, \emptyset, \delta_{11})$. Its transitions are as follows:

1) $\oplus[Error, q] \to Error$ and $\oplus[q, Error] \to Error$ for all states q.
$\oplus[(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11}), (\gamma'_1, \gamma'_2, \delta'_{00}, \delta'_{01}, \delta'_{10}, \delta'_{11})] \to$
$(\overline{\gamma}_1, \overline{\gamma}_2, \delta_{00} \cup \delta'_{00}, \delta_{01} \cup \delta'_{01}, \delta_{10} \cup \delta'_{10}, \delta_{11} \cup \delta'_{11})$
where $\overline{\gamma}_1 := (\gamma_1 - (\gamma'_1 \cup \gamma'_2)) \cup (\gamma'_1 - (\gamma_1 \cup \gamma_2))$ and $\overline{\gamma}_2 := \gamma_2 \cup \gamma'_2 \cup (\gamma_1 \cap \gamma'_1)$.

2) $relab_h[Error] \to Error$ and
$relab_h[(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11})] \to (\overline{\gamma}_1, \overline{\gamma}_2, h(\delta_{00}), h(\delta_{01}), h(\delta_{10}), h(\delta_{11}))$
where $\overline{\gamma}_1$ is the set of labels in $h(\gamma_1)$ that are the image of a single label in γ_1, and $\overline{\gamma}_2 := h(\gamma_1 \cup \gamma_2) - \overline{\gamma}_1$.

For $c \in C$ and $d \in D$:

3) $\mathbf{c} \to (\{c\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ and $\mathbf{d} \to (\emptyset, \emptyset, \{d\}, \emptyset, \emptyset, \emptyset)$.

4) $\overrightarrow{add}_{c,d}[Error] \to Error$
$\overrightarrow{add}_{c,d}[(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11})] \to q$ where the following holds:
if $c \notin \gamma_1 \cup \gamma_2$ or $d \notin \delta_{00} \cup \delta_{01} \cup \delta_{10} \cup \delta_{11}$, then
$q := (\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11})$;
otherwise [then $c \in \gamma_1 \cup \gamma_2$, $d \in \delta_{00} \cup \delta_{01} \cup \delta_{10} \cup \delta_{11}$],

if $c \in \gamma_2$ or $d \in \delta_{11} \cup \delta_{10}$, then $q := Error$;
otherwise $[c \in \gamma_1$ and $d \in \delta_{00} \cup \delta_{01} - (\delta_{11} \cup \delta_{10})]$ we have :
$q := (\gamma_1, \gamma_2, \overline{\delta}_{00}, \overline{\delta}_{01}, \overline{\delta}_{10}, \overline{\delta}_{11})$ where:
$\overline{\delta}_{00} := \delta_{00} - \{d\}$
$\overline{\delta}_{10} :=$ `if` $d \in \delta_{00}$ `then` $\delta_{10} \cup \{d\}$ `else` δ_{10}
$\overline{\delta}_{01} := \delta_{01} - \{d\}$
$\overline{\delta}_{11} :=$ `if` $d \in \delta_{01}$ `then` $\delta_{11} \cup \{d\}$ `else` δ_{11}.

5) The case of $\overrightarrow{add}_{d,c}$ is fully similar.

This construction is intended for *irredundant terms*: we mean by this that an operation $\overrightarrow{add}_{c,d}$ never tries to create an edge from a vertex x labelled by c to a vertex u labelled by d if we already have $x \to u$ (and similarly for $\overrightarrow{add}_{d,c}$ with $c \in C$ and $d \in D$). Every term can be made irredundant by an easy preprocessing [4,6]. This condition insures the correctness of the transition $\overrightarrow{add}_{c,d}[(\gamma_1, \gamma_2, \delta_{00}, \delta_{01}, \delta_{10}, \delta_{11})] \to Error$ when $c \in \gamma_1$ and $d \in \delta_{11} \cup \delta_{10}$.

The number of states evaluated in terms of $k := |C| + |D|$ may seem large, but actually, for FA, the maximal size of a state in the run on an input term matters more than the number of states (that may actually be infinite, see footnote 2, Section 2). This size is here $O(k)$ (with no huge hidden constant).

C.2 *The automata for* $edg(X,Y)$, $Link^{\exists\exists}(X,Y)$, $Link^{\forall\exists}(X,Y)$ and $Link^{\forall\forall}(X,Y)$.

All automata constructed below are intended to run on correct and irredundant terms. Correctness can be checked by $\mathcal{A}_{CT}$ and irredundancy by another FA [4]. $\mathcal{P}_{\leq 1}(C)$ is the set of subsets of C with at most one element.

C.2.1 *The deterministic FA* $\mathcal{A}_{edg(X,Y)}$ *for checking* $edg(X,Y)$.

The property $edg(X,Y)$ takes as arguments an incidence graph $G(t)$ and two sets of vertices X and Y of it, labelled in C. In a term t, X and Y are encoded by a pair (i,j) of Booleans (0 or 1) attached to each occurrence w of a constant in C. Actually, w can be considered as (or even *is*) the vertex defined by this occurrence, $i = 1$ iff $w \in X$, and $j = 1$ iff $w \in Y$. (As an easy example, the automaton checking the property $X \subseteq Y$ need only verify that no pair (1,0) occurs.)

The states of $\mathcal{A}_{edg(X,Y)}$ are Ok, $Error$ and the tuples $(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2) \in \mathcal{P}_{\leq 1}(C)^2 \times \mathcal{P}(D)^3$.

Consider a correct and irredundant term $t \in T(F_{C,D})$ equiped with pairs of Booleans. It defines an incidence graph $G(t) = Inc(H(t))$ and a pair X, Y of sets of vertices of $H(t)$. Let t' be a subterm of t. It is irredundant and defines a bipartite graph $G(t')$ that may not be an incidence graph. Let $X' = X \cap V_{G(t')}, Y' = Y \cap V_{G(t')}$.

At the root of t', $\mathcal{A}_{edg(X,Y)}$ reaches the following state:

the state Ok iff X' and Y' are singletons $\{x\}$ and $\{y\}$ and $x \to y$,

the state $Error$ iff X' or Y' has cardinality 2 or more,

the state $(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)$ if and only if:

$\gamma_1 = \pi_{G(t')}(X')$ and $|X'| \leq 1$, ($\pi_{G(t')}(X')$ is the set labels in $G(t')$ of the vertices in X'),

$\gamma_2 = \pi_{G(t')}(Y')$ and $|Y'| \leq 1$,

δ_0 is the set of labels in D of isolated vertices (they are in $EV_{H(t)}$),

δ_1 is the set of labels in D of vertices u in $EV_{H(t)}$ such that, in $G(t')$, $X' \to u \nrightarrow Y'$ (which means there is an edge from a vertex of X' to u and no edge from u to any vertex of Y'),

δ_2 is the set of labels in D of vertices u in $EV_{H(t)}$ such that, in $G(t')$, $X' \nrightarrow u \to Y'$

and the conditions for Ok do not hold.

The accepting state is Ok. Some transitions are given below (the others easy or similar).

$\oplus[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2), Error] \to Error$,

$\oplus[Ok, Ok] \to Error$ (because of cardinality conditions),

$\oplus[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2), (\gamma'_1, \gamma'_2, \delta'_0, \delta'_1, \delta'_2)] \to q$ where

$q := Error$ if $|\gamma_1| + |\gamma'_1| \geq 2$ or $|\gamma_2| + |\gamma'_2| \geq 2$ and

$q := (\gamma_1 \cup \gamma'_1, \gamma_2 \cup \gamma'_2, \delta_0 \cup \delta'_0, \delta_1 \cup \delta'_1, \delta_2 \cup \delta'_2)$ otherwise.

$\overrightarrow{add}_{c,d}[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)] \to q$ where the following holds:

if $\gamma_1 \neq \{c\}$ or $d \notin \delta_0 \cup \delta_2$, then $q := (\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)$

else [then $\gamma_1 = \{c\}, d \in \delta_0 \cup \delta_2$]

if $d \in \delta_2$ then $q := Ok$,

else [then $\gamma_1 = \{c\}, d \in \delta_0 - \delta_2$] $q := (\gamma_1, \gamma_2, \delta_0 - \{d\}, \delta_1 \cup \{d\}, \delta_2)$.

Remarks : (1) Since this automaton is intended to run on a correct and irredundant term t, it needs no transition of the form $\overrightarrow{add}_{c,d}[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)] \to q$

where $\gamma_1 = \{c\}$, $d \in \delta_1 - (\delta_0 - \delta_2)$. Otherwise, the graph $G(t)$ would have a vertex labelled in D of indegree ≥ 2.

(2) The role of *Error* is to shorten the computation when X or Y is too large.

C.2.2 The deterministic fly-automaton $\mathcal{A}_{Link^{\exists\exists}(X,Y)}$ that checks the property $Link^{\exists\exists}(X,Y)$.

It is similar to $\mathcal{A}_{edg(X,Y)}$ but the differences are interesting: it is simpler because it need not check that X and Y are singletons but it has more states. Its states are *Success* and the tuples $(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2) \in \mathcal{P}(C)^2 \times \mathcal{P}(D)^3$. It is intended to run on correct and irredundant terms $t \in T(F_{C,D})$ equiped with pairs of Booleans that define incidence graphs $Inc(H(t))$ and pairs X, Y of sets of vertices of $H(t)$. Let t' be a subterm of t, $X' = X \cap V_{G(t')}, Y' = Y \cap V_{G(t')}$ as in C.2.1. At the root of t', $\mathcal{A}_{Link^{\exists\exists}(X,Y)}$ reaches the following state:

the state *Success* iff $X' \to Y'$,
the state $(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)$ if and only if:
$\gamma_1 = \pi_{G(t')}(X')$,
$\gamma_2 = \pi_{G(t')}(Y')$,
δ_0 is the set of labels in D of isolated vertices (they are in $EV_{H(t)} \cap V_{G(t')}$),
δ_1 is the set of labels in D of vertices u in $EV_{H(t)} \cap V_{G(t')}$ such that, in $G(t')$, $X \to u \nrightarrow Y$,
δ_2 is the set of labels in D of vertices u in $EV_{H(t)} \cap V_{G(t')}$ such that $X \nrightarrow u \to Y$,
and the conditions for *Success* do not hold.

The accepting state is *Success*. Some transitions are given below (the others are similar or easy).

$\oplus[Success, q] \to Success$ for any state q,
$\oplus[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2), (\gamma'_1, \gamma'_2, \delta'_0, \delta'_1, \delta'_2)] \to q$ where
$q := (\gamma_1 \cup \gamma'_1, \gamma_2 \cup \gamma'_2, \delta_0 \cup \delta'_0, \delta_1 \cup \delta'_1, \delta_2 \cup \delta'_2)$ (no *Error* case),
$\overrightarrow{add}_{c,d}[Success] \to Success$,
$\overrightarrow{add}_{c,d}[(\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)] \to q$ where the following holds:
if $c \notin \gamma_1$ or $d \notin \delta_0 \cup \delta_2$, then $q := (\gamma_1, \gamma_2, \delta_0, \delta_1, \delta_2)$;
otherwise $[c \in \gamma_1, d \in \delta_0 \cup \delta_2]$

if $d \in \delta_2$ then $q := Success$,
else $[c \in \gamma_1, d \in \delta_0 - \delta_2]$ $q := (\gamma_1, \gamma_2, \delta_0 - \{d\}, \delta_1 \cup \{d\}, \delta_2)$.

C.2.3 The deterministic fly-automaton $\mathcal{A}_{Link^{\forall\forall}(X,Y)}$ that checks the property $Link^{\forall\forall}(X, Y)$.

We use the same hypotheses and notation as in the previous case.

The state at the root of t' will contain the following relation:

$\theta := \{(\pi_{G(t')}(x), \pi_{G(t')}(y)) \mid x \to X', y \to Y', x \nrightarrow y\}$.

A state will be accepting iff this relation is empty. However, for defining the transition rules (they implement inductive computations), additional information is needed.

Let G be a directed bipartite graph. If $x \in V_G$, we let $Out_G(x)$ be the set of vertices u of outdegree 0 such that $x \to u$ and $In_G(x)$ the set of vertices u of indegree 0 such that $u \to x$.

A state is a tuple $(\gamma, \delta, \Delta, \Lambda, \Theta)$ belonging to

$$\mathcal{P}(C) \times \mathcal{P}(D) \times \mathcal{P}(C \times \mathcal{P}(D)) \times \mathcal{P}(\mathcal{P}(D) \times C) \times \mathcal{P}(C \times \mathcal{P}(D) \times \mathcal{P}(D) \times C).$$

The state at the root of t' will be $(\gamma, \delta, \Delta, \Lambda, \Theta)$ such that :

$\gamma = \pi(G(t')) \cap C$,
δ is the set of labels in D of isolated vertices (they are in $EV_{H(t)} \cap V_{G(t')}$),
$\Delta = \{(\pi_{G(t')}(x), \pi_{G(t')}(Out_{G(t')}(x))) \mid x \in X'\}$,
$\Lambda = \{(\pi_{G(t')}(In_{G(t')}(x)), \pi_{G(t')}(x)) \mid x \in Y'\}$,
$\Theta = \{(\pi_{G(t')}(x), \pi_{G(t')}(Out_{G(t')}(x)), \pi_{G(t')}(In_{G(t')}(y)), \pi_{G(t')}(y)) \mid$
$x \in X', y \in Y', x \nrightarrow y\}$.

The accepting states are those whose component Θ is empty.

Note that $\Theta \subseteq \Delta \times \Lambda$ and that θ can be computed from Θ. More notation will be useful:

$\gamma(\Delta) := \{c \in C \mid (c, \eta) \in \Delta \text{ for some } \eta\}$,
$\gamma(\Lambda) := \{c \in C \mid (\eta, c) \in \Lambda \text{ for some } \eta\}$.

Some representative transitions are as follows:

$\oplus[(\gamma_1, \delta_1, \Delta_1, \Lambda_1, \Theta_1), (\gamma_2, \delta_2, \Delta_2, \Lambda_2, \Theta_2)] \to$
$(\gamma_1 \cup \gamma_2, \delta_1 \cup \delta_2, \Delta_1 \cup \Delta_2, \Lambda_1 \cup \Lambda_2, \overline{\Theta})$ where
$\overline{\Theta} := \Theta_1 \cup \Theta_2 \cup \{(c, \eta, \eta', c') \mid$
$((c, \eta) \in \Delta_1 \text{ and } (\eta', c') \in \Lambda_2) \text{ or } ((c, \eta) \in \Delta_2 \text{ and } (\eta', c') \in \Lambda_1))\}$.

$\mathbf{c}(1, 1) \to (\{c\}, \emptyset, \{(c, \emptyset)\}, \{(\emptyset, c)\}, \{(c, \emptyset, \emptyset, c)\})$,

$\mathbf{c}(1,0) \to (\{c\}, \emptyset, \{(c,\emptyset)\}, \emptyset, \emptyset)$,
$\mathbf{c}(0,0) \to (\{c\}, \emptyset, \emptyset, \emptyset, \emptyset)$,
$\mathbf{d} \to (\emptyset, \{d\}, \emptyset, \emptyset, \emptyset)$.

$\overrightarrow{add}_{c,d}[(\gamma, \delta, \Delta, \Lambda, \Theta)] \to q$ where the following holds:

if $c \notin \gamma$ or d does not occur in $(\delta, \Delta, \Lambda)$, then $q := (\gamma, \delta, \Delta, \Lambda, \Theta)$,

otherwise , $q := (\gamma, \overline{\delta}, \overline{\Delta}, \overline{\Lambda}, \overline{\Theta})$, where $\overline{\delta}, \overline{\Delta}, \overline{\Lambda}, \overline{\Theta}$ are defined respectively as follows from $\delta, \Delta, \Lambda, \Theta$:

$\overline{\delta} := \delta - \{d\}$, (there is no change if $d \notin \delta$);

$\overline{\Delta}$: if $d \in \delta$ then, each pair (c, η) in Δ is replaced by $(c, \eta \cup \{d\})$; otherwise, $\overline{\Delta} := \Delta$;

$\overline{\Lambda}$: each pair (η', c') in Λ is replaced by $(\eta' - \{d\}, c')$;

$\overline{\Theta}$ is defined as follows:

every tuple of the form (c, η, η', c') is deleted if $d \in \eta'$,

otherwise, if $d \in \delta$, it is replaced by $(c, \eta \cup \{d\}, \eta', c')$.

Remarks : The tuples (a, η, η', c') in Θ such that $a \neq c$ are not modified.

We cannot have $c \in \gamma$ and d occurring in a pair of Δ because the input term t is assumed irredundant and correct.

C.2.4 The deterministic fly-automaton $\mathcal{A}_{Link^{\forall\exists}(X,Y)}$ that checks the property $Link^{\forall\exists}(X, Y)$.

The construction is similar to the previous one. The set of states is :

$\mathcal{P}(C)^2 \times \mathcal{P}(D)^2 \times \mathcal{P}(C \times \mathcal{P}(D))^2$.