Acyclicity and Finite Linear Extendability: a Formal and Constructive Equivalence

Stéphane Le Roux*

LIP, École normale supérieure de Lyon, France
stephane.le.roux@ens-lyon.fr

Abstract. Linear extension of partial orders was addressed in the late 1920's. Its computer-oriented version, *i.e.*, topological sorting of finite partial orders, arose in the late 1950's. However, those issues have not yet been considered from a viewpoint both formal and constructive; this paper discusses a few related claims formally proved with the constructive proof assistant Coq. For example it states that a given decidable binary relation is acyclic and equality is decidable on its domain *iff* an irreflexive linear extension can be computed uniformly for any of its finite restriction.

Keywords: Binary relation, finite restriction, linear extension, (non-)uniform computability, topological sorting, constructivism, induction, proof assistant.

1 Introduction

This paper is a shortened version of the research report [8]. The report is readable by a mathematician unfamiliar with constructive and computability issues and Coq formalism; this paper assumes basic knowledge in those fields.

1.1 Transitive Closure, Linear Extension, and Topological Sorting

The calculus of binary relations was developed by De Morgan around 1860. The notion of transitive closure of a binary relation (smallest transitive binary relation including a given binary relation) was defined in different manners by different people about 1890. See Pratt [9] for a historical account. In 1930, Szpilrajn [10] proved that, assuming the axiom of choice, any partial order has a linear extension, *i.e.*, is included in some total order. The proof invokes a notion close to transitive closure. Szpilrajn acknowledged that Banach, Kuratowsky, and Tarski had found unpublished proofs of the same result. In the late 1950's, The US Navy [2] designed PERT (Program Evaluation Research Task or Project Evaluation Review Techniques) for management and scheduling purpose. This tool partly consists in splitting a big project into small jobs on a chart and expressing with arrows when one job has to be done before another one can start

* This research was partly supported by Collège Doctoral Franco-Japonais

up. In order to study the resulting directed graph, Jarnagin [4] introduced a finite and algorithmic version of Szpilrajn's result. This gave birth to the widely studied topological sorting issue, which spread to the industry early 1960's (see [7] and [5]). Some technical details and computer-oriented examples can be found in Knuth's book [6].

1.2 Contribution

This paper revisits a few folklore results involving transitive closure, excluded middle, computability, linear extension, and topological sorting. Most of the properties are logical equivalences instead of one-way implications, which suggests maximal generality. Claims have been fully formalized (and proved) in Coq and then slightly modified in order to fit in the Coq-related CoLoR library [3].

In this paper, a binary relation over an arbitrary set is said to be middle-excluding if for any two elements in the set, either they are related or they are not. The intermediate result of this paper implies that in an arbitrary set with decidable (resp. middle-excluding) equality, a binary relation is decidable (resp. middle-excluding) *iff* the transitive closures of its finite restrictions are uniformly decidable (resp. middle-excluding). The main result splits into two parts, one on excluded middle and one on computability: First, consider a middle-excluding relation. It is acyclic and equality on its domain is middle-excluding *iff* its restriction to any finite set has a middle-excluding irreflexive linear extension. Second, consider R a decidable binary relation over A. The following three propositions are equivalent. Note that computability of linear extensions is non-uniform in the second proposition but uniform in the third one.

- Equality on A is decidable and R is acyclic.
- Equality on A is decidable and every finite restriction of R has a decidable linear extension.
- There exists a computable function that waits for finite restrictions of R and returns (decidable) linear extensions of them.

This paper follows the structure of the underlying Coq development but some straightforward results are omitted. Proofs are written in plain English. The main result in this paper relies on an intermediate one, and is itself invoked in a game theoretic proof (in Coq) not published yet.

1.3 Contents

Some basic objects from the Coq Library [1] are not defined in this paper. Section 2 talks about excluded middle and decidability and section 3 about lists. Section 4 discusses the notion of transitive closure, irreflexivity, and finite restrictions of a binary relation. Section 5 defines paths with respect to a binary relation and proves their correspondence with transitive closure. It also defines bounded paths that are proved to preserve decidability and middle-exclusion properties of the original relation. Since bounded paths and paths are by some

means equivalent on finite sets, subsection 5.4 states the intermediate result. Subsection 6.1 defines relation totality over finite sets. Subsections 6.2 to 6.5 define an acyclicity-preserving conditional single-arc addition (to a relation), and an acyclicity-preserving multi-stage arc addition over finite sets, which consists in repeating in turn single-arc addition and transitive closure. This procedure helps state equivalences for linear extension in 6.6 and topological sorting in 6.7.

1.4 Convention

Let A be a *Set*. Throughout this paper x, y, z, and t implicitly refer to objects of type A. In the same way R, R', and R'' refer to binary relations over A; l, l', and l'' to lists over A, and n to natural numbers. For the sake of readability, types will sometimes be omitted according to the above convention, even in formal statements where Coq could not infer them. The notation $\neg P$ stands for $P \to$ *False*, $x \neq y$ for $x = y \to$ *False*, and $\nexists x,\ P$ for $(\exists x,\ P) \to$ *False*.

2 On Excluded Middle and Decidability

The following two definitions respectively say that equality on A is middle-excluding and that a given binary relation over A is middle-excluding.

Definition $eq\_midex := \forall\ x\ y,\ x = y \vee x \neq y$.
Definition $rel\_midex\ R := \forall\ x\ y,\ R\ x\ y \vee \neg R\ x\ y$.

The next two definitions respectively say that equality on A is decidable and that a given binary relation over A is decidable.

Definition $eq\_dec := \forall\ x\ y,\ \{x = y\} + \{x \neq y\}$.
Definition $rel\_dec\ R := \forall\ x\ y,\ \{R\ x\ y\} + \{\neg R\ x\ y\}$.

The following two lemmas justify the representation of decidability used in this paper.

Lemma $rel\_dec\_bool : \forall\ R$,
$rel\_dec\ R \to \{f : A \to A \to bool \mid \forall\ x\ y : A,\ if\ f\ x\ y\ then\ R\ x\ y\ else\ \neg R\ x\ y\}$.

Lemma $bool\_rel\_dec : \forall\ R$,
$\{f : A \to A \to bool \mid \forall\ x\ y : A,\ if\ f\ x\ y\ then\ R\ x\ y\ else\ \neg R\ x\ y\} \to rel\_dec\ R$.

"Decidability implies excluded middle", as shown below.

Lemma $eq\_dec\_midex : eq\_dec \to eq\_midex$.
Lemma $rel\_dec\_midex : rel\_dec \to rel\_midex$.

3 On Lists

If equality is middle-excluding on A and if an element occurs in a list built over A, then the list can be decomposed into three parts: a list, one occurrence of the element, and a second list where the element does not occur.

Lemma *In_elim_right* : *eq_midex* → ∀ *x l*,
In x l → ∃ *l'*, ∃ *l"*, *l*=*l'*++(*x*::*l"*) ∧ ¬*In x l"*.

Proof By induction on *l*. For the inductive case, case split on *x* occurring in the head or the tail of *l*. □

The predicate *repeat_free* says that no element occurs more than once in a given list. It is defined by recursion on its sole argument.

Fixpoint *repeat_free l* : *Prop* :=
match l with
— *nil* ⇒ *True*
— *x*::*l'* ⇒ ¬*In x l'* ∧ *repeat_free l'*
end.

If equality is middle-excluding on *A* then a *repeat_free* list included in another list is not longer than the other list. This is proved by induction on the *repeat_free* list. For the inductive step, invoke *In_elim_right* to decompose the other list along the head of the *repeat_free* list.

Lemma *repeat_free_incl_length* : *eq_midex* → ∀ *l l'*,
repeat_free l → *incl l l'* → *length l*≤*length l'*.

4 On Relations

4.1 Transitive Closure in the Coq Standard Library

Traditionally, the transitive closure of a binary relation is *the* smallest transitive binary relation including the original relation. The notion of transitive closure can be formally defined by induction, like in the Coq Standard Library. The following function *clos_trans* waits for a relation over *A* and yields its transitive closure, which is also a relation over *A*.

Inductive *clos_trans R* : *A* → *A* → *Prop* :=
— *t_step* : ∀ *x y*, *R x y* → *clos_trans R x y*
— *t_trans* :
∀ *x y z*, *clos_trans R x y* → *clos_trans R y z* → *clos_trans R x z*.

Intuitively, two elements are related by the transitive closure of a binary relation if one can start at the first element and reach the second one in finitely many steps of the original relation. Therefore replacing *clos_trans R x y* → *clos_trans R y z* → *clos_trans R x z* by *R x y* → *clos_trans R y z* → *clos_trans R x z* or *clos_trans R x y* → *R y z* → *clos_trans R x z* in the definition of *clos_trans* would yield two relations coinciding with *clos_trans*. Those three relations are yet different in intension: only *clos_trans* captures the meaning of the terminology "transitive closure".

In addition, this paper needs the notion of subrelation.

Definition *sub_rel R R'* : *Prop* := ∀ *x y*, *R x y* → *R' x y*.

The next lemma asserts that a transitive relation contains its own transitive closure (they actually coincide).

Lemma *transitive_sub_rel_clos_trans* : $\forall$ R,
transitive $R \rightarrow$ *sub_rel* (*clos_trans* R) R.

Proof Let R be a transitive relation over A. Prove the subrelation property by the induction principle of *clos_trans*. The base case is trivial and the inductive case follows from the transitivity of R. □

4.2 Irreflexivity

A relation is irreflexive if no element is related to itself. Therefore irreflexivity of a relation implies irreflexivity of any subrelation.

Definition *irreflexive* R : *Prop* := $\forall$ x, $\neg R\ x\ x$.

Lemma *irreflexive_preserved* : $\forall$ R R',
sub_rel R R' $\rightarrow$ *irreflexive* R' $\rightarrow$ *irreflexive* R.

4.3 Restrictions

Throughout this paper, finite "subsets" of A are represented by lists over A. For that specific use of lists, the number and the order of occurrences of elements in a list are irrelevant. Let R be a binary relation over A and l be a list over A. The binary relation *restriction* R l relates elements that are both occurring in l and related by R. The predicate *is_restricted* says that "the support of the given binary relation R is included in the list l". And the next lemma shows that transitive closure preserves restriction to a given finite set.

Definition *restriction* R l x y : *Prop* := *In* x $l \wedge$ *In* y $l \wedge R\ x\ y$.

Definition *is_restricted* R l : *Prop* := $\forall$ x y, $R\ x\ y \rightarrow$ *In* x $l \wedge$ *In* y l.

Lemma *restricted_clos_trans* : $\forall$ R l,
is_restricted R $l \rightarrow$ *is_restricted* (*clos_trans* R) l.

Proof Assume that R is restricted to l. Let x and y in A be such that *clos_trans* R x y, and prove by induction on that last hypothesis that x and y are in l. The base case, where "*clos_trans* R x y comes from R x y", follows by definition of restriction. For the inductive case, where "*clos_trans* R x y comes from *clos_trans* R x z and *clos_trans* R z y for some z in A", induction hypotheses are *In* x $l \wedge$ *In* z l and *In* z $l \wedge$ *In* y l, which allows to conclude. □

If the support of a relation involves only two (possibly equal) elements, and if those two elements are related by the transitive closure, then they are also related by the original relation. By the induction principle for *clos_trans* and lemma *restricted_clos_trans*.

Lemma *clos_trans_restricted_pair* : $\forall$ R x y,
is_restricted R (x::y::*nil*) $\rightarrow$ *clos_trans* R x $y \rightarrow R\ x\ y$.

5 On Paths and Transitive Closure

5.1 Paths

The notion of path relates to one interpretation of transitive closure. Informally, a path is a list recording consecutive steps of a given relation. The following predicate says that a given list is a path between two given elements with respect to a given relation.

Fixpoint *is_path R x y l* {*struct l*} : *Prop* :=
match l with
— *nil* ⇒ *R x y*
— *z*::*l'* ⇒ *R x z* ∧ *is_path R z y l'*
end.

The following two lemmas show the correspondence between paths and transitive closure. The first is proved by the induction principle of *clos_trans* and an appending property on paths proved by induction on lists. For the second, let y be in A and prove ∀ *l x*, *is_path R x y l* → *clos_trans R x y* by induction on *l*. Note that the two lemmas imply ∀ *x y*, *clos_trans R x y* ↔ ∃ *l*, *is_path R x y l*.

Lemma *clos_trans_path* : ∀ *x y*, *clos_trans R x y* → ∃ *l*, *is_path R x y l*.

Lemma *path_clos_trans* : ∀ *y l x*, *is_path R x y l* → *clos_trans R x y*.

Assume that equality is middle-excluding on A and consider a path between two points. Between those two points there is a *repeat_free* path avoiding them and (point-wise) included in the first path. The inclusion is also arc-wise by construction, but this paper does not need it.

Lemma *path_repeat_free_length* : *eq_midex* → ∀ *y l x*,
is_path R x y l →
∃ *l'*, ¬*In x l'* ∧ ¬*In y l'* ∧ *repeat_free l'* ∧
length l'≤ *length l* ∧ *incl l' l* ∧ *is_path R x y l'*.

Proof Assume that equality is middle-excluding on A, let y be in A, and perform an induction on *l*. For the inductive step, call *a* the head of *l*. If *a* equals y then the empty list is a witness for the existential quantifier. Now assume that *a* and y are distinct. Use the induction hypothesis with *a* and get a list *l'*. Case split on x occurring in *l'*. If x occurs in *l'* then invoke lemma *In_elim_right* and decompose *l'* along x, and get two lists. In order to prove that the second list, where x does not occur, is a witness for the existential quantifier, notice that splitting a path yields two paths (*a priori* between different elements) and that appending reflects the *repeat_free* predicate (if the appending of two lists is *repeat_free* then the original lists also are). Next, assume that x does not occur in *l'*. If x equals *a* then *l'* is a witness for the existential quantifier. If x and *a* are distinct then *a*::*l'* is a witness. □

5.2 Bounded Paths

Given a relation and a natural number, the function *bounded_path* returns a relation saying that there exists a path of length at most the given natural number between two given elements.

Inductive $bounded\_path\ R\ n : A \to A \to Prop :=$
— $bp\_intro : \forall\ x\ y\ l,\ length\ l \leq n \to is\_path\ R\ x\ y\ l \to bounded\_path\ R\ n\ x\ y.$

Below, two lemmas relate *bounded_path* and *clos_trans*. The first one follows from *path_clos_trans*; the second one from *clos_trans_path*, *path_repeat_free_length*, *repeat_free_incl_length*, and a path of a restricted relation being included in the support of the relation. Especially, the second lemma says that in order to know whether two elements are related by the transitive closure of a restricted relation, it suffices to check whether there is, between those two elements, a path of length at most the "cardinal" of the support of the relation.

Lemma $bounded\_path\_clos\_trans : \forall\ R\ n,$
$sub\_rel\ (bounded\_path\ R\ n)\ (clos\_trans\ R).$

Lemma $clos\_trans\_bounded\_path : eq\_midex \to \forall\ R\ l,$
$is\_restricted\ R\ l \to sub\_rel\ (clos\_trans\ R)\ (bounded\_path\ R\ (length\ l))\ .$

5.3 Restriction, Decidability, and Transitive Closure

The following lemma says that it is decidable whether one step of a given decidable relation from a given starting point to some point z in a given finite set and one step of another given decidable relation from the same point z can lead to another given ending point. Moreover such an intermediate point z is computable when it exists, hence the syntax $\{z : A \mid \dots\}$.

Lemma $dec\_lem : \forall\ R'\ R''\ x\ y\ l,\ rel\_dec\ R' \to rel\_dec\ R'' \to$
$\{z : A \mid In\ z\ l \wedge R'\ x\ z \wedge R''\ z\ y\} + \{\nexists\ z : A,\ In\ z\ l \wedge R'\ x\ z \wedge R''\ z\ y\}.$

The following lemma is the middle-excluding version of the previous lemma.

Lemma $midex\_lem : \forall\ R'\ R''\ x\ y\ l,\ rel\_midex\ R' \to rel\_midex\ R'' \to$
$(\exists\ z : A,\ In\ z\ l \wedge R'\ x\ z \wedge R''\ z\ y) \vee (\nexists\ z : A,\ In\ z\ l \wedge R'\ x\ z \wedge R''\ z\ y).$

Proof By induction on l. For the inductive step, call a the head of l. Then case split on the induction hypothesis. In the case of existence, any witness for the induction hypothesis is also a witness for the wanted property. In the case of non-existence, case split on $R'\ x\ a$ and $R''\ a\ y$. □

By unfolding the definition *rel_midex*, the next result implies that given a restricted and middle-excluding relation, a given natural number and two given points, either there is a path of length at most that number between those points or there is no such path. Replacing *midex* by *dec* in the lemma yields a correct lemma about decidability.

Lemma $bounded\_path\_midex : \forall\ R\ l\ n,$

is_restricted R l → *rel_midex R* → *rel_midex* (*bounded_path R n*).

Proof First prove three simple lemmas relating *bounded_path*, *n*, and *S n*. Then let *R* be a middle-excluding relation restricted to *l* and *x* and *y* be in *A*. Perform an induction on *n*. For the inductive step, case split on the induction hypothesis with *x* and *y*. If *bounded_path R n x y* holds then it is straightforward. If its negation holds then case split on *em_lem* with *R*, *bounded_path R n*, *x*, *y*, and *l*. In the existence case, just notice that a path of length less than *n* is of length less than *S n*. In the non-existence case, show the negation of *bounded_path* in the wanted property. □

Let equality and a restricted relation be middle-excluding over *A*, then the transitive closure of the relation is also middle-excluding. The proof invokes *bounded_path_midex*, *bounded_path_clos_trans*, and *clos_trans_bounded_path*. The decidability version of it is also correct.

Lemma *restricted_midex_clos_trans_midex* : *eq_midex* → ∀ *R l*,
rel_midex R → *is_restricted R l* → *rel_midex* (*clos_trans R*).

5.4 Intermediate Results

The following theorems state the equivalence between decidability of a relation and decidability of the transitive closures of its finite restrictions. The first result invokes *clos_trans_restricted_pair* and the second implication uses *restricted_dec_clos_trans_dec*. Note that decidable equality is required only for the second implication. These results remain correct when considering excluded middle instead of decidability.

Theorem *clos_trans_restriction_dec_R_dec* : ∀ *R*
(∀ *l*, *rel_dec* (*clos_trans* (*restriction R l*))) → *rel_dec R*.

Theorem *R_dec_clos_trans_restriction_dec* : *eq_dec* → ∀ *R*
rel_dec R → ∀ *l*, *rel_dec* (*clos_trans* (*restriction R l*)).

6 Linear Extension and Topological Sorting

Consider *R* a binary relation over *A* and *l* a list over *A*. This section presents a way of preserving acyclicity of *R* while "adding arcs" to the restriction of *R* to *l* in order to build a total and transitive relation over *l*. In particular, if *R* is acyclic, then its image by the relation completion procedure must be a strict total order. The basic idea is to compute the transitive closure of the restriction of *R* to *l*, add an arc *iff* it can be done without creating any cycle, take the transitive closure, add an arc if possible, etc. All those steps preserve existing arcs, and since *l* is finite there are finitely many eligible arcs, therefore the process terminates. This is not the fastest topological sort algorithm but its fairly simple expression leads to a simple proof of correctness.

6.1 Total

R is said to be total on l if any two distinct elements in l are related either way. Such a trichotomy property for a relation implies trichotomy for any bigger relation.

Definition *trichotomy R x y* : *Prop* := $R\ x\ y \vee x{=}y \vee R\ y\ x$.

Definition *total R l* : *Prop* := $\forall\ x\ y$, *In x l* $\rightarrow$ *In y l* $\rightarrow$ *trichotomy R x y*.

Lemma *trichotomy_preserved* : $\forall$ *R R' x y*,
sub_rel R R' $\rightarrow$ *trichotomy R x y* $\rightarrow$ *trichotomy R' x y*.

6.2 Try Add Arc

If x and y are equal or related either way then define the relation *try_add_arc R x y* as R, else define it as the disjoint union of R and the arc (x,y).

Inductive *try_add_arc R x y* : $A \rightarrow A \rightarrow$ *Prop* :=
— *keep* : $\forall$ *z t*, *R z t* $\rightarrow$ *try_add_arc x y z t*
— *try_add* : $x{\neq}y \rightarrow \neg$*R y x* $\rightarrow$ *try_add_arc x y x y*.

Prove by induction on l and a few case splittings that, under some conditions, a path with respect to an image of *try_add_arc* is also a path with respect to the original relation.

Lemma *path_try_add_arc_path* : $\forall$ *R t x y l z*,
$\neg(x{=}z \vee$ *In x l*$) \vee \neg(y{=}t \vee$ *In y l*$) \rightarrow$
is_path R (*try_add_arc R x y*) *z t l* $\rightarrow$ *is_path R z t l*.

The next three lemmas lead to the conclusion that the function *try_add_arc* does not create cycles. The first one follows from a few case splittings and the last one highly relies on the second one but also invokes *clos_trans_path*.

Lemma *trans_try_add_arc_sym* : $\forall$ *R x y z t*,
transitive R $\rightarrow$ *try_add_arc x y z t* $\rightarrow$ *try_add_arc x y t z* $\rightarrow$ *R z z*.

Lemma *trans_bounded_path_try_add_arc* : *eq_midex* $\rightarrow \forall$ *R x y z n*,
transitive R $\rightarrow$ *bounded_path* (*try_add_arc x y*) *n z z* $\rightarrow$ *R z z*.

Proof By induction on *n*. The base case requires only *trans_try_add_arc_sym*. For the inductive case, consider a path of length less than or equal to *n+1* and build one of length less than *n+1* as follows. By *path_repeat_free_length* the path may be *repeat_free*, *i.e.*, without circuit. Proceed by case splitting on the construction of the path: when the path is *nil*, it is straightforward. If the length of the path is one then invoke *sub_rel_try_add_arc trans_try_add_arc_sym* else perform a 4-case splitting (induced by the disjunctive definition of*try_add_arc*) on the first two (*try_add_arc x y*)-steps of the path. Two cases out of the four need lemmas *transitive_sub_rel_clos_trans*, *path_clos_trans*, and *path_try_add_arc_path*. □

Lemma *try_add_arc_irefl* : *eq_midex* → ∀ *R x y*,
transitive R → *irreflexive R* → *irreflexive* (*clos_trans* (*try_add_arc x y*)).

6.3 Try Add Arc (One to Many)

The function *try_add_arc_one_to_many* recursively tries to (by preserving acyclicity) add all arcs starting at a given point and ending in a given list.

Fixpoint *try_add_arc_one_to_many R x l* {*struct l*} : *A* → *A* → *Prop* :=
match l with
— *nil* ⇒ *R*
— *y*::*l'* ⇒ *clos_trans* (*try_add_arc* (*try_add_arc_one_to_many R x l'*) *x y*)
end.

The following three lemmas prove preservation properties about the function *try_add_arc_one_to_many*: namely, arc preservation, restriction preservation, and middle-exclusion preservation. Decidability preservation is also correct, although not formally stated here.

Lemma *sub_rel_try_add_arc_one_to_many* : ∀ *R x l*,
sub_rel R (*try_add_arc_one_to_many R x l*).

Proof By induction on *l*. For the inductive step, call *a* the head of *l* and *l'* its tail. Use transitivity of *sub_rel* with *try_add_arc_one_to_many x l'* and *try_add_arc* (*try_add_arc_one_to_many x l'*) *x a*. Also invoke *clos_trans* and a similar arc preservation property for *try_add_arc*. □

Lemma *restricted_try_add_arc_one_to_many* : ∀ *R l x l'*, *In x l* → *incl l' l* →
is_restricted R l → *is_restricted* (*try_add_arc_one_to_many R x l'*) *l*.

Proof By induction on *l'*, *restricted_clos_trans*, and a similar restriction preservation property for *try_add_arc*. □

Lemma *try_add_arc_one_to_many_midex* :
eq_midex → ∀ *R x l l'*, *In x l* → *incl l' l* → *is_restricted R l* →
rel_midex R → *rel_midex* (*try_add_arc_one_to_many R x l'*).

Proof By induction on *l'*. Also invoke *restricted_try_add_arc_one_to_many*, *restricted_midex_clos_trans_midex* with *l*, and a similar middle-exclusion preservation property for *try_add_arc*. □

Next, a step towards totality.

Lemma *try_add_arc_one_to_many_trichotomy* : *eq_midex* → ∀ *R x y l l'*,
In y l' → *In x l* → *incl l' l* → *is_restricted R l* → *rel_midex R* →
trichotomy (*try_add_arc_one_to_many R x l'*) *x y*.

Proof By induction on *l'*. For the inductive step, invoke *trichotomy_preserved*, case split on *y* being the head of *l'* or *y* occurring in the tail of *l'*. Also refer to a similar trichotomy property for *try_add_arc*. □

6.4 Try Add Arc (Many to Many)

The function *try_add_arc_many_to_many* requires a relation and two lists. Then, using *try_add_arc_one_to_many*, it recursively tries to safely add all arcs starting in first list argument and ending in the second one.

Fixpoint *try_add_arc_many_to_many R l' l {struct l'} : A → A → Prop :=*
match l' with
— nil ⇒ R
— x::l" ⇒ try_add_arc_one_to_many (try_add_arc_many_to_many R l" l) x l
end.

The following three results proved by induction on the list *l'* state arc, restriction, and decidability preservation properties of *try_add_arc_many_to_many*. For the inductive case of the first lemma, call *l"* the tail of *l'*, apply the transitivity of *sub_rel* with (*try_add_arc_many_to_many R l" l*), and invoke lemma *sub_rel_try_add_arc_one_to_many*. Use *restricted_try_add_arc_one_to_many* for the second lemma. For the third one invoke*try_add_arc_one_to_many_dec* and *restricted_try_add_arc_many_to_many*. Middle-exclusion preservation is also correct, although not formally stated here.

Lemma *sub_rel_try_add_arc_many_to_many : ∀ R l l',*
sub_rel R (try_add_arc_many_to_many R l' l).

Lemma *restricted_try_add_arc_many_to_many : ∀ R l l', incl l' l →*
is_restricted R l → is_restricted (try_add_arc_many_to_many R l' l) l.

Lemma *try_add_arc_many_to_many_dec : → ∀ R l l', incl l' l →*
is_restricted R l → rel_dec R → rel_dec (try_add_arc_many_to_many R l' l).

The next two results state a trichotomy property and that the function *try_add_arc_many_to_many* does not create any cycle.

Lemma *try_add_arc_many_to_many_trichotomy : eq_midex → ∀ R l x y l',*
incl l' l → In y l → In x l' → restricted R l → rel_midex R →
trichotomy (try_add_arc_many_to_many R l' l) x y.

Proof By induction on *l'*. Start the inductive step by case splitting on *x* being the head of *l'* or occurring in its tail *l"*. Conclude the first case by *try_add_arc_one_to_many_trichotomy*, *try_add_arc_many_to_many_midex*, and *restricted_try_add_arc_many_to_many*. Use *trichotomy_preserved*, the induction hypothesis, and *sub_rel_try_add_arc_one_to_many* for the second case. □

Lemma *try_add_arc_many_to_many_irrefl : eq_midex → ∀ R l l',*
incl l' l → is_restricted R l → transitive A R →
irreflexive R → irreflexive (try_add_arc_many_to_many R l' l).

Proof By induction on *l'*. For the inductive step, first prove a similar irreflexivity property for *try_add_arc_one_to_many* by induction on lists and *try_add_arc_irrefl*. Then invoke *restricted_try_add_arc_many_to_many*. Both this proof and the one for *try_add_arc_one_to_many* also require transitivity of the transitive closure and an additional case splitting on *l'* being *nil* or not. □

6.5 Linear Extension/Topological Sort Function

Consider the restriction of a given relation to a given list. The following function tries to add all arcs both starting and ending in that list to that restriction while preserving acyclicity.

Definition $LETS\ R\ l : A \to A \to Prop :=$
$try\_add\_arc\_many\_to\_many\ (clos\_trans\ (restriction\ R\ l))\ l\ l$.

The next three lemmas are proved by *sub_rel_try_add_arc_many_to_many*, *transitive_clos_trans*, and *restricted_try_add_arc_many_to_many* respectively.

Lemma $LETS\_sub\_rel : \forall\ R\ l,$
$sub\_rel\ (clos\_trans\ (restriction\ R\ l))\ (LETS\ R\ l)$.

Lemma $LETS\_transitive : \forall\ R\ l,\ transitive\ (LETS\ R\ l)$.

Lemma $LETS\_restricted : \forall\ R\ l,\ is\_restricted\ (LETS\ R\ l)\ l$.

Under middle-excluding equality, the finite restriction of R to l has no cycle *iff* $LETS\ R\ l$ is irreflexive. Prove left to right by *try_add_arc_many_to_many_irrefl*, and right to left by *irreflexive_preserved* and *LETS_sub_rel*.

Lemma $LETS\_irrefl : eq\_midex \to \forall\ R\ l,$
$(irreflexive\ (clos\_trans\ (restriction\ R\ l)) \leftrightarrow irreflexive\ (LETS\ R\ l))$.

If R and equality on A are middle-excluding then $LETS\ R\ l$ is total on l. By *R_midex_clos_trans_restriction_midex* (from the first main result) and *try_add_arc_many_to_many_trichotomy*.

Lemma $LETS\_total : eq\_midex \to \forall\ R\ l,\ rel\_midex\ R \to total\ (LETS\ R\ l)\ l$.

The next two lemmas show that if R and equality on A are middle-excluding (resp. decidable) then so is $LETS\ R\ l$: by *try_add_arc_many_to_many_midex* (resp. *try_add_arc_many_to_many_dec*) and *R_midex_clos_trans_restriction_midex* (resp. *R_dec_clos_trans_restriction_dec*).

Lemma $LETS\_midex : eq\_midex \to \forall\ R\ l,$
$rel\_midex\ R \to rel\_midex\ (LETS\ R\ l)$.

Lemma $LETS\_dec : eq\_dec \to \forall\ R,\ rel\_dec\ R \to \forall\ l,\ rel\_dec\ (LETS\ R\ l)$.

6.6 Linear Extension

Traditionally, a linear extension of a partial order is a total order including the partial order. Below, a linear extension (over a list) of a binary relation is a strict total order (over the list) that is bigger than the original relation (restricted to the list).

Definition $linear\_extension\ R\ l\ R' := is\_restricted\ R'\ l \wedge$
$sub\_rel\ (restriction\ R\ l)\ R' \wedge transitive\ A\ R' \wedge irreflexive\ R' \wedge total\ R'\ l$.

The next two lemmas say that a relation "locally" contained in some acyclic relation is "globally" acyclic and that if for any list over A there is a middle-excluding total order over that list, then equality is middle-excluding on A.

Lemma *local_global_acyclic* : $\forall$ R,
($\forall$ l, $\exists$ R', *sub_rel* (*restriction* R l) R' $\wedge$ *transitive* R' $\wedge$ *irreflexive* R') $\rightarrow$
irreflexive (*clos_trans* R).

Proof Let R be a relation over A. Assume that any finite restriction of R is included in some strict partial order. Let x be in A such that *clos_trans* R x x. Then derive *False* as follows. Invoke *clos_trans_path* and get a path. It is still a path for the restriction of R to the path itself (the path is a list seen as a subset of A). Use *path_clos_trans*, then the main assumption, *transitive_sub_rel_clos_trans*, and the monotonicity of *clos_trans* with respect to *sub_rel*. □

Lemma *total_order_eq_midex* :
($\forall$ l, $\exists$ R, *transitive* R $\wedge$ *irreflexive* R $\wedge$ *total* R l $\wedge$ *rel_midex* R) $\rightarrow$ *eq_midex*.

Proof Assume the left conjunct, let x and y be in A, use the assumption with x::y::*nil*, get a relation, and double case split on x and y being related either way. □

Consider a middle-excluding relation on A. It is acyclic and equality is middle-excluding on A *iff* for any list over A there exists, on the given list, a decidable strict total order containing the original relation.

Theorem *linearly_extendable* : $\forall$ R, *rel_midex* R $\rightarrow$
(*eq_midex* $\wedge$ *irreflexive* (*clos_trans* R) $\leftrightarrow$
$\forall$ l, $\exists$ R', *linear_extension* R l R' $\wedge$ *rel_midex* R').

Proof Left to right: by the relevant lemmas of subsection 6.5, (*LETS* R l) is a witness for the existential quantifier. Right to left by *local_global_acyclic* and *total_order_eq_midex*. □

6.7 Topological Sorting

In this subsection, excluded-middle results of subsection 6.6 are translated into decidability results and augmented: as there is only one concept of linear extension in subsection 6.6, this section presents three slightly different concepts of topological sort. Instead of the equivalence of theorem *linearly_extendable*, those three definitions yield a quadruple equivalence.

From now on a decidable relation may be represented by a function to booleans instead of a function to *Prop* satisfying the definition *rel_dec*. However, those two representations are "equivalent" thanks to lemmas *rel_dec_bool* and *bool_rel_dec* in subsection 2.

In this article, a given relation over A is said to be non-uniformly (topologically) sortable if the restriction of the relation to any list has a decidable linear extension.

Definition *non_uni_topo_sortable* $R :=$
$\forall\ l, \exists\ R' : A \to A \to bool$, *linear_extension* $R\ l$ (*fun* $x\ y \Rightarrow R'\ x\ y$=*true*).

In the definition above, *R'* represents a decidable binary relation that intends to be a linear extension of R over the list l. But *R'* has type $A \to A \to bool$ so it cannot be used with the predicate *linear_extension* $R\ l$ that waits for an object of type $A \to A \to Prop$, which is the usual type for representing binary relations in Coq. The function *fun* $x\ y \Rightarrow (R'\ x\ y)$=*true* above is the translation of *R'* in the suitable type/representation. It waits for two elements x and y in A and returns the proposition *R'* $x\ y$=*true*, in *Prop*.

In this article, a given relation over A is said to be uniformly sortable if there exists a computable function waiting for a list over A and producing, over the list argument, a (decidable) linear extension of the original relation.

Definition *uni_topo_sortable* $R := \{F : list\ A \to A \to A \to bool \mid$
$\forall\ l$, *linear_extension* $R\ l$ (*fun* $x\ y \Rightarrow (F\ l\ x\ y)$=*true*)$\}$.

The third definition of topological sort requires the concept of *asymmetry*, which is now informally introduced; from an algorithmic viewpoint: given a way of representing binary relations, different objects may represent the same binary relation; from a logical viewpoint: two binary relations different in intension, *i.e.* their definitions intend different things, may still coincide, *i.e.* may be logically equivalent. In an arbitrary topological sort algorithm, the returned linear extension may depend on which object has been chosen to represent the original binary relation. For example, given a two-element set, a topological sort algorithm that is input the empty relation on the given set may produce the two possible linear extensions depending on the order in which the two elements constituting the set are given. This remark leads to the following definition.

Definition *asym* $R\ G := \forall\ x\ y : A$,
$x \neq y \to \neg R\ x\ y \to \neg R\ y\ x \to \neg(G\ (x::y::nil)\ x\ y \wedge G\ (y::x::nil)\ x\ y)$.

Next comes the definition of asymmetry for a topological sort of a binary relation. The syntax *let variable*:= *formula in formula'* avoids writing *formula* several times in *formula'*.

Definition *asym_topo_sortable* $R := \{F : list\ A \to A \to A \to bool \mid$
let G:= (*fun* $l\ x\ y \Rightarrow F\ l\ x\ y$=*true*) *in*
asym $R\ G \wedge \forall\ l$, *linear_extension* $R\ l\ (G\ l)\}$.

Given a binary relation R over A, the remainder of this subsection proves that the four following assertions are equivalent:

1. Equality on A is decidable, and R is decidable and acyclic.
2. R is middle-excluding and asymmetrically sortable.
3. R is decidable and uniformly sortable.
4. Equality on A is decidable, and R is decidable and non-uniformly sortable.

The following lemma says that if there exists a computable function waiting for a list over A and producing a (decidable) strict total order over A, then equality on A is decidable. The proof is similar to the one for *total_order_eq_midex*.

Lemma *total_order_eq_dec* :
$\{F : list\ A \to A \to A \to bool$ — $\forall\ l,\ let\ G := fun\ x\ y \Rightarrow F\ l\ x\ y{=}true\ in$
$transitive\ A\ G \wedge irreflexive\ G \wedge total\ G\ l\} \to eq\_dec\ A$.

Next lemma shows that *LETS* yields asymmetric topological sort.

Lemma *LETS_asym* : $\forall\ R,\ asym\ R\ (LETS\ R)$.

Proof Assume all possible premises, especially let x and y be in A. As a preliminary: the hypotheses involve one relation image of *restriction* and four relations images of *try_add_arc*. Prove that all of them are restricted to $x{::}y{::}nil$. Then perform a few cases splittings and apply *clos_trans_restricted_pair* seven times. □

The quadruple equivalence claimed above follows from *rel_dec_midex* and the six theorems below. The proofs are rather similar to the middle-excluding case in subsection 6.6. The first theorem proves $1 \to 2$ by the relevant lemmas of subsection 6.5 and *LETS* producing a witness for the computational existence. The second (straightforward) and the third show $2 \to 3$. The fourth (straightforward) and the fifth, proved by *total_order_eq_dec*, yield $3 \to 4$. The last shows $4 \to 1$ by invoking *local_global_acyclic*.

Theorem *possible_asym_topo_sorting* : $\forall\ R$,
$eq\_dec\ A \to rel\_dec\ R \to irreflexive\ (clos\_trans\ A\ R) \to asym\_topo\_sortable\ R$.

Theorem *asym_topo_sortable_uni_topo_sortable* : $\forall\ R$,
$asym\_topo\_sortable\ R \to uni\_topo\_sortable\ R$.

Theorem *asym_topo_sortable_rel_dec* : $\forall\ R$,
$rel\_midex\ R \to asym\_topo\_sortable\ R \to rel\_dec\ R$.

Proof First notice that R is acyclic by *local_global_acyclic* and that equality on A is decidable by *total_order_eq_dec*. Then let x and y by in A. By decidable equality, case split on x and y being equal. If they are equal then they are not related by acyclicity. Now consider that they are distinct. Thanks to the assumption, get TS an asymmetric topological sort of R. Case split on x and y being related by TS $(x{::}y{::}nil)$. If they are not then they cannot be related by R by subrelation property. If they are related then case split again on x and y being related by TS $(y{::}x{::}nil)$. If they are not then they cannot be related by R by subrelation property. If they are then they also are by R by the asymmetry property. □

Theorem *uni_topo_sortable_non_uni_topo_sortable* : $\forall\ R$,
$uni\_topo\_sortable\ R \to non\_uni\_topo\_sortable\ R$.

Theorem *rel_dec_uni_topo_sortable_eq_dec* : $\forall\ R$,
$rel\_dec\ R \to uni\_topo\_sortable\ R \to eq\_dec\ A$.

Theorem *rel_dec_non_uni_topo_sortable_acyclic* : $\forall\ R$,
$rel\_dec\ R \to non\_uni\_topo\_sortable\ R \to irreflexive\ (clos\_trans\ A\ R)$.

7 Conclusion

This paper has given a detailed account on a few facts related to linear extensions of acyclic binary relations. The discussion is based on a formal proof developed with the proof assistant Coq. The three main results are stated again below. First, a binary relation over a set with decidable/middle-excluding equality is decidable/middle-excluding *iff* transitive closures of its finite restrictions are also decidable/middle-excluding. That theorem is involved in the proof of the second and third main results. Second, consider a middle-excluding relation over an arbitrary domain. It is acyclic and equality on its domain is middle-excluding *iff* any of its finite restriction has a middle-excluding linear extension. Third, consider R a decidable binary relation over A. The following three propositions are equivalent:

- Equality on A is decidable and R is acyclic.
- Equality on A is decidable and R is *non-uniformly* sortable.
- R is *uniformly* sortable.

8 Acknowledgement

I thank Pierre Lescanne for his careful reading and helpful comments, as well as Guillaume Melquiond and Victor Poupet for discussions.

References

1. The Coq proof assistant, version 8.1, http://coq.inria.fr/.
2. Anonymous. Program evaluation research task. Summary report Phase 1 and 2, U.S. Government Printing Office, Washington, D.C., 1958.
3. F. Blanqui, S. Coupet-Grimal, W. Delobel, S. Hinderer, and A. Koprowski. CoLoR, a Coq Library on rewriting and termination. In *Workshop on Termination*, 2006. http://color.loria.fr/.
4. M.P. Jarnagin. Automatic machine methods of testing pert networks for consistency. Technical Memorandum K-24/60, U. S. Naval Weapons Laboratory, Dahlgren, Va, 1960.
5. A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962.
6. Donald E. Knuth. *The Art of Computer Programming*, volume 1, second edition. Addison Wesley, 1973.
7. Daniel J. Lasser. Topological ordering of a list of randomly-numbered elements of a network. *Commun. ACM*, 4(4):167–168, 1961.
8. Stéphane Le Roux. Acyclicity and finite linear extendability: a formal and constructive equivalence. Research report RR2007-14, LIP, École normale supérieure de Lyon, 2007.
9. Vaughan Pratt. Origins of the calculus of binary relations. In *Logic in Computer Science*, 1992.
10. Edward Szpilrajn. Sur l'extension de l'ordre partiel. *Fund. Math*, 1930.