Equivalence in Answer Set Programming

Mauricio Osorio[1], Juan Antonio Navarro[2], Jose Arrazola[1]

[1] Universidad de las Américas, CENTIA
Sta. Catarina Mártir, Cholula, Puebla
72820 México
josorio@mail.udlap.mx
[2] Universidad de las Américas
Dep. de Matemáticas
Sta Catarina Mártir, Cholula, Puebla
72820 México

Abstract. We study the problem of determining if two programs are strongly equivalent in ASP (Answer set programming). We show how to reduce some subcases of this problem to provability in intuitionistic/classical logic. We also present a linear reduction of free programs (general programs extended to possibly include negated atoms in their heads) into standard general programs.

1 Introduction

Answer Set Programming (ASP) (Stable Logic Programming [4] or A-Prolog) is the realization of much theoretical work on Nonmonotonic Reasoning and AI applications of Logic Programming in the last 12 years. The main syntactic restriction needed in this paradigm is to eliminate function symbols from the language.

One important issue is to know when two programs are "equivalent" with respect to ASP. We consider a definition for "equivalence" that is given in [6]. We say that P_1 and P_2 are strongly equivalent iff for every program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets. If two programs are strongly equivalent we know that we can replace one by the other in any larger program without changing the declarative semantics. This is an important concept for software engineering. It is known that Jankov logic as well as the logic of here-and-there (HT) characterize the class of strongly equivalent augmented programs under STABLE [6, 3]. Since there is much research and available software on classical logic and intuitionistic logic, we present some results of strongly equivalence with respect to these logics.

Augmented logic programs allow nested expressions permitted in the bodies and heads of rules. These expressions are formed from literals using negation (intended to represent negation as failure), conjunction and disjunction that can be nested arbitrarily. Given two augmented programs P_1 and P_2, we show that there is a linear time computable algorithm that constructs a formula F (of linear size with respect to $P_1 \cup P_2$) such that P_1 is strongly equivalent to

P_2 iff F is a theorem in intuitionistic logic. Given an augmented program P and a negated atom $\neg a$, we show that P is strongly equivalent to $P \cup \{\neg a\}$ iff $P \vdash_C \neg a$, where $\vdash_C$ denotes the provability relation in classical logic. Given a disjunctive program P and an atom a, we show that there is linear time computable algorithm that constructs a formula F (of the same size as P) such that P is strongly equivalent to $P \cup \{\ a\}$ iff $F \vdash_I\ a$. The symbol $\vdash_I$ denotes the provability relation in intuitionistic logic. We also present a linear reduction of free programs into standard general programs.

Our results will contribute to a better understand for the notion of ASP in general propositional theories. In this paper we restrict attention to finite propositional theories; the semantics can be extended to theories with variables by grounding[1]. This is a standard procedure in ASP. We assume that the reader has some basic background in logic and ASP.

Our paper is structured as follows: in section 2 we present the syntax of general clauses and define several types of programs. In section 3 we present the definition of some si-logics and our results for the Jankov logic and the definition of our transformations rules for general clauses. In section 4 we present an application of our results to ASP. In section 5 we present our conclusions. Finally in section 6 we give the proofs of our results.

2 Background

A signature $\mathcal{L}$ is a finite set of elements that we call proposition symbols. If F is a set of formulas, then by $\mathcal{L}_F$ we understand it to mean the signature of F, i.e. the set of proposition symbols that occur in F. The language of propositional logic has an alphabet consisting of proposition symbols: $p_0, p_1, \ldots$

$$\text{connectives} : \vee, \wedge, \leftarrow, \bot \qquad \text{auxiliary symbols} : (\ ,\ ).$$

Where $\vee, \wedge, \leftarrow$ are 2-place connectives and $\bot$ is a 0-place connective. The proposition symbols are also called *atoms*, or *atomic propositions*. Formulas are defined as usual in logic. The formula $\neg F$ is introduced as an abbreviation of $\bot \leftarrow F$, and $F \equiv G$ as an abbreviation of $(F \to G) \wedge (G \to F)$. The formula $G \to F$ is another way of writing the formula $F \leftarrow G$. We use $\odot$ to represent any binary connective. A *literal* is either an atom a, or the negation of an atom $\neg a$. The *complement* of a literal is defined as $(\neg a)^c := a$ and $a^c := \neg a$. We also define the *size* of a formula as $\text{size}(\bot) = \text{size}(a) = \text{size}(\neg a) = \text{size}(\neg\neg a) = 1$ where a is an atom, and $\text{size}(F_1 \odot F_2) = \text{size}(F_1) + \text{size}(F_2)$ in any other case.

Given a set of formulas F, we define $\neg F = \{\neg f | f \in F\}$. Also for a finite set of formulas $F = \{f_1, \ldots, f_n\}$, we define $\bigwedge F = f_1 \wedge \cdots \wedge f_n$ and $\bigvee F = f_1 \vee \cdots \vee f_n$. If $F = \emptyset$ then we define $\bigwedge F = \bot$ and $\bigvee F = \top$, where $\top$ is $\neg\bot$. A $\wedge\vee\neg$formula is a formula constructed by using (only) the logical connectives $\{\vee, \wedge, \neg\}$ arbitrarily nested. For instance $\neg(a \wedge b) \vee (\neg c \wedge (n \vee \neg m))$ is a $\wedge \vee \neg$formula. A *simple conjunction* is a formula of the form $\bigwedge L_1 \cup \neg L_2 \cup \neg\neg L_3$, and a *simple disjunction*

[1] Without function symbols to ensure that a ground program would be finite.

is a formula of the form $\bigvee L_1 \cup \neg L_2 \cup \neg\neg L_3$ where L_1, L_2, L_3 are sets of atoms which satisfy $L_1 \cup \neg L_2 \cup \neg\neg L_3 \neq \emptyset$.

A *clause* is a formula of the form $H \leftarrow B$ where H and B are, in principle, arbitrary formulas and are called the *head* and *body* of the clause respectively. We define several types of clauses. If $H = \bot$ the clause is called a *constraint*, we may omit the symbol $\bot$ and write the clause as $\leftarrow B$. An *augmented clause* is a clause where H and B are any $\wedge\vee\neg$formulas. Note that a broader form of this class of programs is considered in [9] where they also include two kinds of negation as well as implications in the body. We only include one kind of negation that corresponds to their default negation[2].

A *free clause* is a clause of the form $\bigvee(H_p \cup \neg H_n) \leftarrow \bigwedge(B_p \cup \neg B_n)$ where H_p, H_n, B_p, B_n are, possibly empty, sets of atoms. If $H_n = \emptyset$ the clause is called a *general clause* and if, in addition, $H_p \neq \emptyset$ (i.e. it is not a constraint) the clause is called a *disjunctive clause*. For a disjunctive clause C as just denoted, we write $tr(C)$ to denote the augmented clause $\bigvee(H_p \cup \neg\neg B_n) \leftarrow \bigwedge B_p$.

A *program* is then a finite set of clauses. If all the clauses in a program are of a certain type we say the program is also of this type. For instance a set of augmented clauses is an *augmented program*, a set of free clauses is a *free program* and so on. Even thought a program P is a set of formulas, it can also be considered as a formula (i.e. the conjunction of all formulas in the program $\bigwedge P$). We will use this convention freely several times in the paper. For a disjunctive program P, we define $tr(P) := \{tr(C) : C \in P\}$.

We write $\vdash_{\mathrm{X}}$ to denote the provability relation in a logic X. We say that two formulas $\mathcal{A}$ and $\mathcal{B}$ are equivalent under a given logic X iff $\vdash_{\mathrm{X}} \mathcal{A} \equiv \mathcal{B}$, also denoted as $\mathcal{A} \equiv_{\mathrm{X}} \mathcal{B}$. Gödel observed that there are infinetely many logics located between intuitionistic logic (I) and classical logic (C) that are sometimes called superintuitionistic logics (or si-logics for short), [11]). For an axiomatic theory, as I, we also use its name to denote the set of axioms that define it. Jankov Logic (Jn) is the si-logic $\mathrm{Jn} = \mathrm{I} \cup \{\neg p \vee \neg\neg p\}$, where the axiom schema $\neg p \vee \neg\neg p$ characterizing it is also called *weak law of excluded middle*. The logic HT is the propositional si-logic $I \cup \{(\neg q \to p) \to (((p \to q) \to p) \to p)\}$[3].

3 Results on logic

We present several technical results on logic that will be applied in the next section.

Lemma 3.1. *Let T be any theory, and let F, G be a pair of equivalent formulas under any si-logic* X. *Any theory obtained from T by replacing some occurrences of F by G is equivalent to T (under* X*).*

[2] However they use *not* for our negation ($\neg$).

[3] This logic is the greatest intermediate propositional logic properly included in the classical one.

We present several results on intuitionistic and Jankov logics. We suspect that some of them could be known, however we did not find any of these results in literature.

Proposition 3.1. *Let Γ be a set of propositions and $\mathcal{A}$ any formula. $\Gamma \vdash_{\mathrm{C}} \neg\mathcal{A}$ if and only if $\Gamma \vdash_{\mathrm{I}} \neg\mathcal{A}$.*

Definition 3.1. *The set* $\mathbf{Wk}$ *of* weak formulas *is the smallest set* $\mathbf{X}$ *with the properties: If $\mathcal{A}$ is any formula then $(\neg\mathcal{A}) \in \mathbf{X}$. If $\mathcal{A}$ is any formula and $\mathcal{B} \in \mathbf{X}$ then $(\mathcal{A} \to \mathcal{B}) \in \mathbf{X}$. If $\mathcal{A}, \mathcal{B} \in \mathbf{X}$ then $(\mathcal{A} \vee \mathcal{B}), (\mathcal{A} \wedge \mathcal{B}) \in \mathbf{X}$.*

Definition 3.2. *The set* $\mathbf{P}$ *of* positive formulas *is the smallest set containing all formulas without negation connectives ($\neg$). For a given set of formulas Γ, the* positive subset *of Γ, denoted as* $\mathrm{Pos}(\Gamma)$*, is the set $\Gamma \setminus \mathbf{N}$.*
The set $\mathbf{N}$ *of* two-negated formulas *is the smallest set* $\mathbf{X}$ *with the properties:*

1. *If a is an atom then $(\neg\neg a) \in \mathbf{X}$.*
2. *If $\mathcal{A} \in \mathbf{X}$ then $(\neg\neg\mathcal{A}) \in \mathbf{X}$.*
3. *If $\mathcal{A}, \mathcal{B} \in \mathbf{X}$ then $(\mathcal{A} \wedge \mathcal{B}) \in \mathbf{X}$.*
4. *If $\mathcal{A} \in \mathbf{X}$ and $\mathcal{B}$ is any formula then $(\mathcal{A} \vee \mathcal{B}), (\mathcal{B} \vee \mathcal{A}), (\mathcal{B} \to \mathcal{A}) \in \mathbf{X}$.*

Proposition 3.2. *Let Γ be a subset of $\mathbf{P} \cup \mathbf{N}$, and let $\mathcal{A} \in \mathbf{P}$ be a positive formula. If $\Gamma \vdash_{\mathrm{I}} \mathcal{A}$ then $\mathrm{Pos}(\Gamma) \vdash_{\mathrm{I}} \mathcal{A}$.*

Jankov showed that Jn is the greatest si-logic containing the same negation free theorems as I [11]. The following result is in the same direction[4].

Lemma 3.2. *Let $\mathcal{A}$ be a positive formula and Γ be a subset of $\mathbf{P} \cup \mathbf{N}$. $\Gamma \vdash_{\mathrm{Jn}} \mathcal{A}$ iff $\Gamma \vdash_{\mathrm{I}} \mathcal{A}$.*

Proposition 3.3. *Let $\mathcal{A}$ be a weak formula, and $\{a_1, \ldots, a_n\}$ the set of all atoms occurring in $\mathcal{A}$. If $\vdash_{\mathrm{C}} \mathcal{A}$ then $(\neg a_1 \vee \neg\neg a_1), \ldots, (\neg a_n \vee \neg\neg a_n) \vdash_{\mathrm{I}} \mathcal{A}$ and therefore $\vdash_{\mathrm{Jn}} \mathcal{A}$.*

Proposition 3.4. *Let $a_1, \ldots, a_n$ be all the atoms occurring in a formula $\mathcal{A}$. Then $\mathcal{A}$ is a theorem in the* Jn *logic iff $(\neg a_1 \vee \neg\neg a_1), \ldots, (\neg a_n \vee \neg\neg a_n) \vdash_{\mathrm{I}} \mathcal{A}$.*

4 Applications to ASP

We show how we can apply our results of the last section for the stable semantics. As defined in [6] two programs P_1 and P_2 are said to be *strong equivalent* if for every program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets.

Theorem 4.1 ([3, 6]). *Let F_1 and F_2 be two augmented programs. Then $F1$ and F_2 are strongly equivalent iff F_1 and F_2 are equivalent in logic X, where X is either Jankov or HT.*

[4] However the authors of this paper do not know if the results by Jankov also include lemma 3.2

Lemma 4.1. *Let P_1 and P_2 be two augmented programs. Let $\mathcal{L}_{P_1 \cup P_2}$ be the set $\{a_1, \ldots, a_n\}$. Then P_1 is strongly equivalent to P_2 iff $(\neg a_1 \vee \neg\neg a_1), \ldots, (\neg a_n \vee \neg\neg a_n) \vdash_{\mathrm{I}} P_1 \equiv P_2$.*

Consider the following program P_1 that shows that intuitionistic logic cannot be used in theorem 4.1 (the program, as well as the observation, was given in [6]).

$$q \leftarrow \neg p. \qquad p \leftarrow \neg q. \qquad r \leftarrow p, q. \qquad s \leftarrow p. \qquad s \leftarrow q.$$

Let P_2 be $P_1 \cup \{s \leftarrow \neg r\}$. We can show, by Theorem 4.1, that they are strongly equivalent by showing that they are equivalent in Jankov logic. Note that by using our proposition 3.4 we can use intuitionistic logic to get the desired result. Furthermore, this is the case where an automatic theorem prover like *porgi*[5] to help us solve our task.

We can also use lemma 3.2 to reduce a proof of strong equivalence to a proof in intuitionistic logic without having to add the extra assumptions $(\neg a_i \vee \neg\neg a_i)$. Of course, we need the hypothesis of the lemma to hold. An interesting case is the following.

Lemma 4.2. *Given a disjunctive program P and an atom a, if $F := tr(P)$ then*

- *F is linear time computable.*
- *F is of the same size as P.*
- *P is strongly equivalent to $P \cup \{a\}$ iff $F \vdash_{\mathrm{I}} a$.*

Lemma 4.3. *Given an augmented program P and a negated atom $\neg a$, then P is strongly equivalent to $P \cup \{\neg a\}$ iff $P \vdash_{\mathrm{C}} \neg a$*

Of course instead of $\neg a$ we can use any other weak formula in previous lemma.

4.1 Simplifying a Program

We apply several results to simplify our programs. The following result corresponds to proposition 7 in [9].

Theorem 4.2 ([9]). *Any augmented program is strongly equivalent under stable to a free program.*

For the following definition we need to explain some notation. Given a formula of the form: $L_1 \odot \cdots \odot L_k \odot \neg L_{k+1} \cdots \odot \neg L_m$, we use $Lit(F)$ to denote the set $\{L_1, \ldots, L_k, \neg L_{k+1}, \ldots, \neg L_m\}$.

Definition 4.1 (Basic Transformation Rules for free programs: $\mathcal{CS}$++). *A transformation rule is a binary relation on free programs over $\mathcal{L}$. The following transformation rules are called basic. Let P be a program.*

[5] A software implemented in standard ML and available online at http://cis.ksu.edu/~allen/porgi.html

(a) *Delete a clause* $C_1 := A_1 \leftarrow B_1$ *if there is a clause* $A \leftarrow \top$ *such that* $\neg Lit(A) \subseteq Lit(B_1)$.

(b) *Suppose that there are two clauses* $C_1 := A_1 \leftarrow B_1$ *and* $C_2 := A_2 \leftarrow B_2$ *such that* $Lit(A_1) \subseteq Lit(A_2)$ *and* $Lit(B_1) \subseteq Lit(B_2)$*, then we remove* C_2.

(c) *Suppose that there is a clause of the form* $l \leftarrow \top$ *where* l *is a literal, then in all the clauses of the form* $A \leftarrow B$ *such that* $l \in Lit(B)$ *delete* l *from* B[6].

(d) *Suppose that there is a clause of the form* $l \leftarrow \top$ *where* l *is a literal, then in all the clauses of the form* $\mathcal{A} \leftarrow B$ *such that* $l^c \in \mathcal{A}$ *delete* l^c *from* A.

(e) *Suppose that* P *includes a clause that is a simple theorem in* Jn*, then we delete such clause. By definition, the only two simple theorems in* Jn *are:* $\beta \vee l \leftarrow l \wedge \alpha$*, (where* l *is a literal) and* $\beta \leftarrow a \wedge \neg a \wedge \alpha$.

Lemma 4.4 **($\mathcal{CS}$++ is closed under** Jn **Logic).** *Let* P_1 *and* P_2 *two programs related by any transformation in* $\mathcal{CS}$++*. Then* $P_1 \equiv_{Jn} P_2$.

Why is $\mathcal{CS}$++ useful in ASP? These transformations are quadratic time computable. They are generalizations of transformations given for normal programs. In the case of normal programs $\mathcal{CS}$++[7] plus two more transformations (loop and failure) are strong enough to compute the WFS semantics efficiently, see [2]. In many useful cases the stable semantics has only one model that corresponds to the WFS semantics. Moreover, sometimes these transformations can transform a program to a tight one [7]. For tight programs the stable semantics corresponds to the supported semantics. Recent research [1] has shown that, in this case, a satisfiability solver (e.g. SATO [12]) can be used to obtain stable models.

Example 4.1. Consider the following example that illustrates some reductions obtained applying $\mathcal{CS}$++ transformations to a program:

$$\begin{array}{l} \neg a \vee b \leftarrow c \wedge d \\ \neg a \leftarrow c \wedge t \\ \neg k \vee p \leftarrow p \wedge q \\ t \leftarrow \top \end{array} \quad \Longrightarrow \quad \begin{array}{l} \neg a \leftarrow c \\ t \leftarrow \top \end{array}$$

Inoue and Sakama showed [8] that every free program P can be transformed into disjunctive one P' such that P' is a conservative extension of P. We say that P' is a *conservative extension* of P to denote the fact that M is a stable model of P iff M' is a stable model of P' such that $M = M' \cap \mathcal{L}_P$. However their transformation is very expensive and involves a cubic increase in the size of programs. We show that this same goal can be achieved with a linear time computable transformation whose increase of programs size is also linear.

Theorem 4.3. *Let* P *be a free program with* $\{a_1, \ldots, a_n\}$ *atoms that appear negated* $(\neg a_i)$ *in the head of the clauses. Let* Σ *be a set of atoms such that* $|\Sigma| = n$ *and* $\Sigma \cap \mathcal{L}_P = \emptyset$*, and* f *a bijection from* $\{a_1, \ldots, a_n\}$ *to* Σ*. If* $\Delta_n =$

[6] The reader should understand that deleting say a from $a \vee \neg a$ leaves $\neg a$, while deleting $\neg a$ from $a \vee \neg a$ leaves a.

[7] Taking out $\beta \leftarrow a \vee \neg a \vee \alpha$

$\bigcup_{i=1}^{n}(\{f(a_i) \leftarrow \neg a_i,\ \ \bot \leftarrow a_i \wedge f(a_i)\})$, *then* $P_{\Delta_n} \cup \Delta_n$ *is a conservative extension of* P *(where* P_{Δ_n} *is obtained from* P *replacing each occurrence of* $\neg a_i$ *with* $f(a_i)$ *for* $i = 1, \ldots, n$*).*

Now $P_{\Delta_n} \cup \Delta_n$ is a general program since it contains no negations in the head of clauses. It is well known that general programs are equivalent to disjunctive programs under a simple translation. Namely, replace every constraint clause $\bot \leftarrow A$ by $p \leftarrow A \wedge \neg p$, where p is an atom not in $\mathcal{L}_P \cup \{f(a_1), \ldots, f(a_n)\}$.

Notice that Theorem 4.2 together with Theorem 4.3 stand an equivalence between programs that shows augmented programs are not more expressive than disjunctive ones with respect to the STABLE semantics.

5 Conclusions

We presented several results that relate provability in Jankov logic with provability in intuitionistic and classical logic. Based on our results we propose several ideas that help to decide if two theories are strongly equivalent. We also present a simple transformation from free programs to disjunctive programs such that the stable semantics is preserved. Our results are given for propositional theories but they can easily be generalized to universally quantified theories without functional symbols. We are developing a software tool that integrates our results that would help a programmer to improve his code. We have obtained good empirical results so far, and we plan to report them in a complete paper.

References

1. Yu. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, pages ?–? 2000.
2. Stefan Brass, Jürgen Dix, Burkhard Freitag, and Ulrich Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, to appear, 2001.
3. D.H.J. de Jongh and A. Hendriks. Characterization of strongly equivalent logic programs in intermediate logics. *http://turing.wins.uva.nl/ lhendrik/*, 2001.
4. Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
5. Jean-Yves Girard. *Proofs and types.* Cambridge, 1889.
6. Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, to appear.
7. Mauricio Osorio, J. C. Nieves, and Chris Giannella. Useful transformation in answer set programming. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 146–152. Stanford, USA, 2001.
8. Ch. Sakama and K. Inoue. Negation as Failure in the Head. *Journal of Logic Programming*, 35(1):39–78, 1998.

9. L. R. Tang V. Lifschitz and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
10. Dirk van Dalen. *Logic and Structure.* Springer, Berlin, second edition, 1980.
11. M. Zakharyaschev, F. Wolter, and A. Chagrov. Advanced modal logic. In D.M. Gabbay, editor, *To appear as a chapter in the 2nd edition of the Handbook of Philosophical Logic*, pages ?? – ?? Kluwer Academic Publishers, 2001.
12. H. Zhang. Sato: A decision procedure for propositional logic. *Association for Automated Reasoning Newsletter*, 22:1–3, March 1993.

6 Appendix: Proofs

Proof of Proposition 3.1. Let $\Gamma = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$. The proof is by induction on the number n of elements in Γ. The case $n = 0$ is discussed as an exercise for intuitionistic logic in [10] (p. 192). Assume now that the proposition holds for $n - 1$. If $\Gamma \vdash_{\mathrm{C}} \neg\mathcal{A}$ then by Deduction Theorem $\Gamma' \vdash_{\mathrm{C}} \mathcal{B}_n \to \neg\mathcal{A}$, where $\Gamma' = \{\mathcal{B}_1, \ldots, \mathcal{B}_{n-1}\}$. Then, since $\mathcal{A} \equiv_{\mathrm{C}} \neg\neg\mathcal{A}$, $\Gamma' \vdash_{\mathrm{C}} \neg\neg(\mathcal{B}_n \to \neg\mathcal{A})$, and by the inductive hypothesis $\Gamma' \vdash_{\mathrm{I}} \neg\neg(\mathcal{B}_n \to \neg\mathcal{A})$. Now it is not difficult to prove $\Gamma' \vdash_{\mathrm{I}} \mathcal{B}_n \to \neg\mathcal{A}$. Using again Deduction Theorem (backwards) $\Gamma \vdash_{\mathrm{I}} \neg\mathcal{A}$. The converse is obvious.
Proof of Proposition 3.2. The proof is by induction on the length of the deduction for $\Gamma \vdash_{\mathrm{I}} \mathcal{A}$ using the sequent calculus defined in [5].

If $\Gamma \vdash_{\mathrm{I}} \mathcal{A}$ is an axiom, the base of induction, then $\Gamma = \{\mathcal{A}\}$ and the proposition is trivial.

If the last rule in the deduction of $\Gamma \vdash_{\mathrm{I}} \mathcal{A}$ is a left structural rule $\mathcal{L}\mathsf{X}$, $\mathcal{L}\mathsf{W}$, $\mathcal{L}\mathsf{C}$ or one of the logical rules $\mathcal{L}i\wedge$, $\mathcal{R}\wedge$, $\mathcal{L}\vee$, $\mathcal{R}i\vee$, $\mathcal{L}\!\to$, $\mathcal{R}\!\to$, the inductive hypothesis can be applied to the previous sequent(s) in the deduction. And since taking positive subsets in the premises of this rules preserves "provability" it is possible to prove $\mathrm{Pos}(\Gamma) \vdash_{\mathrm{I}} \mathcal{A}$.

For example, if the last rule is

$$\frac{\Gamma_1, \mathcal{B} \vdash_{\mathrm{I}} \mathcal{A} \qquad \Gamma_2, \mathcal{C} \vdash_{\mathrm{I}} \mathcal{A}}{\Gamma_1, \Gamma_2, \mathcal{B} \vee \mathcal{C} \vdash_{\mathrm{I}} \mathcal{A}}\ \mathcal{L}\vee$$

where $\Gamma = \Gamma_1, \Gamma_2, \mathcal{B} \vee \mathcal{C}$.

If $\mathcal{B} \vee \mathcal{C} \in \mathbf{P}$ then also $\mathcal{B}$ and $\mathcal{A}$ are in $\mathbf{P}$, so by the inductive hypothesis we have proofs for $\mathrm{Pos}(\Gamma_1, \mathcal{B}) \vdash_{\mathrm{I}} \mathcal{A}$ and $\mathrm{Pos}(\Gamma_2, \mathcal{C}) \vdash_{\mathrm{I}} \mathcal{A}$. Using again rule $\mathcal{L}\vee$ now we prove

$$\frac{\mathrm{Pos}(\Gamma_1), \mathcal{B} \vdash_{\mathrm{I}} \mathcal{A} \qquad \mathrm{Pos}(\Gamma_2), \mathcal{C} \vdash_{\mathrm{I}} \mathcal{A}}{\mathrm{Pos}(\Gamma_1, \Gamma_2), \mathcal{B} \vee \mathcal{C} \vdash_{\mathrm{I}} \mathcal{A}}$$

which finally leads to $\mathrm{Pos}(\Gamma) \vdash_{\mathrm{I}} \mathcal{A}$.

If $\mathcal{B} \vee \mathcal{C} \in \mathbf{N}$ then either $\mathcal{B}$ or $\mathcal{C}$ are in $\mathbf{N}$. Suppose that $\mathcal{B} \in \mathbf{N}$, the other case is analogous. Applying the inductive hypothesis on $\Gamma_1, \mathcal{B} \vdash_{\mathrm{I}} \mathcal{A}$ we have a proof for $\mathrm{Pos}(\Gamma_1) \vdash_{\mathrm{I}} \mathcal{A}$. Now, using the weakening rule $\mathcal{L}\mathsf{W}$, it is possible to prove $\mathrm{Pos}(\Gamma_1), \mathrm{Pos}(\Gamma_2) \vdash_{\mathrm{I}} \mathcal{A}$ which is equivalent to $\mathrm{Pos}(\Gamma) \vdash_{\mathrm{I}} \mathcal{A}$.

The last rule cannot be a negation rule since $\mathcal{L}\neg$ will need $\mathcal{A}$ to be empty, and $\mathcal{R}\neg$ implies $\mathcal{A} \notin \mathbf{P}$ contradicting the hypothesis of the proposition. The last rule is also not $\mathcal{R}$W since it will assert $\Gamma \vdash_{\mathrm{I}}$ or, equivalently, $\vdash_{\mathrm{I}} \neg(\bigwedge \Gamma)$. But $\vdash_{\mathrm{C}} \neg(\bigwedge \Gamma)$ is not even provable since there is an interpretation, namely $I_{\top}$, which evaluates the formula to false.

Proof of Lemma 3.2. Proposition 3.2 shows that Jankov's axiom schema $\neg\mathcal{A} \vee \neg\neg\mathcal{A}$, since it is in $\mathbf{N}$, is not needed to prove $\mathcal{A}$ and allows a pure intuitionistic deduction for it.

Proof of Proposition 3.3. We first prove ($\mathbf{A}$): Let $\mathcal{A}$ be a weak formula and let $b_1, \ldots, b_n$ be the atoms that occur in $\mathcal{A}$. For a given (classical) interpretation I, let b_i' and $\mathcal{A}'$ be defined as

$$b_i' = \begin{cases} \neg\neg b_i & I(b_i) = 1 \\ \neg b_i & I(b_i) = 0 \end{cases}, \qquad \mathcal{A}' = \begin{cases} \mathcal{A} & I(\mathcal{A}) = 1 \\ \neg\mathcal{A} & I(\mathcal{A}) = 0 \end{cases}.$$

Then $b_1', \ldots, b_n' \vdash_{\mathrm{I}} \mathcal{A}'$.

The proof is by induction on the number k of connectives in $\mathcal{A}$.

For $k = 1$ (there is always at least one $\neg$ connective) we have $\mathcal{A}$ is $\neg b_1$, then the lemma reduces to either $\neg b_1 \vdash_{\mathrm{I}} \neg b_1$ or $\neg\neg b_1 \vdash_{\mathrm{I}} \neg\neg b_1$. Assume now that the lemma holds for all $j < k$. To simplify writing let $\Gamma = \{b_1', \ldots, b_n'\}$.

Case 0. $I(\mathcal{A}) = 0$. Then $\mathcal{A}' = \neg\mathcal{A}$ and, by Proposition 3.1, $\Gamma \vdash_{\mathrm{I}} \neg\mathcal{A}$ is provable since it is in C. We have left to prove only the cases where $I(\mathcal{A}) = 1$.

Case 1. $\mathcal{A}$ is $\neg\mathcal{B}$. $I(\mathcal{B}) = 0$, hence $I(\mathcal{A}) = 1$. We have to prove $\Gamma \vdash_{\mathrm{I}} \neg\mathcal{B}$ which, by Proposition 3.1, is provable since it is in C.

Case 2. $\mathcal{A}$ is $\mathcal{B} \to \mathcal{C}$.

Case 2a. $I(\mathcal{B}) = 0$, hence $I(\mathcal{A}) = 1$. We have, again by Proposition 3.1, $\Gamma \vdash_{\mathrm{I}} \neg\mathcal{B}$. Since $\vdash_{\mathrm{I}} \neg\mathcal{B} \to (\mathcal{B} \to \mathcal{C})$ it follows $\Gamma \vdash_{\mathrm{I}} \mathcal{B} \to \mathcal{C}$.

Case 2b. $I(\mathcal{C}) = 1$, hence $I(\mathcal{A}) = 1$. Since $\mathcal{C}$ is in $\mathbf{Wk}$ we have, by inductive hypothesis, $\Gamma \vdash_{\mathrm{I}} \mathcal{C}$. Since $\vdash_{\mathrm{I}} \mathcal{C} \to (\mathcal{B} \to \mathcal{C})$ it follows $\Gamma \vdash_{\mathrm{I}} \mathcal{B} \to \mathcal{C}$.

Case 3. $\mathcal{A}$ is $\mathcal{B} \wedge \mathcal{C}$. $I(\mathcal{B}) = 1$ and $I(\mathcal{C}) = 1$, hence $I(\mathcal{A}) = 1$. By inductive hypothesis $\Gamma \vdash_{\mathrm{I}} \mathcal{B}$ and $\Gamma \vdash_{\mathrm{I}} \mathcal{C}$. It follows $\Gamma \vdash_{\mathrm{I}} \mathcal{B} \wedge \mathcal{C}$.

Case 4. $\mathcal{A}$ is $\mathcal{B} \vee \mathcal{C}$. $I(\mathcal{B}) = 1$ (or $I(\mathcal{C}) = 1$), hence $I(\mathcal{A}) = 1$. By inductive hypothesis $\Gamma \vdash_{\mathrm{I}} \mathcal{B}$ (or $\mathcal{C}$). Either case it follows $\Gamma \vdash_{\mathrm{I}} \mathcal{B} \vee \mathcal{C}$.

Notice that induction was used only in cases 2b, 3 and 4.

We return to our main proof. Let $b_1, \ldots, b_n$ be the letters that occur in $\mathcal{A}$. Since for any interpretation $I(\mathcal{A}) = 1$ ($\mathcal{A}$ is a tautology) we have, by $\mathbf{A}$, $b_1', \ldots, b_n' \vdash_{\mathrm{I}} \mathcal{A}$. If $I(b_n)$ is set to 1, $b_1', \ldots, b_{n-1}', \neg\neg b_n \vdash_{\mathrm{I}} \mathcal{A}$, and, if $I(b_n)$ is set to 0, $b_1', \ldots, b_{n-1}', \neg b_n \vdash_{\mathrm{I}} \mathcal{A}$. So, by the Deduction Theorem, $b_1', \ldots, b_{n-1}' \vdash_{\mathrm{I}} \neg\neg b_n \to \mathcal{A}$ and $b_1', \ldots, b_{n-1}' \vdash_{\mathrm{I}} \neg b_n \to \mathcal{A}$. Then it follows $b_1', \ldots, b_{n-1}' \vdash_{\mathrm{I}} (\neg b_n \vee \neg\neg b_n) \to \mathcal{A}$ or, equivalently, $b_1', \ldots, b_{n-1}', \neg b_n \vee \neg\neg b_n \vdash_{\mathrm{I}} \mathcal{A}$. Similarly, $I(b_{n-1})$ may be set to 1 or 0, apply Deduction Theorem and introduce hypothesis $\neg b_{n-1} \vee \neg\neg b_{n-1}$ to eliminate b_{n-1}. After n such steps, we finally obtain $\neg b_1 \vee \neg\neg b_1, \ldots, \neg b_n \vee \neg\neg b_n \vdash_{\mathrm{I}} \mathcal{A}$.

Proof of Proposition 3.4. Notice that Jankov's axiom schema $\neg\mathcal{A} \vee \neg\neg\mathcal{A}$ is a weak formula and by Proposition 3.3 any instance of it can be proved in intuitionistic logic using only the specific axioms $\neg a_i \vee \neg\neg a_i$ as hypothesis. The converse is trivial.

Proof of Lemma 4.1. Direct by theorem 4.1 and proposition 3.4.

Proof of Lemma 4.2. It suffices to see that $tr(P)$ is a subset of $\mathbf{P} \cup \mathbf{N}$.
Proof of Lemma 4.3. Direct consequence of proposition 3.3 and theorem 4.1.
Proof of Lemma 4.4. Let P_1 and P_2 be two programs related by any transfirmation in $\mathcal{CS}$++ but **(e)**. Then it is easy to check that $P_1 \equiv_{\mathrm{I}} P_2$ (note that we used intuitionistic logic). The case **(e)** is direct.
Proof of Theorem 4.3. For given programs P_1 and P_2 the fact that M is a stable model of P_1 if and only if M is a stable model of P_2 will be noted as $P_1 \Leftrightarrow_{\mathrm{M}} P_2$.

First, we prove the following **(B)**: Let P be a program, a and x atoms such than $a \in \mathcal{L}_{\mathcal{P}}$ and $x \notin \mathcal{L}_{\mathcal{P}}$. The function π defined, for a given set of atoms S, as

$$\pi(S) = \begin{cases} S & a \in S \\ S \cup \{x\} & a \notin S \text{ and } x \notin S \\ S \setminus \{x\} & a \notin S \text{ and } x \in S \end{cases} .$$

satisfies that M is a stable model of P if and only if $\pi(M)$ is a stable model of $P \cup \{x \leftarrow \neg a\}$[8]. Moreover $\{a, x\} \not\subset \pi(M)$.

Here is the proof of this lemma. First notice, since $x \notin \mathcal{L}_P$, if M is a stable model of P then $x \notin M$ or, conversely, if $x \in M$ then M cannot be a stable model of P. The proof is separated in several cases.

Case 1. Suppose $a \in M$, hence $(P \cup \{x \leftarrow \neg a\})^M = P^M$ and $\pi(M) = M$. Then the following equation holds

$$P \Leftrightarrow_{\mathrm{M}} P^M = (P \cup \{x \leftarrow \neg a\})^M \Leftrightarrow_{\mathrm{M}} P \cup \{x \leftarrow \neg a\} .$$

Equation above shows, when $a \in M$, M is a stable model of P iff $M = \pi(M)$ is a stable model of $P \cup \{x \leftarrow \neg a\}$. It also shows if $\{a, x\} \subset M$, and hence M is not a stable model of P, that $\pi(M) = M$ is also not a stable model of $P \cup \{x \leftarrow \neg a\}$. In other words, if M is a stable model of $P \cup \{x \leftarrow \neg a\}$ then $\{a, x\} \not\subset M$.

Case 2. Finally suppose $a \notin M$ and $x \notin M$, hence $\pi(M) = M \cup \{x\}$. Again $P \Leftrightarrow_{\mathrm{M}} P^M$ and, since $x \notin \mathcal{L}_P$, M is a stable model of P^M iff $M \cup \{x\} = \pi(M)$ is a stable model of $P^M \cup \{x\}$. Now

$$P^M \cup \{x\} = (P \cup \{x \leftarrow \neg a\})^M = (P \cup \{x \leftarrow \neg a\})^{\pi(M)} ,$$

and $(P \cup \{x \leftarrow \neg a\})^{\pi(M)} \Leftrightarrow_{\pi(\mathrm{M})} P \cup \{x \leftarrow \neg a\}$. Following bi-implications it is stated M is a stable model of P iff $\pi(M)$ is a stable model of $P \cup \{x \leftarrow \neg a\}$.

Case 3. Suppose now $a \notin M$ and $x \in M$, hence $(P \cup \{x \leftarrow \neg a\})^M = P^M \cup \{x\}$. Here notice if M' is a stable model of $P^M \cup \{x\}$ then $x \in M'$. Thus M is not a stable model of P (since $x \in M$) nor $\pi(M) = M \setminus \{x\}$ is a stable model of $P^M \cup \{x\}$ (since $x \notin M \setminus \{x\}$). In other words, if M' is a stable model of $P \cup \{x \leftarrow \neg a\}$ then $M' \cap \{a, x\} \neq \emptyset$.

We return to prove our theorem. First it is possible to show $P \cup \Delta_n$ is a conservative extension of P. The proof is by induction over n, note if $n = 0$ there is nothing to prove. Suppose $P \cup \Delta_{n-1}$ is a conservative extension of P now **(B)** justifies the inductive step. It is not hard to show, using a straightforward induction, that $P \cup \Delta_n \equiv_{\mathrm{I}} P_{\Delta_n} \cup \Delta_n$, and therefore they are strongly equivalent.

[8] Notice that $\pi(\pi(S)) = S$.